WP1-D2

Report on querying the combined metamodel

Project title:	Marrying Ontology and Software Technology
Project acronym:	MOST
Project number:	ICT-2008-216691
Project instrument:	EU FP7 STREP
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	ICT216691/UoKL/WP1-D2/D/PU/b1
Responsible editors:	Yuting Zhao, Jeff. Z. Pan, Nophadol Jekjantuk, Fernando Silva Parreiras, Gerd Gröner, Tobias Walter
Reviewers:	Krzysztof Miksa, Tirdad Rahmani
Contributing participants:	UNIABDN, UoKL, CMR, SAP
Contributing workpackages:	WP1, WP3, WP5, WP6
Contractual date of deliverable:	31 January 2009
Actual submission date:	31 January 2009

Abstract
The MOST artefact integration technology will allow for managing software development, ontology-based validation and knowledge management in an integrated platform. After providing a unified view of metamodels and addressing the integration of modelling languages in MOST Deliverable D1.1, this deliverable describes a querying solution to support developers in querying and transforming integrated models. We elicit the requirements for such a solution, analyse existing approaches, describe our solution and validate it according to the requirements

Keyword List
software modeling, ontologies, integrated modeling, query language

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Seventh Framework Programme.

Report on querying the combined metamodel

Nophadol Jekjantuk[1], Jeff. Z. Pan[1], Yuting Zhao[1], Fernando Silva Parreiras[2], Gerd Gröner[2], Tobias Walter[2]

[1]Department of Computing Science, University of Aberdeen, United Kingdom
Email: {n.jekjantuk, jeff.z.pan}@abdn.ac.uk, yuting.zhao@gmail.com,
[2] ISWeb — Information Systems and Semantic Web
Institute for Computer Science, University of Koblenz-Landau
Email: {parreiras, groener, walter}@uni-koblenz.de

31 January 2009

Abstract
The MOST artefact integration technology will allow for managing software development, ontology-based validation and knowledge management in an integrated platform. After providing a unified view of metamodels and addressing the integration of modelling languages in MOST Deliverable D1.1, this deliverable describes a querying solution to support developers in querying and transforming integrated models. We elicit the requirements for such a solution, analyse existing approaches, describe our solution and validate it according to the requirements

Keyword List
software modeling, ontologies, integrated modeling, query language

Contents

Change Log

Version	Date	Author(s)	Changes
1.0	31.01.09	Fernando Silva Parreiras, Tobias Walter	CMR required changes included

1 Terms and Definitions

Abstract Syntax. The abstract syntax delineates the body of concepts and how they may be combined to create models. It comprises definitions of the concepts, the relationships between concepts and well-formedness rules.

Concrete Syntax. The concrete syntax provides the notation to present the constructs defined in the abstract syntax. It can be categorized in textual syntax and visual syntax.

Class-based modeling. Class-based modeling is an approach consisting basically of the essential constructs used to model a UML class diagram that are common to other metamodeling approaches like MOF and Ecore. Examples of these constructs are Class, Property, Operation, Classifier, Attribute.

UML-based Models. UML-based models are models described by metamodels using different architectures derived or based on UML. Examples of UML-based models are MOF models, Ecore models, SysML models or BPMN models.

Metamodel. Metamodel is a model defined on the M2 level.

Metaclass. Metaclass is the class construct on the M2 level according to the the OMG's Four layered metamodel architecture. In fact, when describing metamodels, metaclasess are simply referred to as classes.

Model Transformation. Model transformation is a function that receives a source model, a source metamodel, a target metamodel and a transformation script as input and produces a target model conforming with a target metamodel.

Reference Layer. Reference Layer is a set of abstract classes that are common over different packages. It defines the core elements of a given domain.

Implementation Layer. Implementation Layer is the set of classes that extend abstract classes in the Reference Layer by redefining or specifying their properties and operations.

Metamodeling Architecture. Metamodeling Architecture comprises the set of metamodels and packages declared on M2 level, one or more concrete syntaxes to design models conforming with the set of metamodels and mapping rules to accomplish the translation from the concrete syntax to the abstract syntax (metamodels).

UML-based Metamodeling. UML-based metamodeling consists of different metamodels that use constructs, such as class, property and operation as essential constructs. We use the term UML-based metamodeling to collectively refer to the metamodels UML, MOF and Ecore.

OMG's Four layered metamodel architecture. It is an architecture defined by OMG with four different levels: the metametamodel level (M3), the metamodel level (M2), the model level (M1) and the objects level (M0) (or real world).

Mapping Rules. Mapping rules are relationships among constructs in two distinct metamodels.

Ontology. In this document, the terms ontology and OWL ontology are used interchangeably. For using UML profiles we focus on OWL as language for ontologies.

Scenario. Scenarios are outlines of set of use cases where the integrated metamodel happens to be used. They describe the users' work and are used to extract use cases from it.

1.1 List of Abbreviations

AS	Abstract Syntax
KB	Knowledge Base
CS	Concrete Syntax
CbML	Class-based modeling language
CWA	Closed World Assumption
DFA	deterministic finite automaton
DL	Description Logics
DSL	Domain Specific Language
M0	Metamodel Level 0
M1	Metamodel Level 1
M2	Metamodel Level 2
M3	Metamodel Level 3
MBOTL	Model-based Ontology Translation Language
MDA	Model Driven Architecture
MM	Metamodel
MOF	Meta-Object Facility 2.0
MOST	Marrying Ontologies and Software Technologies
MTL	Model Transformation Language
NA	Not Available
NFA	nondeterministic finite automaton
OCL	UML 2.0 Object Constraint Language
ODM	Ontology Definition Metamodel
OMG	Object Management Group
OWA	Open World Assumption
OWL	Web Ontology Language
OWL 2	Web Ontology Language 2
OWL DL	The Description Logics Dialect of OWL
OWL Full	The Most Expressive Dialect of OWL
QVT	Query / View / Transformation
RDF	Resource Description Framework
RDFS	RDF Schema
RL	Reference Layer
TU	TwoUse
UML	Unified Modeling Language 2.0
WFR	Well Formedness Rules

2 Introduction

To make use of hybrid models addressed by the integration at the level of modeling languages in the MOST Deliverable 1.1[Parreiras and Walter, 2008], it is important to provide the proper tools to the developer to manage and understand the models. Thus, tools that can work on the models need to be integrated. An important service to allow developers insight into their models, and provide for improved model management, is integrated querying.

In order to be able to query and transform integrated models, a query framework needs to be integrated on the level of Model Driven Engineering (MDE). Queries provide support to the developer to fulfill requirements and address modeling decisions, so that the development of integrated tools will be simplified.

The objective of this deliverable is to investigate the possibilities for querying the combined metamodel in order to access models in a flexible manner using or combining existing languages.

The research issues pursued in this deliverable are the following: How to query and validate the integrated models? How should an integrated query language look like? How can it be mapped to reasoning technology made available by WP3?

This deliverable is organized as follows: Section 3 summarizes the motivation of the integrated metamodel. Subsequently, the requirements of such a querying solution, based on literature and on practice, are elicited and described in Sect.4. Existing query languages are studied in Sect.5. Some existing approaches to be based on are described in Sect.6. An analysis of possible solutions for querying the combined metamodel is described in Sec.7. Finally, Sect.8 finishes this document.

3 Background

3.1 Motivation of the Integrated Metamodel

In Deliverable D1.1, we presented a framework involving the integration of existing metamodels and profiles for UML and OWL modeling, including relevant (sub)standards such as OCL and considering newer developments such as SWRL, a weaving metamodel and an UML profile for developing integrated models.

As applications of ontologies in MDE spreads, many questions come to light. Disciplines like model transformation and domain specific languages become essential in order to support different kinds of models in an model driven environment. Understanding how ontology technologies like can be integrated in this field is crucial to leverage the development of such disciplines.

Aware of the need for an extensible solution, Deliverable D1.1 [Parreiras and Walter, 2008] has presented an reference layer with support to different model languages and ontology languages. Prominent applications of the reference layer are the Twouse approach
[Silva Parreiras et al.,] [Silva Parreiras et al., 2008b], the integration between KM3 and OWL, and a Model-Based Ontology Translation Language [Silva Parreiras et al., 2008a].

3.2 Applications of the Integrated Metamodel

3.2.1 TwoUse

TwoUse[Silva Parreiras et al.,][Silva Parreiras et al., 2008b] is a framework for developing integrated models, comprising the benefits of UML models and OWL ontologies. It illustrates

a concrete implementation of the MOST Reference Layer[Parreiras and Walter, 2008] and it is based on four core ideas:

- It provides an integrated MOF based metamodel as a common backbone for UML (including OCL) and OWL modeling;
- It uses an UML profile as its integrated syntactic basis supporting UML2 extension mechanisms and mappings from the profile onto the metamodel;
- It provides a canonical set of transformation rules in order to deal with integration at the semantic level.
- It extends the basic library provided by the OCL specification, which we call OCL-DL.

3.2.2 Ontologies in Domain Specific Languages

The integrated metamodel is used as basis for an Ontology Based DSL Framework for developing domain specific languages as well. Since DSL programmers have different levels of experience and have incomplete knowledge about the languages, a new technical space that allows the use of ontologies to suggest the right constructs to be used is required. Constraints are defined at the metamodel level and they are fulfilled at the modeling level, deriving guidance for the modeling process of domain models.

3.2.3 Ontology-based Transformation

The Model-Based Ontology Translation Language (MBOTL) integrates different translation problems into a representation based on the combined metamodel. In
[Silva Parreiras et al., 2008a], an platform independent approach for ontology translation based on model-driven engineering (MDE) of ontology alignments is proposed in order to reconcile semantic reasoning with idiosyncratic lexical and syntactic translations. The framework includes a language to specify ontology translations, an integrated metamodel (ATL, OCL, OWL)and translations to realize the unified language.

4 Query Requirements

This section describes requirements of a query language for the integrated model. These requirements are divided into functional and non-functional requirements.

4.1 Functional Query Requirements

Functional query requirements analyze the query functionality with respect to a structured data set. This includes all requirements concerning the technical implementation like the query processing, data representation, as well as relationship between input and output.

Complete Knowledge vs. Incomplete Knowledge The result of a query depends on the assumption about the world, i.e. how is the existing data set on which the query is executed defined. There are two different kinds of knowledge representation. In a closed world assumption the intention is that everything which is known is contained in the data set, i.e. the dataset

is considered as complete knowledge. This assumption is realized in database systems. In an open world it is assumed that the available data is not necessarily the complete knowledge of the world.

From the logical point of view there is a difference with respect to the truth value of statements which are not known to be true. The truth value of a statement in the closed world assumption is true if the statement is valid otherwise the statement is false. This also includes the case when a statement is unknown, it is assigned to be false. In the open world assumption this statement is unknown. The open world assumption realizes incomplete knowledge. In knowledge representation the intention of incomplete knowledge is quite natural, i.e. it is not possible to know everything. OWL realizes the open world assumption.

Obviously the result of a query depends on the assumption. This is demonstrated by the following example. The knowledge base consists of the statement: $Bob\ sells\ Car$, Bob and Car are individuals, and *sells* is a relationship or role in DL. The query is $Bob\ sells\ Book$. The result with the closed world assumption is no, since there is no statement in the knowledge base that Bob sells a Book. In the open world assumption the answer is unknown, since the neither the statement nor the negation is in the knowledge base.

This assumption leads to a more sophisticated effect if there are cardinality restrictions added to roles or relationships. If the *sells* relationship has the cardinality restriction 1, i.e. every individual can only sell one object. Suppose that both statements above are axioms in the knowledge base. In the closed world assumption this causes an error, because it is not allowed that Bob sells a Car and a Book. In the open world assumption this infers a statement that the Car and the Book are equal objects.

Negation as failure is related to the closed world assumption. This strategy is realized in logic programming. A logic program is a finite set of horn rules. If it is not possible to prove that a statement is valid then is is assumed that this statement is false. All unsuccessful proves are considered as a failure.

Functions

Functions are an additional part of a query language which facilitates the query mechanism. They are reusable blocks of queries and can be used as helpers. The following list contains basic functions which are (partially) implemented in query languages like SPARQL [Prud'hommeaux and Seaborne, 2008] or SQL [Kline and Kline, 2000].

- Aggregate functions: average, minimum, maximum, count, summary of result tuples.
- Scalar functions: The result of a scalar function is a single value. There is either no function input or one single value. In [Kline and Kline, 2000] there is a distinction of the following scalar functions.
  - Built-in scalar functions, e.g. current date, current user or timestamps.
  - Numeric functions: operation on numeric values.
  - String functions: operations on string values, e.g. string concatenate.
  - Date and time functions: operations on time and date types.
- Solution modifier: It supports query result modification like result sorting.
- Null value support: Operators can deal with null values as input and output of a query.

- Type conversion, like a mapping from a value to a string representation.
- Test predicates, e.g. bound test of variables: $bound(X)$.
- Basic built-in functions:
  - Logical operators: negation ($\neg$), conjunction($\wedge$) and disjunction ($\vee$).
  - Arithmetic operators: $+$, $-$, $*$, $/$
  - Comparison: Equality ($=$) and inequality ($<$, $\leq$, $>$, $\geq$)

Transitive Closure

The transitive closure of a binary relation R on a set S is the smallest *transitive relation* on this set S that contain R. In the graph theory the transitive closure describes the reachability relations between nodes. The relations are the edges.

The transitive closure is also considered in database research. One example is the generation of a view which contains the transitive closure. In theoretical database research this is described as maintaining transitive closure of graphs in relational calculus [Dong et al., 1999]. The problem of transitive closure in SQL and relational calculus is outlined in [T and Su, 2000]. SQL does not directly support the transitive closure, i.e. to execute a query in which the tuples in the result are in the transitive closure.

Unique Name Assumption

The unique name assumption implies that every name is unique. Names here refers to representatives of objects, e.g. IDs or URIs like in the web. It is assumed that if there are two different names (identifiers) the corresponding objects are also different. The unique name assumption only refers to constants.

There is a connection between unique name assumption and CWA/OWA. In the closed world assumption the unique name assumption is useful. In the open world assumption it is meaningful to omit the unique name assumption due to the incomplete knowledge base. In this case it is possible to add explicit statements that two things are same or different. In OWL there is no unique name assumption, this has to be explicitly specified with statements. In DL the unique name assumption holds.

Language Semantics

The semantics of a query language is a formal definition of the meaning of the language which is based on a logical and mathematical model. There are different language semantics like the declarative and the procedural semantics. For the integrated model a declarative query language is dedicated. A declarative semantics associates to each element of the language a mathematical meaning. A corresponding semantic function maps a value to each element of the domain.

4.2 Non-Functional Query Requirements

The non-functional query requirements describe more user-oriented requirements in order to allow the user an adequate usage of the query facilities. The aim is to ease querying for the user.

Syntax

In order to provide a common applicability for the user it is recommended to use a syntax based on the syntax of existing query languages. Obviously these existing query languages should provide similar functionality, e.g. querying models with multiple layers. For the syntax there are at least the following two candidates:

- OCL (Object Constraint Language) is a declarative language that supports query formulation for MOF-based models and metamodels.
- SPARQL in combination with an extension SPARQL FA allows querying an ontology which consists of multiple layers.

User Interface

The user interface is the connection between the user and the application for data input and output. There are two relevant types of user interfaces:

- A graphical user interface (GUI) supports data input and output in a graphical layout.
- A web user interface (WUI) enables the data input and output with a web browser.

5 Existing Query Languages

5.1 OCL

UML class diagrams alone are not expressive enough to describe behavior of operations. For example, an operation `getSalesOrder()` (Fig. 1) queries the country of the customer and returns the respective specialization of the sales order. In common UML modeling practice, a textual query language such as OCL[OMG, 2005] may be used to specify such a query.

Beyond querying, OCL may also be used to specify invariants on classes and types in the class model, to describe pre- and post conditions on operations and methods, and to specify initial and derived rules over a UML model [OMG, 2005].

The OCL syntax differs from well know query languages like SQL and SPARQL. Indeed, SQL and SPARQL do not require a starting point for query, i.e., it takes a global point of view. OCL, on the other hand, takes the object-oriented point of view, starting the queries from one given class.

In OCL, expressions are written "in the context of an instance of an specific type" [OMG, 2005]. The reserved word `self` is used to denote this instance.

OCL expressions may be used to specify the body of query operations. Since OCL is a typed language, i.e., each OCL expression is evaluated to a value, expressions may be chained to specify complex queries or invariants.

Example

Let us consider the case of an international e-commerce system elaborated in [Shalloway and Trott, 2001]. The characteristics of the system to be designed include:

- It is supposed to be a sales order system for Canada and the United States.

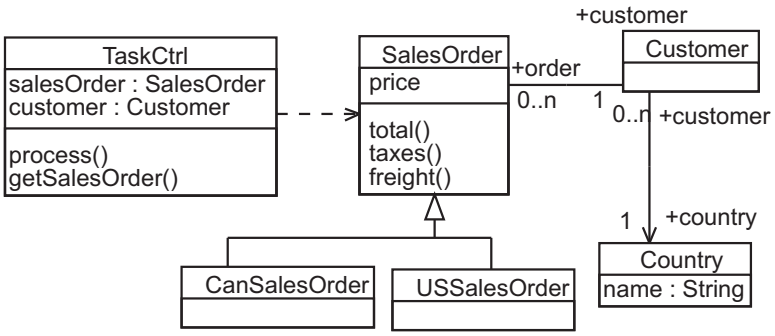

Figure 1: UML Class Diagram of the problem domain.

- Calculate freight and taxes based on the country.
- Use Government Sales Tax (GST) and Provincial Sales Tax (PST) for tax in Canada.

A snippet of the corresponding UML class diagram is presented in Fig. 1. The class `TaskCtrl` is responsible for controlling the sales orders. A `SalesOrder` can be a `USSalesOrder` or a `CanSalesOrder`, according to the `Country` where the `Customer` lives.

The operation getSalesOrder() queries the country of the customer and returns the specialization of the sales order. Following the example mentioned above, the target behavior could be theoretically denoted by the following OCL expression:

```
context TaskCtrl::getSalesOrder():SalesOrder
 body:
  if customer.country.name = 'USA' then
    self.salesOrder.oclAsType(USSalesOrder)
  else
    if customer.country.name = 'Canada' then
      self.salesOrder.oclAsType(CanSalesOrder)
    endif
  endif
```

However, this way of specifying the operation `getSalesOrder()` exhibits some shortcomings. The semantics of the subclasses of `SalesOrder`, i.e. the semantics of `USSalesOrder` and `CanSalesOrder`, are embedded in nested conditions in the operation specification of a method of `TaskCtrl`. Hence, the semantics of `USSalesOrder` and `CanSalesOrder` may be difficult to find and understand in larger domains. They may even appear redundantly when the same conditions need to be applied somewhere else in the specification. Furthermore, the description of the classes `CanSalesOrder` and `USSalesOrder` are stated at least twice: once in the class declaration and once, implicitly, as an expression of the operation `TaskCtrl.getSalesOrder()`.

OCL defines a predefined class called `OclAny`, which acts as a superclass for all the types except for the OCL pre-defined `collection` types. Hence, features of `OclAny` are available on each object in all OCL expressions, and all classes in a UML model inherit all operations defined on `OclAny`. We highlight two of these operations:

- oclIsTypeOf(typespec: **OclType**): **Boolean**. Evaluates to `true` if the given object is of the type identified by `typespec`.

- oclIsKindOf(typespec: **OclType**): **Boolean**. Evaluates to `true` if the object is of the type identified by `typespec` or one of its subtypes.

Let us exemplify these operations. The first one evaluates to true if we have an instance of `SalesOrder` and ask whether it is an instance of `SalesOrder`. The second one evaluates to true if the prior example evaluates to true or if we have an instance of `USSalesOrder` and ask whether it is an instance of `SalesOrder`, *but not the opposite*.

5.1.1 Semantics

The specification of OCL is given in natural language, although an informative formal semantics is annexed to the specification. Despite this informative formal semantics based on [Richters, 2002], other strategies have been used to define the semantics of OCL.

Roe et al.[Roe et al., 2003] have proposed mappings between UML Models incorporating OCL Constraints and Object-Z[Smith, 2000]. Object-Z prescribes a formal specification for object-orientation extending the language Z.

Schmitt et al.[Schmitt, 2001] propose a model theoretic Semantics of UML class diagrams and OCL. It defines the semantics of model properties and excludes the previous values of properties in postconditions. This work was extended later to deal with this issue, when translating OCL into Dynamic Logic.

Additionally, Beckert *et al.* proposes a translation of OCL into First-order Predicate Logic [Beckert et al., 2002].

Bucker presents a representation of the semantics of OCL in higher-order logic [Brucker and Wolff, 2002]. The proof calculi allowing the implementation of an OCL reasoner is defined in [Brucker and Wolff, 2006].

5.1.2 Complexity

In despite of the available semantics of OCL described above, reasoning with OCL is undecidable in worst case scenario.

5.2 Conjunctive Query

In database theory, a conjunctive query is a restricted form of first-order queries. A large part of queries issued on relational databases can be written as conjunctive queries, and large parts of other first-order queries can be written as conjunctive queries. Conjunctive queries also have a number of desirable theoretical properties that larger classes of queries (e.g., the relational algebra queries) do not share.

5.2.1 Syntax

The conjunctive queries are simply the fragment of first-order logic given by the set of formulae that can be constructed from atomic formulae using conjunction($\wedge$) and existential quantification ($\exists$), but not using disjunction ($\vee$), negation($\neg$), or universal quantification ($\forall$). A conjunctive query consists of two parts, the head, and the body. The head states which variables from the query should be returned to the user and the body of the query consists of one or more atoms which bind variables or literal values to concepts or roles within the ontology.

Conjunctive queries have a normal form:

$$(\exists y_1) \ldots (\exists y_n)(p_1(x_1, \ldots, x_m, y_1, \ldots, y_n) \wedge \ldots \wedge p_k(x_1, \ldots, x_m, y_1, \ldots, y_n)).$$

They can also be expressed in Datalog:

$$q(x_1, \ldots, x_m) : -p_1(x_1, \ldots, x_m, y_1, \ldots, y_), \ldots, p_k(x_1, \ldots, x_m, y_1, \ldots, y_).$$

Example

$$Q(x, y) : -Student(x) \wedge hasFriend(x, y)$$

The above query is a query for any instance of the concept Person and binds it to the variable X, and finds any instance of the hasFriend role where the Student X has some friend which is bound to the variable Y.

5.2.2 Semantics

The interpretation $\mathcal{I}$ satisfies a query $Q = q_1 \wedge \ldots \wedge q_n$ and a tuple t is a certain answer to a query Q given some source instances if it is in the answer to this query over every instance of the integrated database that satisfies all the source annotations.

5.2.3 Complexity

For the study of the computational complexity of evaluating conjunctive queries, two problems have to be distinguished. The first is the problem of evaluating a conjunctive query on a relational database where both the query and the database are considered part of the input. The complexity of this problem is usually referred to a combined complexity, whiles the complexity of the problem of evaluating a query on a relational database, where the query is assumed fixed, and is called data complexity [Vardi, 1982].

Conjunctive queries are NP-complete with respect to combined complexity [Chandra and Merlin, 1977], while the data complexity of conjunctive queries is very low, in the parallel complexity class AC0, which is contained in LOGSPACE and thus in polynomial time. The NP-hardness of conjunctive queries may appear surprising, since relational algebra and SQL strictly subsume the conjunctive queries and are thus at least as hard (in fact, relational algebra is PSPACE-complete with respect to combined complexity and is therefore even harder under widely-held complexity-theoretic assumptions). However, in the usual application scenario, databases are large, while queries are very small, and the data complexity model may be appropriate for studying and describing their difficulty.

5.3 SPARQL

SPARQL [Prud'hommeaux and Seaborne, 2008]is W3C standard query language for RDF data on the semantic web. SPARQL is based on graph pattern matching. A SPARQL query matches a graph pattern against a dataset consisting of one or more input graphs. The resulting variable bindings are either returned in tabular form ("select queries") or embedded into a template description in order to generate new RDF data ("construct queries").

5.3.1 Syntax

We assume basic familiarity of the reader with RDF and SPARQL, and will only briefly introduce some basics here: We define a SPARQL query as a tuple (E, DS, R) where E is a SPARQL algebra expression, DS is a RDF dataset, and R is a query form. SPARQL has four query forms; For a SELECT query, a result from R is simply a set of variables, a CONSTRUCT query returns is a set of triple patterns. An ASK query returns a boolean representing whether a query pattern matches or not. While, DESCRIBE query returns an RDF graph that describes the resources found.

We assume the pairwise disjoint, infinite sets $\mathcal{V}_{uri}, \mathcal{V}_{bnode}, \mathcal{V}_{lit}$, and $\mathcal{V}_{uri}$, which denote URIs, blank node identifiers, RDF literals, and variables respectively. Graph patterns are recursively defined as follows:

- $\{s, p, o\}$ is a graph pattern where $s, o \in \mathcal{V}_{uri} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{lit} \cup \mathcal{V}_{uri}$ and $p \in \mathcal{V}_{uri} \cup \mathcal{V}_{var}$.
- A set of graph patterns is a graph pattern.
- Let P, P_1, P_2 be graph patterns, R a filter expression, and $i \in \mathcal{V}_{uri} \cup \mathcal{V}_{var}$, then P_1 OPTIONAL P_2, P_1 UNION P_2, GRAPH iP, and P FILTER R are graph patterns.

Example

Example of SPARQL:

```
SELECT   (?t ?x ?y)
WHERE    {?x rdf:type ?t .
          ?t rdfs:subClassOf ont:Employee.
          ?x ont:teacherOf _:a .
          ?y ont:takesCourse _:a .
         }
```

5.3.2 Semantics

The sematic of SPARQL is based on mappings, partial functions from variables to terms. A mapping μ is a solution of triple pattern t in G if $\mu(t) \in G$ and $dom(\mu) = var(t)$ w.r.t. an evaluation of t in G is the set of solutions (written $[[t]]_G$). The semantics of complex SPARQL query constructs can be defined by using set of mapping. Let M_1 and M_2 be sets of mappings then we can define as folows:

- Join ($M_1 \bowtie M_2$) extending mappings in $M_1$with compatible mappings in M_2.
- Difference($M_1 \backslash M_2$) mappings in M_1 that cannot be extended with mappings in M_2.
- Union ($M_1 \cup M_2$) mappings in M_1 plus mappings in M_2.
- Left Outer Join ($M_1 \bowtie M_2$) is equivalent to $(M_1 \bowtie M_2) \cup (M_1 \backslash M_2)$

The evaluation of a graph pattern over G, denoted by $[[\bullet]]_G$, defined recursively as follows:

- $[[(P_1 \text{ AND } P_2)]]_G = [[P_1]]_G \bowtie [[P_2]]_G$

- $[[(P_1 \text{ UNION } P_1)]]_G = [[P1]]_G \cup [[P2]]_G$
- $[[(P_1 \text{ OPT } P_1)]]_G = [[P1]]_G \bowtie [[P2]]_G$
- $[[(P \text{ FILTER } R)]]_G = \{\mu 2[[P]]_G \mid \mu \text{ satisfies } R\}$

For more detail, refer the reader to [Pérez et al., 2006]

5.3.3 Complexity

From the study of the computational complexity of evaluating SPARQL queries [Pérez et al., 2006]. Given An RDF dataset D, a graph pattern P and a mapping μ. we can define computational complexity of evaluating SPARQL queries as follows. First,the evaluation can be solved in time $\mathcal{O}(\mid P \mid \times \mid D \mid)$ for graph pattern expressions constructed by using only AND and FILTER operators. Second, the evaluation is `NP-complete` for graph pattern expressions constructed by using only AND, FILTER and UNION operators. Third, the valuation is `PSPACE-complete` for graph pattern expressions. Finally, the evaluation(P)(data complexity) is in `LOGSPACE` for every graph pattern expression P.

5.4 SPARQL-DL

SPARQL-DL is a rich query language for OWL-DL ontologies. It provides an OWL-DL-like semantics for SPARQL basic graph patterns which involves as special cases both conjunctive ABox queries and mixed TBox/RBox/ABox queries over Description Logic (DL) ontologies [Sirin and Parsia, 2007].

5.4.1 Syntax

In this section we provide a brief description of SPARQL-DL syntax and refer the reader to [Sirin and Parsia, 2007] for more details. Let $\mathcal{V}_\mathcal{O} = (\mathcal{V}_{cls}, \mathcal{V}_{op}, \mathcal{V}_{dp}, \mathcal{V}_{ap}, \mathcal{V}_{ind}, \mathcal{V}_D, \mathcal{V}_{lit})$ be an OWL-DL vocabulary. Let $\mathcal{V}_{bnode}$ and $\mathcal{V}_{var}$ be the set of bnode identifiers and set of variables.A SPARQL-DL query atom q is of the form:

$q \leftarrow$ Type(a, C) | PropertyValue(a, p, v) | SameAs(a, b) | DifferentFrom(a, b) |
EquivalentClass(C_1, C_2) | SubClassOf(C_1, C_2) | DisjointWith(C_1, C_2) |
ComplementOf(C_1, C_2) | EquivalentProperty(p_1, p_2) | SubPropertyOf(p_1, p_2) |
InverseOf(p_1, p_2) | ObjectProperty(p) | DatatypeProperty(p) | Functional(p) |
InverseFunctional(p) | Transitive(p) | Symmetric(p) | Annotation(s, p_a, o)

where $a, b \in \mathcal{V}_{uri} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{var}, v \in \mathcal{V}_{uri} \cup \mathcal{V}_{lit} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{var}, p, q \in \mathcal{V}_{uri} \cup \mathcal{V}_{var}, C, D \in S_c \cup \mathcal{V}_{var}, s \in \mathcal{V}_{uri}, p_a \in \mathcal{V}_{ap}, o \in \mathcal{V}_{uri} \cup \mathcal{V}_{lit}$. A SPARQL-DL query Q is a finite set of SPARQL-DL query atoms and the query is interpreted as the conjunction of the elements in the set.

Example

Example of SPARQL-DL:

```
Type(?x,?t), SubClassOf(?t,Employee),
PropertyValue(?x,teacherOf,_:a),
PropertyValue(?y,takesCourse,_:a).
```

5.4.2 Semantics

The semantics of SPARQL-DL is very similar to the semantics of OWL-DL.We specify the conditions under which an interpretation satisfies a query atom in much the same way that satisfaction is defined for OWL-DL axioms.

The interpretation $\mathcal{I}$ satisfies a query $Q = q_1 \wedge \ldots \wedge q_n$ w.r.t. an evaluation σ(written $\mathcal{I} \models \sigma q$) iff $\mathcal{I} \models \sigma q_i$ for every $i = 1, ..., n$. Note that, we are only interested in the existence of an evaluation and we simply say that $\mathcal{I}$ satisfies a query Q (written $\mathcal{I} \models Q$) if there exists an evaluation σ such that $\mathcal{I} \models Q$. Finally, we say that Q is a logical consequence of the ontology $\mathcal{O}$ (written $\mathcal{I} \models Q$) if the query is satisfied by every model of $\mathcal{O}$, i.e. $\mathcal{I} \models \mathcal{O}$ implies $\mathcal{I} \models Q$. A solution to a SPARQL-DL query $Q = q_1 \wedge \ldots \wedge q_n$ w.r.t. an OWL-DL ontology $\mathcal{O}$ is a *variablemapping* $\mu : \mathcal{V}_{var} \rightarrow \mathcal{V}_{uri} \cup \mathcal{V}_{lit}$ such that when all the variables in Q are substituted with the corresponding value from μ we get a semi-ground query $\mu(Q)$ compatible with VO and $\mathcal{O} \models \mu(Q)$. The solution set $S(Q)$ for a query Q is the set of all such solutions.

Form of the query atom	Condition on interpretation
Type(a, C)	$\sigma(a) \in C^{\mathcal{I}}$
PropertyValue(a, p, v)	$\langle \sigma(a), \sigma(v) \rangle \in p^{\mathcal{I}}$
SameAs(a, b)	$\sigma(a) = \sigma(b)$
DifferentFrom(a, b)	$\sigma(a) \neq \sigma(b)$
SubClassOf(C_1, C_2)	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
EquivalentClass(C_1, C_2)	$C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$
DisjointWith(C_1, C_2)	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$
ComplementOf(C_1, C_2)	$C_1^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C_2^{\mathcal{I}}$
SubPropertyOf(p, q)	$p^{\mathcal{I}} \subseteq q^{\mathcal{I}}$
EquivalentProperty(p, q)	$p^{\mathcal{I}} = q^{\mathcal{I}}$
Functional(p)	$\langle x, y \rangle \in p^{\mathcal{I}} and \langle x, z \rangle \in p^{\mathcal{I}} implies y = z$
InverseFunctional(p)	$\langle y, x \rangle \in p^{\mathcal{I}} and \langle z, x \rangle \in p^{\mathcal{I}} implies y = z$
Transitive(p)	$\langle x, y \rangle \in p^{\mathcal{I}} and \langle y, z \rangle \in p^{\mathcal{I}} implies \langle x, z \rangle \in p^{\mathcal{I}}$
Symmetric(p)	$\langle x, y \rangle \in p^{\mathcal{I}} implies \langle y, x \rangle \in p^{\mathcal{I}}$
Annotation(s, p_a, o)	$\langle s, o \rangle \in p_a^{\mathcal{I}}$

Table 1: Satisfaction of a SPARQL-DL query atom w.r.t. an interpretation

5.4.3 Complexity

Currently, there is no study about the computational of the complexity of SPARQL-DL yet. However, we believe that the complexity of the evaluation SPARQL-DL query is equivalent to the SPARQL query in the worst case since SPARQL-DL is a substantial subset of SPARQL.

5.5 GReQL

Graphs are well-defined mathematical and formal structures. Different algorithms and methods exist to work efficiently on graphs. Normally, graphs appear all-around in today's software engineering. For example and for simple imagination, UML class diagrams also can be seen as a graph, where, in general, the nodes are represented by classes and the edges by associations.

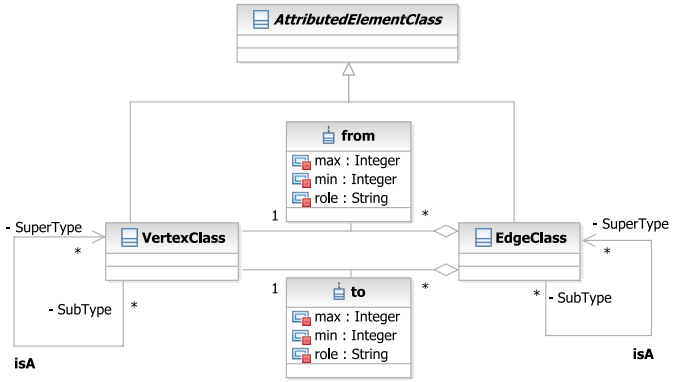

Figure 2: Part of the grUML Metametamodel

A very general kind of graphs are the TGraphs [Ebert et al., 2002]. Such graphs are *directed*, its edges and vertices are *typed* and *attributed* and for each node the incident edges are *ordered*. Each graph is an instance of its corresponding *schema* (metamodel) which for example defines types of edges and vertices and structures them in hierarchies.

5.5.1 Modeling with grUML

The modeling language *grUML* (graphUML) is a sublanguage of UML and is based on TGraphs. Using UML model elements in grUML also graph schemas are defined. Sets of vertices are represented by classes with attributes. Sets of edges are represented by associations which can contain attributes too. Different types of vertices and edges are defined using specialization associations of UML.

Thus a grUML diagram is the visualization of a TGraph at the same meta-layer *Mi*. The TGraph itself is instance of the grUML-Metamodel at the corresponding metameta-layer *Mi+1*.

Let's consider an example to see into using TGraphs and grUML. Figure 2 depicts an extreme simplification of the grUML metametamodel which lies at the M3-layer of the four-layer architecture[Miller et al., 2001]. Here we have one class for vertexes and one class for edges. Each edge connects two vertex classes. `EdgeClass` and `VertexClass` are specializations of `AttributedElementClass` which means, that both can be attributed.

Using the grUML metametamodel at the M3-layer we can create instances of the model at the M2-layer. TGraphs are composed of such instances whose type for example is `VertexClass` or `EdgeClass`. Figure 3 depicts an object model of a simple domain specific language at the M2-layer, that is defined by the corresponding grUML metametamodel. Here we have different objects of type `VertexClass`, for example `Entity`, `Attribute` or `Datatype`. `Attribute` has the two specializations `SimpleAttribute` and `ReferenceAttribute`. Using an object `hasAttribute` of type `EdgeClass Entity` is connected with `Attribute`. In object diagram there are two objects `fromEntity` and `toAttribute` which are instances of the association classes `from` and `to` in the grUML metametamodel. The attributes and its values give information about the role name and the cardinality. In the example an entity has 0 to many (the value -1 means many) attributes but an attribute belongs only to exactly 1 entity. Analogously, the object `referenceTo` of type `EdgeClass` and its corresponding links connect the

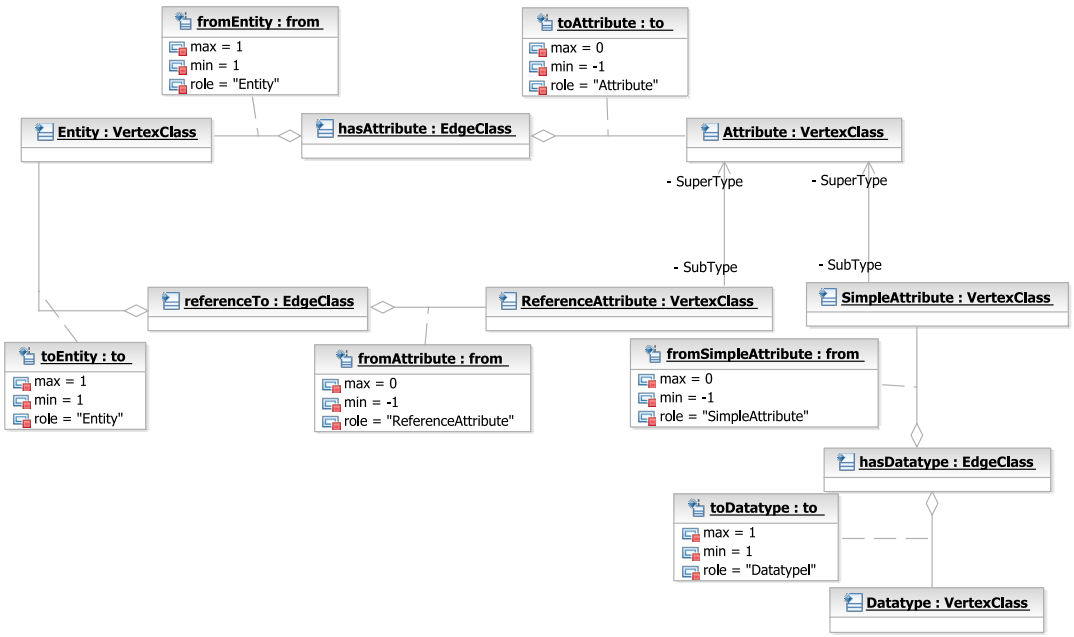

Figure 3: Domain Specific Language represented by instances of the grUML Metamodel

`ReferenceAttribute`-object with exactly one `Entity`-object. The model also gives the information that a `SimpleAttribute` has exactly one `Datatype`, but of course a `Datatype` can be used by many `SimpleAttributes`.

For an easier understandable representation, the object diagram of figure 3 is visualized using a concrete syntax. Using the grUML metametamodel at the M3-layer we can visualize each TGraph again as a grUML-diagram (cf. figure 3). Vice versa, each grUML-diagram can also be visualized by a TGraph because grUML-diagrams have a graph-based extensional semantic.

5.5.2 Querying Models with GReQL

As explained above if we are modeling UML class diagrams (respectively grUML diagrams) the abstract representation is a TGraph.

Now GReQL, the *Graph Repository Query Language*, comes into play. GReQL is a declarative query language for TGraphs which is developed at the *Institute of Software Technology of the University of Koblenz-Landau*[1].

GReQL can be used to extract different kinds of information of TGraphs, for example attributes of vertices and edges or complete structures inside of a graph. In the following we want to give a short overview of the syntax of GReQL.

[1] `http://www.uni-koblenz-landau.de/koblenz/fb4/institute/IST/AGEbert/MainResearch/Graphentechnologie/GReQL`

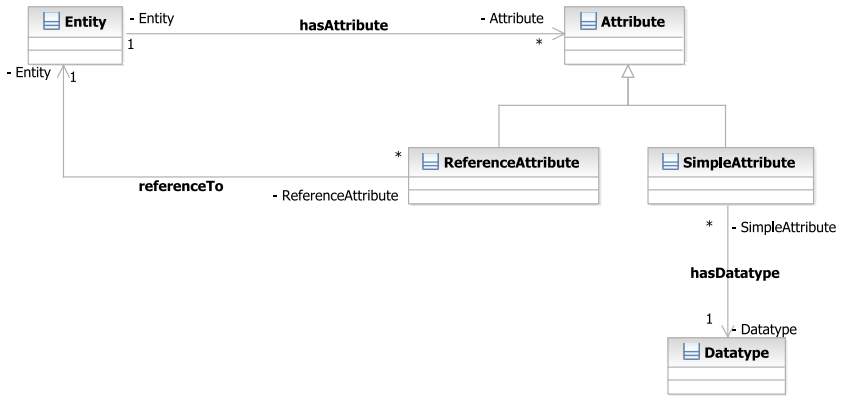

Figure 4: Domain Specific Language visualized by grUML-Diagram

5.5.3 Syntax

GReQL queries have typical syntactic *FWR-structure*, which means, that queries consist of the three clauses `from`, `with` and `report` [Bildhauer, 2006].

The `from`-clause declares variables for the concerned elements (nodes and edges) in the graph with the corresponding domain of each variable. The domain is composed of all types of the graph schema. Regarding to our schema in figure 3 we have the vertex types `Entity`, `Attribute`, `ReferenceAttribute`, `SimpleAttribute` and `Datatype`. For edges we have the types `hasAttribute`, `hasDatatype` and `referenceTo`.

The optional `with`-clause summarizes predicates which have to be fulfilled from the variables. These predicates include powerful graph oriented expressions like regular path expressions. The `with`-clause consists of predicate formulas, therefore boolean operators like `not`, `and` and `or` exist and quantifiers like `forall` and `exists` can be used.

The `report`-clause determines the result structure of the query.

Listing 1 gives a simple example of the structure of a GReQL query. Corresponding to graph represented in figure 3 the query returns all entities that have at least one attribute. In the `from`-clause we define the concerned elements in the query. In the `with`-clause we state that an edge of type `hasAttribute` has to exist between an `Entity`-node and an `Attribute`-node. If this predicate is fulfilled the defined variable in the `result`-clause is returned.

Listing 1: A simple GReQL Query

```
from    entity:V{Entity}
        attribute:V{Entity}
with    entity -->{hasAttribute}
        attribute
report  entity
end
```

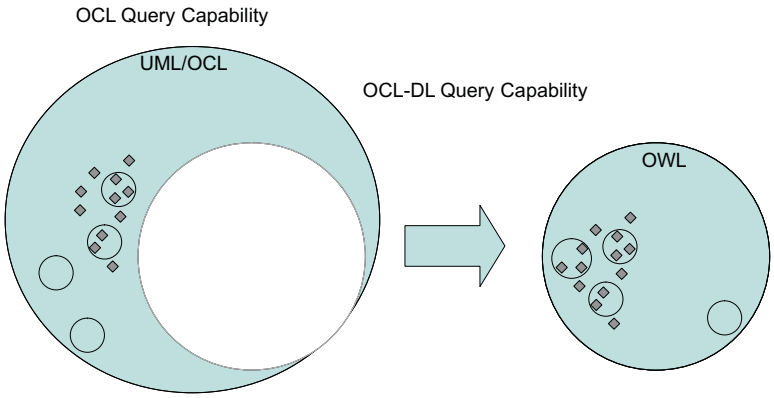

Figure 5: TwoUse OCL overview.

5.5.4 Complexity

The evaluation of path descriptions of GReQL queries is done by an automaton-driven graph traversal. In the worst case, a DFA that accepts exactly the same language as a NFA with n states could have up to 2^n states, but this explosion does not occur practically. To perform the path search, the DFAs are used to drive a worklist algorithm that traverses the graph and marks the vertices with the states of the DFA. This marking ensures that the search algorithms terminates and that its complexity is $O(|S|\cdot\max(|V|;|E|)$ where S is the state set of the DFA and V and E are the vertex and edge set of the graph, respectively [Bildhauer and Ebert, 2008].

6 Existing Approaches

6.1 TwoUse OCL

TwoUse OCL is an extension of the OCL language with support to built-in operations. These built-in operations call reasoning services to reason over models that are translated into OWL. Figure 5 depicts the overview of TwoUse OCL.

Part of the UML/OCL KB is translated to OWL, namely the OWL KB. The OWL KB can be explored by reasoning services or shared on the semantic web. The result of the expressions are translated back into OCL and available to the OCL engine.

TwoUse includes new operations that rely on *reasoning engine services* to extend the boundaries of OCL towards OWL. One may use the TwoUse operation `owlIsInstanceOf(USSalesOrder)` which makes use of a reasoner and returns true if, the properties of the object satisfy the sufficient conditions to be a member of class `USSalesOrder`. The following are the core operations of TwoUse OCL:

- owlIsInstanceOf(typespec: **OclType**): **Boolean**. Evaluates to `true` if the object satisfy all the logical requirements of the OWL class description `typespec`.

- owlAllNamedClass(): **Set**(**OclType**). Returns all named classes classified by a reasoner, whose the object satisfies the logical requirements.

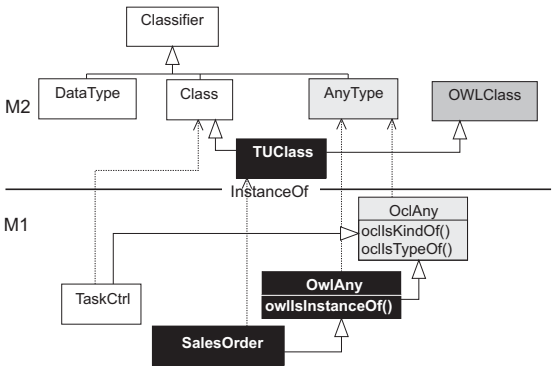

Figure 6: TwoUse OCL Library Extension and sample model classes.

- owlAllInstances(): **Set**(T). This is an introspective operation which returns all instances that satisfy the logical requirements of the OWL class description of the given object.
- owlMostSpecNamedClass(): **OclType**. Returns the intersection of owlAllNamedClass().

All TwoUse OCL operations make use of reasoning services, i.e., they rely on inferences over complex descriptions of concepts and instances. In order to include these operations, we extend the OCL library by adding a new M1 instance of the OCL metaclass `AnyType` called `OwlAny`. While all M1 classifiers except collections conform to `OclAny` in OCL, only M1 instances of TUClass conforms to `OwlAny`. Thus, `OwlAny` acts as a supertype for all TwoUse classes. As the result all TwoUse classes inherit all operations of `OwlAny`.

Figure 6 depicts the TwoUse OCL in the context of four metamodels: OCL, UML, OWL and TwoUse. At level M2, white boxes represent metaclasses imported from UML metamodel and light grey boxes represent metaclasses from OCL metamodel. The black box represent the TwoUse metaclass imported from TwoUse metamodel, which specializes the UML class and the OWL class, that are the dark grey box. At level M1, we show the class `SalesOrder` which is a M1 instance of the metaclass `TUClass` and, because of that, is a subtype of the TwoUse OCL class `OwlAny`. `SalesOrder` is indirectly a M1 instance of the UML metaclass `Class` too and so is indirectly a subtype of the OCL Class `OclAny`. The class `TaskCtrl` is a pure UML Class and is also a subtype of `OclAny`.

Since OWL classes do not support operations and cannot be referred to in OCL expressions, we use a TwoUse Class to build the bridge. A TwoUse Class inherits from both OWL and UML Classes. Therefore, a M1 instance of the metaclass `TUClass` is an instance of the metaclass `OWLClass` too. Thus, OCL implementations are extended to include OWL querying capabilities.

6.1.1 Semantics

OWL has model-theoretic semantics and allows for inferring on expressions that must be explicitly specified in the UML language. Therefore, it is possible to have the same concept specified by different set of statements in OWL and UML.

Nevertheless, the assumptions about these expressions can lead to different results. Recapping our example of Sect.5.1, suppose we change a little the model and declare the class `AlcaCustomer` as subclass of `Customer` and superclass of `CanadianCustomer` and `USCustomer`,

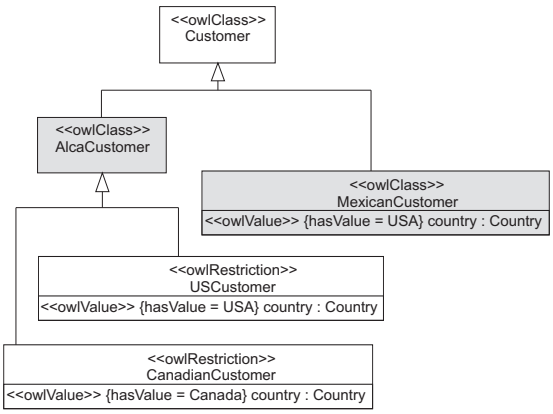

Figure 7: UML Profiled Class Diagram with new classes.

and the class `MexicanCustomer` as subclass of `Customer` (Fig. 7). If we ask the UML model whether an instance of `MexicanCustomer` is a `AlcaCustomer` (oclIsKindOf(AlcaCustomer)), the answer will be *no*. Nevertheless, if we ask the OWL ontology, the answer is *undefined*, unless we explicitly state that the class `MexicanCustomer` is a subclass of the complement of the class `AlcaCustomer`, i.e., a non-ALCA customer. The reason is that the UML, OCL and the databases use the Closed World Assumption (CWA) whereas OWL ontologies use the Open World Assumption (OWA).

The CWA and OWA are not contradictory and should be seen as complementary to each other. Recent results [Motik et al., 2007] show that it is possible to control the degree of incompleteness in an ontology, since OWA assumes incomplete information by default, obtaining a more versatile formalism. Such "underspecification" can be used to allow reuse and extension and does not mean insufficiency. Again using our example, suppose we define a incomplete list of Alca countries comprising just Canada and USA, because these are the countries the store ships to, and we do not need to know the 32 others. In the future, the store starts shipping to Mexico. If we query our ontology whether Mexico is member of Alca, the answer is *undefined*, which is reasonable, providing that our list of Alca countries is incomplete and does not include Mexico and the 31 remaining countries.

TwoUse delegates the decision of using OWA or CWA to the user, which chooses the amount of the OCL operations (CWA) or the TwoUse OCL extension (OWA). Such choice must be done aware of the consequences, which demands the deep understanding of the user about the semantics of OWL and the OCL-DL operations.

6.2 KAON2

KAON2 [KAON2, 2005] is an infrastructure for managing and reasoning with OWL-DL, SWRL, and F-Logic ontologies. In this section, we briefly summarize the functionality that is provided by the tool:

- Provides an integrated API for reading, writing, and management of OWL-DL, SWRL, and F-Logic ontologies. Currently, OWL RDF and OWL XML file formats are supported.

- Provides built-in reasoner for OWL DL except nominals and datatypes (SHIQ), extended with DL-safe subset of SWRL. Reasoning is based on novel algorithms, which reduce an OWL ontology to a (disjunctive) Datalog program. These algorithms allow KAON2 to handle relatively large ontologies with high efficiency.
- Supports the answering of conjunctive queries expressed in SPARQL.
- Supports the DIG interface, and can therefore be used with ontology editors.
- Accesses information that is stored in relational databases based on mappings between ontology entities and database tables.
- Provides tools for creating a meta-ontology beyond OWL DL ontologies within OWL DL.

6.2.1 A Metamodel Approach in KAON2

[Vrandecic et al., 2006] has introduced an approach to offer metadata about the defined classes or properties in an ontology. Their approach relies on the metamodeling features of Model Driven Architecture (MDA), which provide the means for the specification of modeling languages in a standardized, platform independent manner. For more details refer to [Vrandecic et al., 2006].

Example

This example has been taken from [Vrandecic et al., 2006], in order to show how to define an OWL DL meta-ontology. In this ontology we explicitly capture axioms. Ontologies can be transformed to become instance data with regard to the vocabulary of the meta-ontology.

1. CLASS $\sqsubseteq$ ONTOLOGYENTITY
2. AXIOM $\sqsubseteq$ ONTOLOGYELEMENT
3. SUBCLASSOFAXIOM $\sqsubseteq$ AXIOM
4. EQUIVALENTCLASSAXIOM $\sqsubseteq$ AXIOM
5. DISJOINTWITHAXIOM $\sqsubseteq$ AXIOM
6. $\top \sqsubseteq \forall$SUBCLASSOFSUBCLASS.CLASS
7. $\top \sqsubseteq \forall$SUBCLASSOFSUBCLASS^{-1}.SUBCLASSOFAXIOM
8. $\sqsubseteq \leq$ 1 SUBCLASSOFSUBCLASS.$\top$
9. $\top \sqsubseteq \forall$ 8SUBCLASSOFSUPERCLASS.CLASS
10. $\top \sqsubseteq \forall$SUBCLASSOFSUPERCLASS^{-1}.SUBCLASSOFAXIOM
11. $\sqsubseteq \leq$ 1 SUBCLASSOFSUPERCLASS.$\top$
12. $\top \sqsubseteq \forall$EQUIVALENTCLASS.CLASS
13. $\top \sqsubseteq \forall$EQUIVALENTCLASS^{-1}.EQUIVALENTCLASSAXIOM

14. $\top \sqsubseteq \forall$DISJOINTWITH.CLASS

15. $\top \sqsubseteq \forall$DISJOINTWITH^{-1}.DISJOINTWITHAXIOM

Axioms 1-5 define the terms used. Every axiom type is defined by a class of its own (refer to the following example). The rest of the axioms defines the domain and ranges of the used properties. The following is an example of a very simple ontology, with just one axiom that states that all persons are living beings

PERSON $\sqsubseteq$ MORTAL

Using the meta-ontology, we can represent this ontology as follows:

CLASS($Mortal$)

CLASS($Person$)

SUBCLASSOFAXIOM($axiom1$)

SUBCLASSOFSUPERCLASS($axiom1, Mortal$)

SUBCLASSOFSUBCLASS($axiom1, Person$)

SUBCLASSOF($Person, Mortal$)

Please note that the class `PERSON` is something else than its representing individual in the metaontology, which is the individual Person. The axiom of the original ontology is reified explicitly as the individual axiom1, an instance of the class `SUBCLASSOFAXIOM`. The axiom is connected to the entities taking part in that axiom with the given properties. The last axiom offers a direct property instance representing the original axiom. Now it is possible to state further facts about this axiom, like its source or the confidence we put into the axiom, within the ontology:

CREATOR($axiom1, Aristotle$)

CONFIDENCE($axiom1, 0.95$)

Naturally, we also can talk about the entities of the ontology in the same manner:

CREATOR($Person, God$)

6.3 OWL-FA

OWL-FA[Pan et al., 2005] is an extension of OWL-DL, which enables metamodeling. OWL-FA uses the architecture of RDFS(FA). In OWL-DL it is not allowed, that an object is a class, property or individual at the same time, but this is a typical characteristic of metamodel descriptions. The idea is, to interpret objects depending on the layer they belong.

6.3.1 Syntax

The syntax of OWL-FA is similar to that of OWL-DL. The OWL-FA vocabulary consists of the set for class names $\mathcal{V}_{cls_i}$ for each layer i, the sets of abstract property names $\mathcal{V}_{ap_i}$, the set of datatype name $\mathcal{V}_D$, the set of datatype property names $\mathcal{V}_{dp}$ and the set of individual names $\mathcal{V}_{ind}$.

The abstract syntax for an OWL-FA class definition is:

$$C \leftarrow \top \mid \bot \mid CN \mid \neg_i C \mid C \sqcap_i D \mid C \sqcup_i D \mid \{o\} \mid \exists_i R.C \mid \forall_i R.C \mid \leq_i nR \mid \geq_i nR \mid$$
$$(if\ i = 1)\ \exists_1 T.d \mid \forall_1 T.d \mid \leq_1 nT \mid \geq_1 nT$$

In this abstract definition, CN ($CN \in \mathcal{V}_{cls_i}$) is the name of an atomic class. C and D are OWL-FA classes in layer i ($1 \leq i \leq k$, $k \geq 1$), $T \in \mathcal{V}_{dp}$ is a name of a datatype property, R is an OWL-FA property in layer i and $o \in \mathcal{V}_{ind}$ is an individual.

6.3.2 Semantics

OWL-FA has a model theoretic semantics, which is defined in terms of interpretations. Given an OWL-FA alphabet V, a set of built-in datatype names $\mathbf{B} \subseteq \mathcal{V}_D$ and an integer k ≥ 1, an *OWL-FA interpretation* is a pair $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$, where $\Delta^{\mathcal{J}}$ is the domain (a non-empty set) and $\cdot^{\mathcal{J}}$ is the interpretation function, which satisfy the following conditions (where $0 \leq \mathrm{i} \leq \mathrm{k}$):

1. $\Delta^{\mathcal{J}} = \bigcup_{0 \leq \mathrm{i} \leq \mathrm{k}-1} \Delta_{A_\mathrm{i}}^{\mathcal{J}} \cup \Delta_{\mathbf{D}}$, where $\Delta_{A_\mathrm{i}}^{\mathcal{J}}$ is the domain for stratum (layer) i and $\Delta_{\mathbf{D}}$ is the datatype domain;
2. $\Delta_{A_{\mathrm{i}+1}}^{\mathcal{J}} = 2^{\Delta_{A_\mathrm{i}}^{\mathcal{J}}} \cup 2^{\Delta_{A_\mathrm{i}}^{\mathcal{J}} \times \Delta_{A_\mathrm{i}}^{\mathcal{J}}}$ and $\Delta_{\mathbf{D}} \cap \Delta_{A_\mathrm{i}}^{\mathcal{J}} = \emptyset$;
3. $\forall \mathsf{a} \in \mathcal{V}_D :\ \mathsf{a}^{\mathcal{J}} \in \Delta_{A_0}^{\mathcal{J}}$ and $\forall \mathsf{C} \in \mathcal{V}_{cls_{i+1}} :\ \mathsf{C}^{\mathcal{J}} \subseteq \Delta_{A_\mathrm{i}}^{\mathcal{J}}$;
4. $\forall R \in \mathcal{V}_{ap_{i+1}} :\ R^{\mathcal{J}} \subseteq \Delta_{A_\mathrm{i}}^{\mathcal{J}} \times \Delta_{A_\mathrm{i}}^{\mathcal{J}}$ and $\forall T \in \mathcal{V}_{dp} :\ T^{\mathcal{J}} \subseteq \Delta_{A_0}^{\mathcal{J}} \times \Delta_{\mathbf{D}}$;
5. $\bigcup_{\forall d \in \mathbf{B}} V(d) \subseteq \Delta_{\mathbf{D}}$, where $V(d)$ is the value space of d;
6. $\forall d \in \mathcal{V}_D$, if $d \in \mathbf{B}$, then[2]
   (a) $d^{\mathcal{J}} = V(d)$, where $V(d)$ is the value space of d,
   (b) if $v \in L(d)$, then $(\text{“}v\text{”}\hat{}\hat{}d)^{\mathcal{J}} = L2V(d)(v)$, where $L(d)$ is lexical space of d and $L2V(d)$ is the lexical-to-value mapping of d,
   (c) if $v \notin L(d)$, then $(\text{“}v\text{”}\hat{}\hat{}d)^{\mathcal{J}}$ is undefined;

   otherwise, $d^{\mathcal{J}} \subseteq \Delta_{\mathbf{D}}$ and $\text{“}v\text{”}\hat{}\hat{}d \in \Delta_{\mathbf{D}}$.

An OWL-FA ontology $\mathcal{O}$ consists of $\mathcal{O}_1, \ldots, \mathcal{O}_\mathrm{k}$. Each $\mathcal{O}_\mathrm{i}$ consists of a TBox $\mathcal{T}_\mathrm{i}$, an RBox $\mathcal{R}_\mathrm{i}$ and an ABox $\mathcal{A}_\mathrm{i}$. An OWL-FA *TBox* $\mathcal{T}_\mathrm{i}$ is a finite set of class inclusion axioms of the form $C \sqsubseteq_\mathrm{i} D$, where C, D are OWL-FA-classes in stratum i. An interpretation $\mathcal{J}$ satisfies $C \sqsubseteq_\mathrm{i} D$ if $C^{\mathcal{J}} \subseteq D^{\mathcal{J}}$. Let R, S be OWL FA abstract properties in stratum i. An OWL-FA *RBox* $\mathcal{R}_\mathrm{i}$ is a finite set of property axioms; namely, property inclusion axioms ($R \sqsubseteq_\mathrm{i} S$), functional property axioms ($\mathsf{Func}_\mathrm{i}(R)$) and transitive property axioms ($\mathsf{Trans}_\mathrm{i}(R)$). An interpretation $\mathcal{J}$ satisfies

[2]To simplify the presentation, we do not distinguish datatype names and datatype URIrefs here.

$R \sqsubseteq_{\mathrm{i}} S$ if $R^{\mathcal{J}} \subseteq S^{\mathcal{J}}$; $\mathcal{J}$ satisfies $\mathsf{Func}_{\mathrm{i}}(R)$ if, for all $x \in \Delta_{A_{\mathrm{i}-1}}^{\mathcal{J}}$, $\sharp\{y \in \Delta_{A_{\mathrm{i}-1}}^{\mathcal{J}} \mid \langle x, y\rangle \in R^{\mathcal{J}}\} \leq 1$ ($\sharp$ denotes cardinality); $\mathcal{J}$ satisfies $\mathsf{Trans}_{\mathrm{i}}(R)$ if, for all $x, y, z \in \Delta_{A_{\mathrm{i}-1}}^{\mathcal{J}}$, $\{\langle x, y\rangle, \langle y, z\rangle\} \subseteq R^{\mathcal{J}} \rightarrow \langle x, z\rangle \in R^{\mathcal{J}}$. The semantics for datatype property inclusion axioms and functional axioms can be defined in the same way as those in OWL-DL. Like in OWL-DL, there is no transitive datatype property axioms.

Let $\mathtt{a}$, $\mathtt{b} \in \mathbf{I}$ be individuals, C_1 a class in stratum 1, R_1 an abstract property in stratum 1, l a literal, $T \in \mathcal{V}_D$ a datatype property, X, Y classes or abstract properties in stratum i, E a class in stratum i + 1 and S an abstract property in stratum i+1. An OWL-FA *ABox* $\mathcal{A}_1$ is a finite set of individual axioms of the following forms: $\mathtt{a} :_1 C_1$, called *class assertions*, $\langle \mathtt{a}, \mathtt{b}\rangle :_1 R_1$, called *abstract property assertions*, $\langle \mathtt{a}, l\rangle :_1 T$, called *datatype property assertions*, $\mathtt{a} = \mathtt{b}$, called *individual equality axioms* and, $\mathtt{a} \neq \mathtt{b}$, called *individual inequality axioms*. An interpretation $\mathcal{J}$ satisfies $\mathtt{a} :_1 C_1$ if $\mathtt{a}^{\mathcal{J}} \in C_1^{\mathcal{J}}$; it satisfies $\langle \mathtt{a}, \mathtt{b}\rangle :_1 R_1$ if $\langle \mathtt{a}^{\mathcal{J}}, \mathtt{b}^{\mathcal{J}}\rangle \in R_1^{\mathcal{J}}$; it satisfies $\langle \mathtt{a}, l\rangle :_1 T$ if $\langle \mathtt{a}^{\mathcal{J}}, l^{\mathcal{J}}\rangle \in T^{\mathcal{J}}$; it satisfies $\mathtt{a} = \mathtt{b}$ if $\mathtt{a}^{\mathcal{J}} = \mathtt{b}^{\mathcal{J}}$; it satisfies $\mathtt{a} \neq \mathtt{b}$ if $\mathtt{a}^{\mathcal{J}} \neq \mathtt{b}^{\mathcal{J}}$. An OWL-FA *ABox* $\mathcal{A}_{\mathrm{i}}$ is a finite set of axioms of the following forms: $X : E$, called *meta-class assertions*, $\langle X, Y\rangle : R$, called *meta-property assertions*, or $X =_{\mathrm{i}-1} Y$, called *meta individual equality axioms*. An interpretation $\mathcal{J}$ satisfies $X : E$ if $X^{\mathcal{J}} \in E^{\mathcal{J}}$; it satisfies $\langle X, Y\rangle : R$ if $\langle X^{\mathcal{J}}, Y^{\mathcal{J}}\rangle \in R^{\mathcal{J}}$; it satisfies $X =_{\mathrm{i}-1} Y$ if $X^{\mathcal{J}} = Y^{\mathcal{J}}$.

An interpretation $\mathcal{J}$ satisfies an ontology $\mathcal{O}$ if it satisfies all the axioms in $\mathcal{O}$. $\mathcal{O}$ is *satisfiable* (*unsatisfiable*) if there exists (does not exist) such an interpretation $\mathcal{J}$ that satisfies $\mathcal{O}$. Let C, D be OWL-FA-classes in stratum i, C is *satisfiable* w.r.t. $\mathcal{O}$ if there exist an interpretation $\mathcal{J}$ of $\mathcal{O}$ s.t. $C^{\mathcal{J}} \neq \emptyset_{\mathrm{i}}$; C subsumes D w.r.t. $\mathcal{O}$ if for every interpretation $\mathcal{J}$ of $\mathcal{O}$ we have $C^{\mathcal{J}} \subseteq D^{\mathcal{J}}$.

6.4 SPARQL-FA

SPARQL-FA is an extension of the Semantic Web standard query language SPARQL, which allows mixed Meta/TBox/ABox/RBox queries over metamodeling enabled ontology. In this section, we give a very brief overview of SPARQL-FA and its semantics. For more details, refer to [Zhao et al., 2009].

6.4.1 Syntax

Let $\mathcal{V}_O = (\mathcal{V}_{cls_i}, \mathcal{V}_{ap_i}, \mathcal{V}_D, \mathcal{V}_{dp}, \mathcal{V}_{ind})$ be an OWL FA vocabulary. A SPARQL-FA query atom q is of the form:

$q \leftarrow$ FA:Type$_i(a, C)$ | FA:PropertyValue$_i(a, p, v)$ | FA:EquivalentClass$_i(C_1, C_2)$ |
FA:SubClassOf$_i(C_1, C_2)$ |FA:DisjointWith$_i(C_1, C_2)$ |
FA:EquivalentProperty$_i(p_1, p_2)$ | FA:SubPropertyOf$_i(p_1, p_2)$ |
FA:Functional$_i(p)$ | FA:InverseFunctional$_i(p)$ | FA:InverseOf$_i(p)$ |
FA:ObjectProperty$_i(p)$ | FA: DatatypeProperty$_i(p)$ |
FA:Domain$_i(p, C)$ | FA:Range$_i(p, C)$ | FA:Transitive$_i(p)$ |
FA:Symmetric$_i(p)$ | FA:Annotation(s, p_a, o)

where $a, b \in \mathcal{V}_{uri} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{var}, v \in \mathcal{V}_{uri} \cup \mathcal{V}_{lit} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{var}, p, q \in \mathcal{V}_{uri} \cup \mathcal{V}_{var}, C, D \in S_c \cup \mathcal{V}_{var}, s \in \mathcal{V}_{uri}, p_a \in \mathcal{V}_{ap_i}, o \in \mathcal{V}_{uri} \cup \mathcal{V}_{lit}$ and i is a layer (stratum in OWL-FA). A SPARQL-FA query Q is a finite set of SPARQL-FA query atoms and the query is interpreted as the conjunction of the elements in the set.

6.4.2 Semantics

We define an evaluation $\alpha : \mathcal{V}_{ind} \cup \mathcal{V}_{bnode} \cup \mathcal{V}_{lit} \rightarrow \Delta^{\mathcal{J}}$ to be a mapping from the individual names, bnodes, and literals used in the query to the elements of interpretation domain $\Delta^{\mathcal{J}}$ with the requirement that $\alpha(a) = a^{\mathcal{J}}$ if $a \in \mathcal{V}_{ind}$ or $a \in \mathcal{V}_{lit}$. The interpretation $\mathcal{J}$ satisfies a semi-ground query atom q w.r.t. μ (denoted as $\mathcal{I} \models \alpha q$) if q is compatible with $\mathcal{V}_O$ and the corresponding condition listed in Table 2 is met. Note that, the query atoms ObjectProperty and DatatypeProperty are not in this table because they are satisfied by every interpretation as long as they are compatible with $\mathcal{V}_O$.

The interpretation $\mathcal{J}$ satisfies a query $Q = q_1 \wedge \ldots \wedge q_n$ w.r.t. an evaluation σ(written $\mathcal{J} \models \alpha q$) if $\mathcal{J} \models \alpha q_i$ for every $i = 1, ..., n$. $\mathcal{J}$ satisfies a query Q if there exists an evaluation σ such that $\mathcal{I} \models Q$. Q is a logical consequence of the ontology $\mathcal{O}$ if the query is satisfied by every model of $\mathcal{O}$ A solution to a SPARQL-FA query $Q = q_1 \wedge \ldots \wedge q_n$ w.r.t. an OWL-FA ontology $\mathcal{O}$ is a *variablemapping* $\mu : \mathcal{V}_{var} \rightarrow \mathcal{V}_{uri} \cup \mathcal{V}_{lit}$ such that when all the variables in Q are substituted with the corresponding value from μ a semi-ground query $\mu(Q)$ compatible with $\mathcal{V}_O$ and $\mathcal{O} \models \mu(Q)$.

Abstract syntax	Condition on interpretation
FA:Type$_i(a, C)$	$\alpha(a^{\mathcal{J}}) \in C_1^{\mathcal{J}}$
FA:PropertyValue$_i(a, p, v)$	$\langle \alpha(a^{\mathcal{J}}), \alpha(v^{\mathcal{J}}) \rangle \in P^{\mathcal{J}}$
FA:SubClassOf$_i(C_1, C_2)$	$C_1^{\mathcal{J}} \subseteq C_2^{\mathcal{J}}$
FA:EquivalentClass$_i(C_1, C_2)$	$C_1^{\mathcal{J}} = C_2^{\mathcal{J}}$
FA:DisjointWith$_i(C_1, C_2)$	$C_1^{\mathcal{J}} \cap C_2^{\mathcal{J}}$
FA:EquivalentProperty$_i(p, q)$	$p^{\mathcal{J}} = q^{\mathcal{J}}$
FA:SubPropertyOf$_i(p, q)$	$p^{\mathcal{J}} \subseteq q^{\mathcal{J}}$
FA:InverseOf$_i(p)$	$p^{\mathcal{J}} = (p^{\mathcal{J}})^-$
FA:Domain$_i(p, C)$	$p^{\mathcal{J}} \subseteq C^{\mathcal{J}} \times \Delta_{\mathcal{D}}^{\mathcal{J}}$
FA:Range$_i(p, C)$	$p^{\mathcal{J}} \subseteq \Delta_{\mathcal{D}}^{\mathcal{J}} \times C^{\mathcal{J}}$
FA:Functional$_i(p)$	$\{x \in \Delta_{A_{i-1}}^{\mathcal{J}} \mid \sharp\{y \mid \langle x, y \rangle \in P^{\mathcal{J}}\} \leq 1\}$
FA:InverseFunctional$_i(p)$	$\{y \in \Delta_{A_{i-1}}^{\mathcal{J}} \mid \sharp\{x \mid \langle x, y \rangle \in P^{\mathcal{J}}\} \leq 1\}$
FA:Transitive$_i(p)$	$\{x, y, z \in \Delta_{A_{i-1}}^{\mathcal{J}} \mid \{\langle x, y \rangle, \langle y, z \rangle\} \subseteq R^{\mathcal{J}} \rightarrow \langle y, z \rangle \in R^{\mathcal{J}}\}$
FA:Symmetric$_i(p)$	$\langle x, y \rangle \in P^{\mathcal{J}} \rightarrow \langle y, x \rangle \in P^{\mathcal{I}}$
FA:Annotation(s, p_a, o)	$\langle s, o \rangle \in P_a^{\mathcal{J}}$

Table 2: Condition on interpretation of SPARQL-FA abstract syntax

7 Querying the Combined Metamodel

After eliciting the requirements for querying the combined metamodel(Sect.4) and analyzing existing approaches (Sect.6 and Sect.5), we analyze in this section how existing approaches relate to the requirements.

The following sections explores existing approaches to meet the requirements and presents extensions to cover open issues.

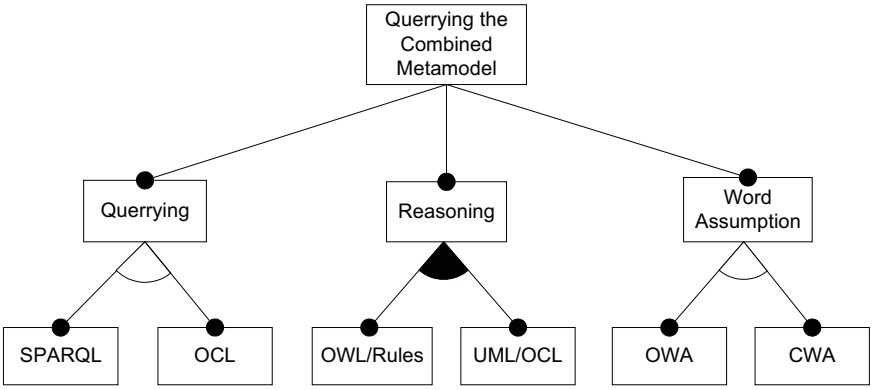

Figure 8: Feature Model of Querying the Combined Metamodel.

7.1 Combining Existing Approaches

Some requirements presented in Sect.4 are realized by existing approaches. For example the definition of functions and the usage of built-in functions are provided by OCL, whereas SPARQL provides an powerful language to query resources in triple pattern, allowing for retrieving the syntax of the language as well as its assertions.

The Unique Name Assumption (UNA) is assumed in OCL and may be mimicked with OWL as well, using constructs like owl:AllDifferent and owl:distinctMembers. Incomplete knowledge can be handled using the open world assumption (OWA) whereas complete knowledge is handled by the closed world assumption (CWA).

The Table 4 presents a relation between the requirements and the approaches discussed above. We call these approaches *features* of the query solution for the combined metamodel. A combination of these features reflects configurations for querying the combined metamodel. These combinations are modeled in the feature model depicted in Fig. 8.

Requirements	Features
Incomplete Knowledge	OWA
Complete Knowledge	CWA
Functions and Built-in Functions	OCL
Built-in Functions	SPARQL
Transitive Closure	OWL/Rules
UNA	OWL/Rules, UML/OCL
Semantics	OWL/Rules, UML/OCL

Table 3: Mapping requirements to features.

The feature model reveals the different possible choices for querying the combined metamodel and can also be used as a taxonomy to categorize the existing approaches.

The relations between the features are constrained by the following composition rules:

```
SPARQL requires OWL/Rules;
SPARQL is is mutually exclusive with UML/OCL;
```

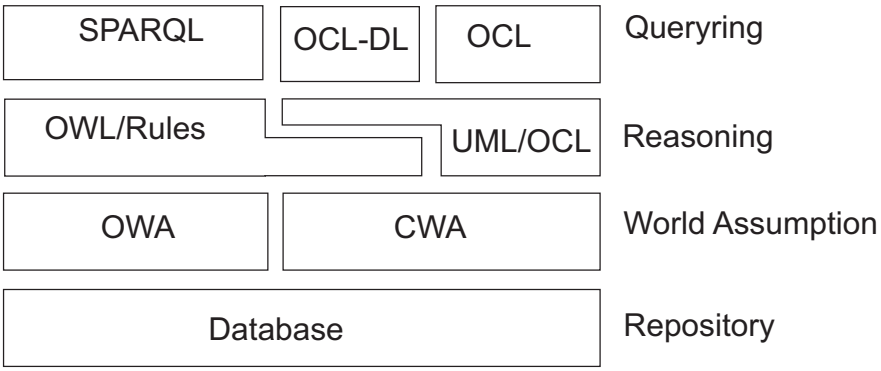

Figure 9: Conceptual Architecture for Querying the Combined Metamodel.

```
OCL requires UML/OCL;
UML/OCL requires CWA;
UML/OCL is mutually exclusive with OWA.
```

Let us analyze these constraints. SPARQL queries require reasoning with OWL or dl rules and does not work with UML/OCL reasoning. On the other hand, OCL queries requires reasoning with UML/OCL and reasoning with UML/OCL requires CWA exclusively.

After ruling out the configurations inconsistent with the constraints above, we have the following possible configurations for querying the combined metamodel:

1. SPARQL, OWL-RULES, OWA (Sect.7.1.1);
2. SPARQL, OWL-RULES, CWA (Sect.7.1.2);
3. OCL, UML-OCL, OWL-RULES, CWA (Sect.7.1.3);
4. OCL, UML-OCL, CWA (Sect.7.1.4).

Figure 9 illustrates how the configurations listed above may take place when querying the combined metamodel. Three out of four configurations can be found by adopting current approaches. One configuration involves a combination of different semantics and reasoning services. All these configurations are described in the following sections.

These four combinations cover all needs for querying the combined metamodel considering the features and restrictions aforementioned. They allow OWA as well as CWA and includes capabilities for reasoning with OWL, UML/OCL and an hybrid approach including both.

7.1.1 Using SPARQL over OWL with OWA

Among existing RDF based query languages for the semantic web, SPARQL is the W3C recommendation. It is based on triples patterns and allows for querying the vocabulary and the assertions of a given domain.

Restrictions on the SPARQL language, i.e., entailment regimes, allow for querying OWL ontologies, including TBox, RBox and ABox. One implementation is that is SPARQL-DL

[Sirin and Parsia, 2007](Sect.5.4). Another one is SPARQL-FA, which implemented on the TrOWL, the infrastructure for the MOST project(Sect.6.4).

SPARQL-DL enables querying OWL ontologies using the Open World Assumption. It is current available together with the Pellet Reasoner. It offers a robust query language with DL reasoning support.

SPARQL-FA allowing mixed Meta/TBox/ABox and RBox queries over metamodeling enabled ontology. Currently, we provide an efficient algorithm for query over SPARQL-FA$_1$, a sub-language of SPARQL-FA that allows variables to be evaluated as not only individuals, but also classes and properties. The algorithm is sound and complete w.r.t. the DL-Lite semantic approximation of a given OWL DL ontology.

7.1.2 Using SPARQL over OWL with CWA

The SPARQL language has been explored in combination with closed-world reasoning as well. Examples of current applications are SPARQL++[Polleres et al., 2007]. SPARQL++ extends SPARQL by supporting aggregate functions, CONSTRUCT query form and built-ins. SPARQL++ queries can be formalized in HEX Programs or description logic programs. SPARQL++ covers only an subset of RDF(S) and how it could be extended towards OWL is still an open issue.

OWL reasoning with closed world assumption is possible by adopting epistemic operators [Grimm and Motik, 2005] [Donini et al., 1998]. Thus, SPARQL-FA can be used with CWA covering the whole expressivity of OWL-DL and OWL-FA. An implementation of that is described in MOST project's deliverable D3.2 [Zhao et al., 2009].

7.1.3 Using OCL over UML/OCL with CWA

This is the standard application of OCL as query language. The KB may be described using UML and OCL constraints. Query operations may be defined and used as helpers for OCL queries and constraints. Default values as well as initial and derived values can be defined by using UML and OCL.

UML/OCL reasoning differs from OWL/DL Rules reasoning in the sense that the former includes behavioral features like operations whereas reasoning with OWL is strictly logical.

Additionally to OCL, GreQL may also be used as query language over UML/OCL models with CWA. In order to do so, a translation from the UML/OCL technical space to grUML/-GreQL technical space is additionally required.

7.1.4 Using OCL over UML/OCL and OWL with CWA: OCL-DL

In some cases, a combination of UML/OCL and OWL/DL Rules is needed. For example, one wants to define complex class descriptions or reuse existing ones. To compute reasoning tasks like *realization, instance checking,* and *retrieval*, automated reasoners implement sound and complete calculi like the tableau algorithms and guarantees decidability. On the other side, to make usage of behavioral features like query operations, helpers and built-ins, UML/OCL reasoning may come into play.

None of the existing approaches allow such combination, except from TwoUse OCL. However, TwoUse OCL built-in operations allow only the reasoning tasks *instance checking*, which verify whether an object satisfies the requirements to be an instance of a given class, and *realization*, which evaluates to the most specific named classes for a given object.

For that purpose, we present a hybrid solution, combining reasoning with OWL/DL rules and UML/OCL in a cascaded way. The OCL-DL box in Fig. 9 illustrates such a combination. Queries are written in OCL-like notation and a subset of these queries is translated into SPARQL and execute against an OWL/DL rules KB (assuming CWA). The results are used as input for the UML/OCL that allows the usage of helpers, query operations and built-ins defined for the classes of the KB to execute the remaining subset of the original query.

In the next section we present how this OCL-like notation can be modeled with the integrated metamodel.

7.2 OCL-DL Language Description

Combining UML/OCL and OWL/DL rule for querying the integrated metamodel is part of an effort towards using OCL for the semantic web, namely OCL-DL[Silva Parreiras, 2008]. OCL-DL allows the specification of both queries and constraints, namely:

- body of query operations;
- invariants on classes;
- initial values;
- derivation rules;
- pre- and post conditions on operations;

Figure 10 displays OCL-DL according to the different layers of the conceptual architecture depicted in Fig. 9. In the repository layer, the MOST KB comprises the OWL/DL rules KB and the UML/OCL KB. Adapters are used to query the MOST KB using OWL/DL rules reasoning. The output is used as input for OCL expressions. These OCL expressions may be chained with other OCL expressions that use again OWL Reasoning. The final result is delivered by the OCL-DL engine.

In the next sections we detail different aspects of OCL-DL like abstract syntax according to MOST Reference Layer, concrete syntax, and preliminary translations into SPARQL.

7.2.1 Relation with Metamodeling

OCL-DL extends the integrated metamodel by allowing to model useful aspects of a domain, as listed above. These aspects take the form of expressions associated with different elements of the combined metamodel.

More specifically, OCL-DL expressions are associated with classes, operations and attributes. Invariants are in the context of classes whereas initialization and derivation expressions are in the context to attributes. Body, definition, pre and post conditions are in the context of operations.

7.2.2 Abstract Syntax

Since we use OCL as concrete syntax, the OCL-DL metamodel is based on the OCL metamodel [OMG, 2005]. In the following paragraphs we describe shortly the OCL metamodel. Please consult the OCL specification for more detailed information.

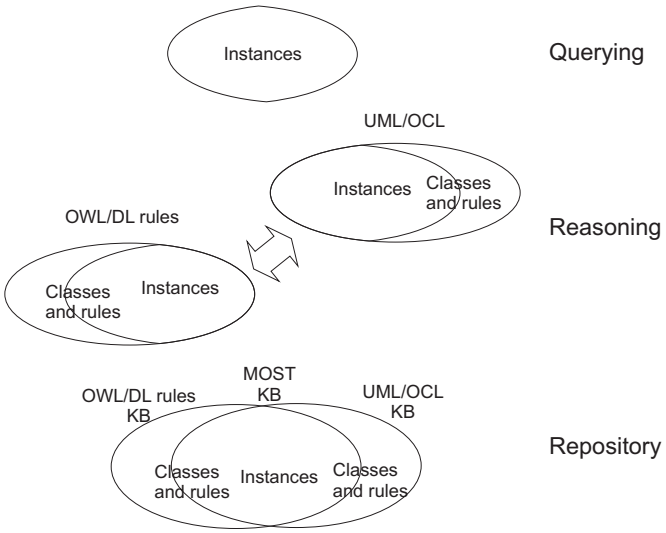

Figure 10: Hybrid approach with integrated reasoning: OCL-DL.

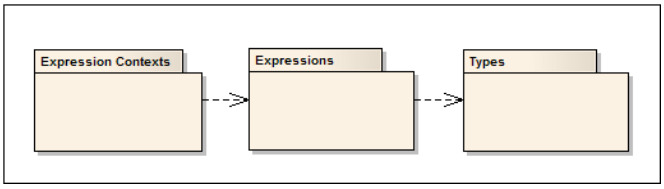

Figure 11: Package OCL.

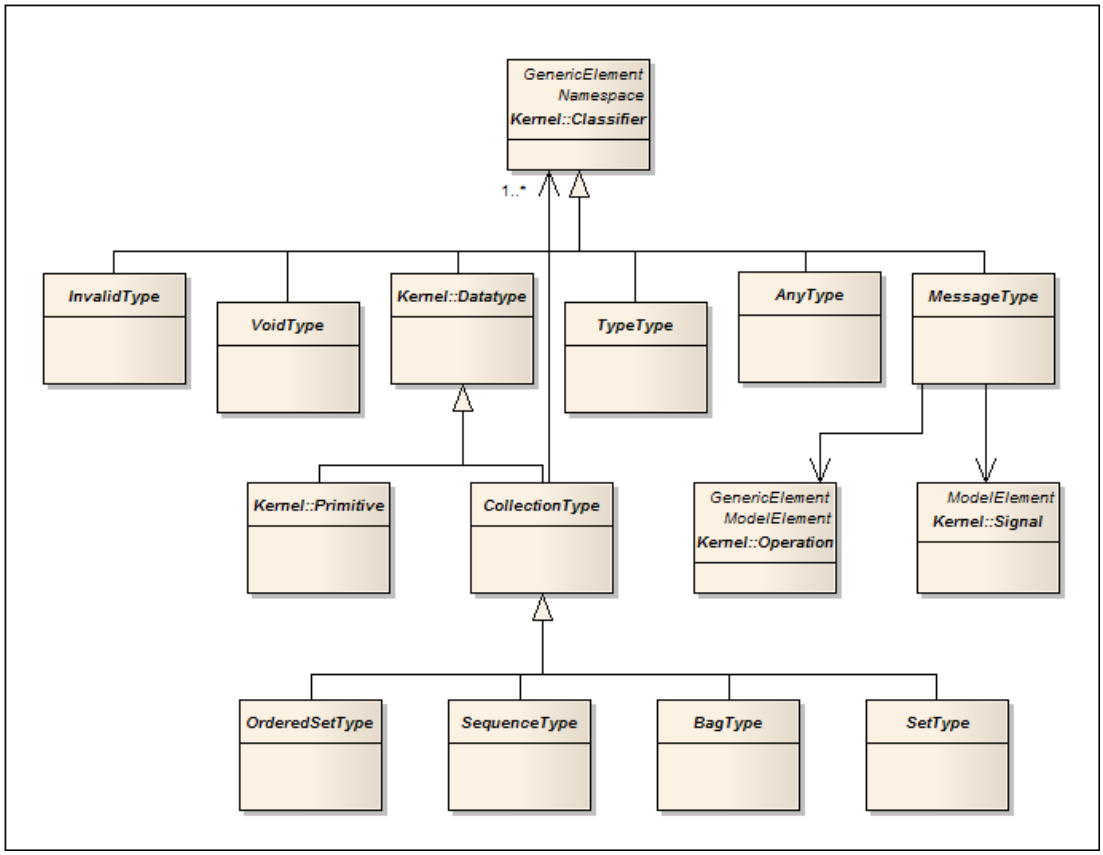

Figure 12: Package OCL::Types.

The OCL package (Fig.11) provides the classes necessary to support the specification of operation queries, invariants, pre- and post conditions.

The subpackages Types(Fig. 12) and Expressions(Fig. 14) are adopted from the OCL specification
[OMG, 2005]. Since OCL is a typed language, every expression has a type and evaluates to a type.

The subpackage Context (Fig. 13) enables the specification of context of OCL expressions, since context specification in the OCL specification is embedded in UML. Here, we follow the approach of Akehurst and Patrascoiu [Akehurst and Patrascoiu, 2004].

From the point of view of the abstract syntax, OCL-DL does not extend much OCL. In order to be able to use OWL constructs and to derive this kind of constructs back, the class MClassifier is added to the OCL::Type package (Fig. 15).

Well Formedness Rules All OCL-DL operations make use of reasoning services, i.e., they rely on inferences over complex descriptions of concepts and instances. In order to include these operations, we extend the OCL library by adding a new M1 instance of the OCL metaclass `AnyType` called `OwlAny`. While all classifiers except collections conform to `OclAny` in OCL, all

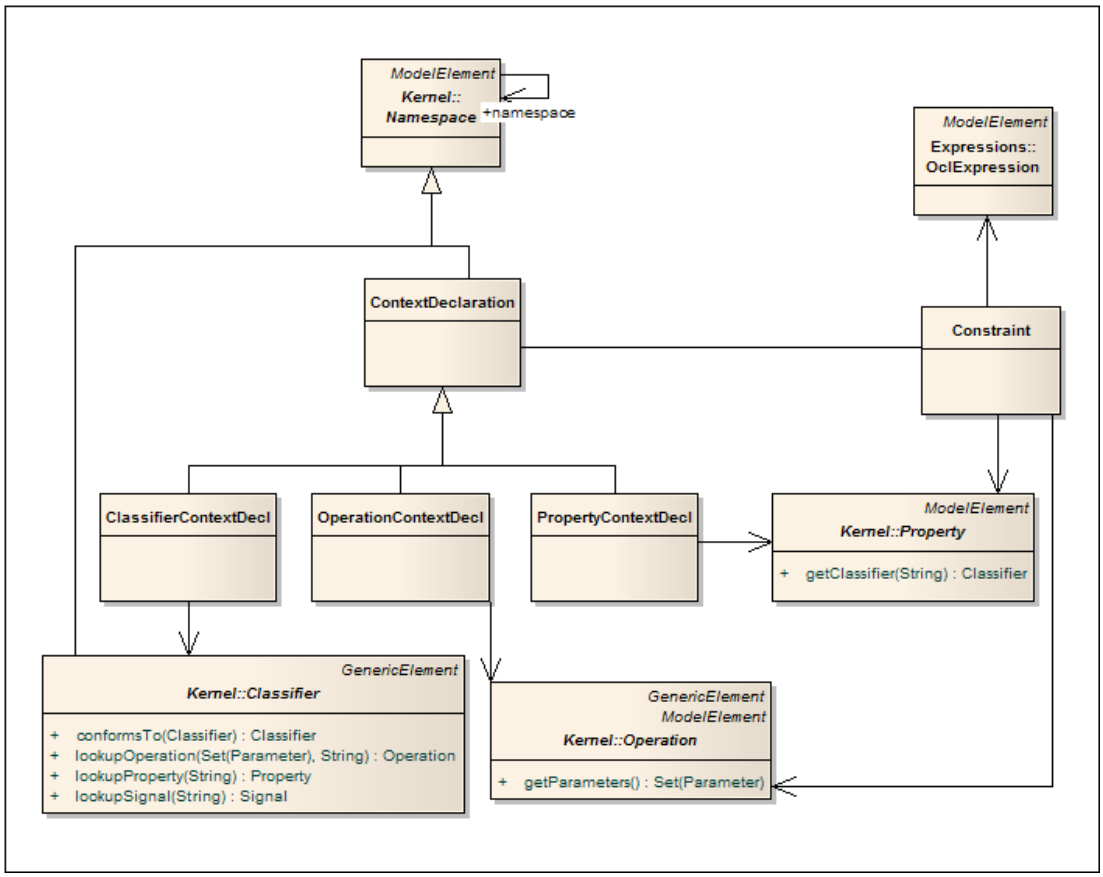

Figure 13: Package OCL::Expression Contexts.

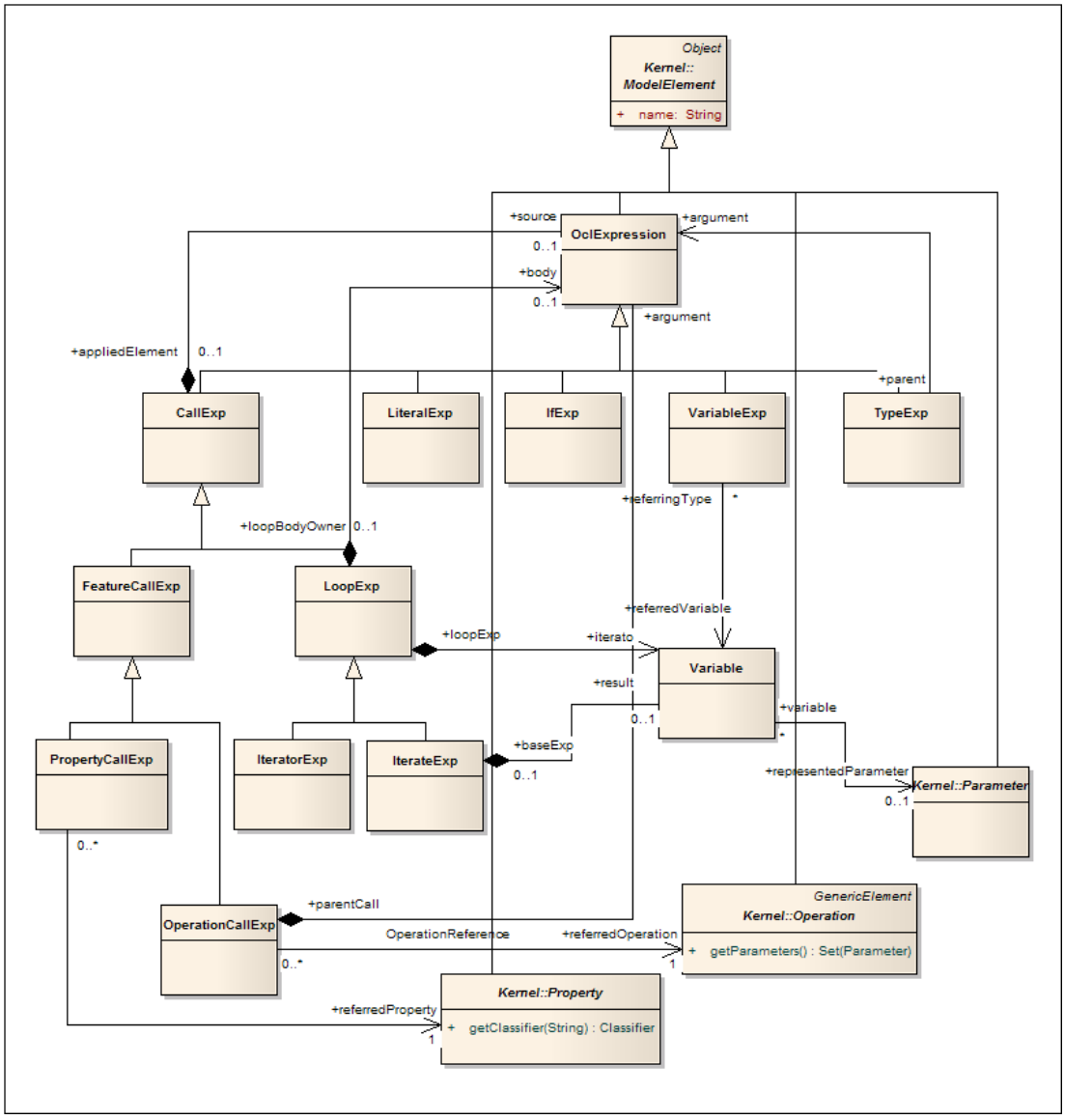

Figure 14: Package OCL::Expressions.

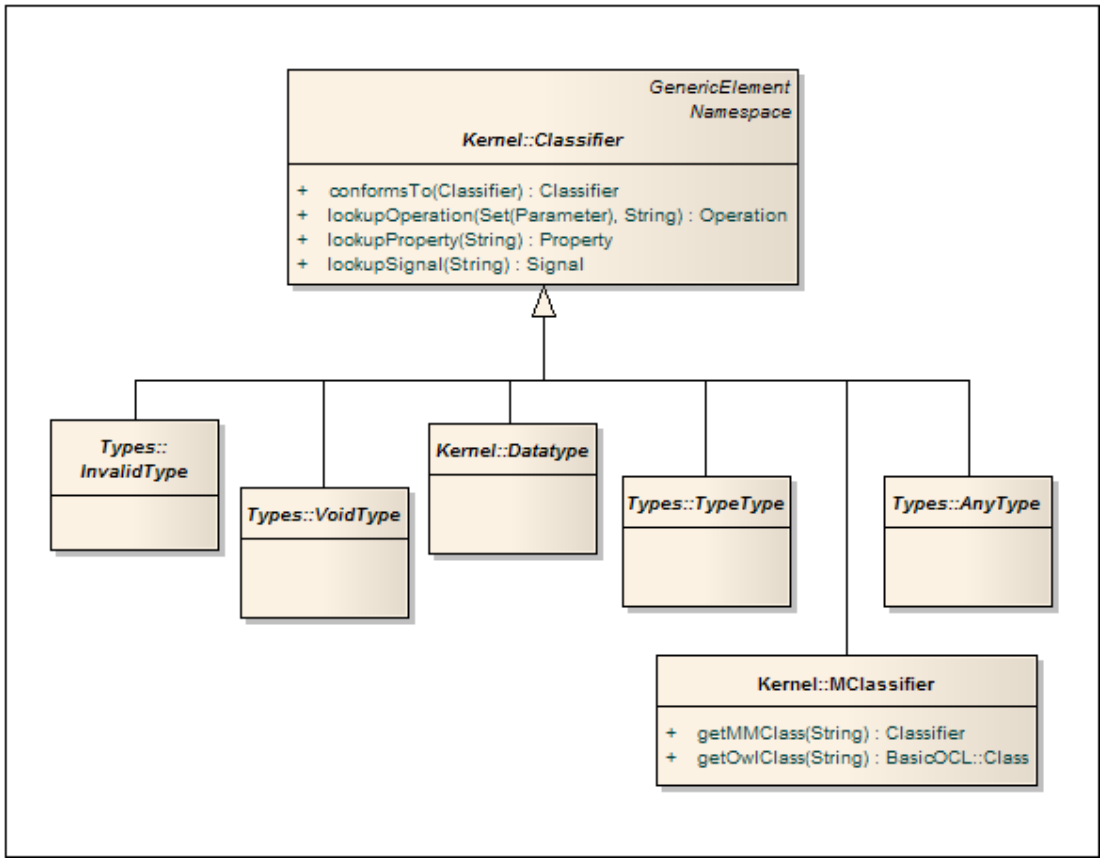

Figure 15: Package OCL-DL::Types.

MClassifiers conforms to `OwlAny`. Thus, `OwlAny` acts as a supertype for MClassifier, that inherit all operations of `OwlAny`.

```
-- All MClassifiers conform to OwlAny. context MClassifier inv:
   AnyType.allInstances()->forAll(p | (p.name = 'OwlAny')
      implies self.conformsTo(p))
```

7.2.3 Model Libraries

The model libraries define a number of datatypes, class identifiers and operations that must be included in the implementation of OCL-DL. These constructs are instances of an abstract syntax class. The foundation library exists at the M1 level, while the abstract syntax (metamodel) exists at M2 level. The foundation library is composed of the XML Schema Datatypes library, the RDF library, the OWL library and the OCL library.

Examples of M1 objects of the XML Schema datatypes library are the datatypes `gDay`, `gMonth` and `gYear`, having the M2 class `RDFS::RDFSDatatype` as metaclass. In the RDF library, for example, the M1 object `nil` has the M2 class `RDFS::RDFList` as metaclass. In the OWL library, interesting M1 objects are `Thing` and `Nothing`, both having the M2 class OWL::OWLClass as metaclass. For a complete description of the foundation library for RDF and OWL, please refer to the ODM specification [OMG, 2008].

An example of M1 object of the extended OCL library is the construct `OclAny`. All types inherit the properties and operations of `OclAny`, except collection types. This invariant allows for attributing predefined operations to classes. Please refer to the OCL specification [OMG, 2005] for a comprehensive description of the OCL library.

7.2.4 Mapping to existing reasoning technology

We apply a translational semantics from OCL-DL into SPARQL-FA (Sect. 6.4). Indeed, SPARQL-FA is a powerful language enable to express different aspects of OCL. The following table summaries the translational semantics of OCL-DL into SPARQL-FA.

OCL-DL expression	SPARQL query form
body or def	SELECT
init or derive	CONSTRUCT
inv	ASK
pre- andpost	ASK

Table 4: Metrics for Object-Oriented Design on Package elements

Since the focus of this deliverable is on a query language for the integrated metamodel, we concentrate on translation OCL-DL query operations into SPARQL queries. More information about the other elements of OCL-DL can be found in [Silva Parreiras, 2008].

For sake of illustration, in Annex A, we give a few examples of equivalent queries in OCL and in SPARQL. A comprehensive set of transformations will be published in Deliverable D1.3.

7.2.5 Complexity

A fundamental issue in every query language is the complexity of query evaluation. Given that we apply a translational semantics from OCL-DL into SPARQL. Roughly speaking, the

complexity of OCL-DL query is equivalent to the complexity of SPARQL query. The complexity of evaluating graph pattern expressions is described in Table 5 [Pérez et al., 2006]

Graph pattern expressions	Complexity
Fixed patterns (data complexity)	LOGSPACE
AND and FILTER	PTIME
AND, FILTER and OPT	PSPACE-complete
AND, FILTER and UNION	NP-complete

Table 5: OCL-DL complexities

7.3 Summary

This section analyzes how current approaches can be used to query the integrated metamodel and possible new combinations. Existing query languages like SPARQL, OCL and GreQL may be used according to different requirements whereas a combination of OCL and SPARQL is used when applying OWL and UML reasoning.

8 Conclusion

When querying ontologies and class based models in an integrated way, requirements like word assumption, support to functions and built-ins must be taken into consideration. Current query languages use different strategies to accomplish these requirements. However, combinations of existing languages are needed to cover all the requirements.

This deliverable describes the state of the art on querying ontologies and class based models. Existing query languages are analyzed against the requirements. Solutions for querying and validating integrated models are based on existing solutions and new combinations of them (Sect.7). The integrated query language has an OCL like notation (Sect.7.2). Initial equivalences between OCL-DL and SPARQL-FA are presented and an comprehensive set of mappings will be defined in Deliverable D1.3.

List of Figures

List of Tables

References

[Akehurst and Patrascoiu, 2004] Akehurst, D. H. and Patrascoiu, O. (2004). Ocl 2.0 - implementing the standard for multiple metamodels. *Electr. Notes Theor. Comput. Sci.*, 102:21–41.

[Beckert et al., 2002] Beckert, B., Keller, U., and Schmitt, P. H. (2002). Translating the object constraint language into first-order predicate logic. In Autexier, S. and Mantel, H., editors, *In Proceedings of the Second Verification Workshop: VERIFY 2002, July 25-26, 2002, Copenhagen, Denmark*, volume 02-07 of *DIKU technical report*. DIKU.

[Bildhauer, 2006] Bildhauer, D. (2006). Ein Interpreter für GReQL 2 - Entwurf und prototypische Implementation. Diplomarbeit, Universität Koblenz-Landau, Institut für Softwaretechnik.

[Bildhauer and Ebert, 2008] Bildhauer, D. and Ebert, J. (2008). Querying software abstraction graphs. In *Working Session on Query Technologies and Applications for Program Comprehension (QTAPC 2008), collocated with ICPC 2008*.

[Brucker and Wolff, 2002] Brucker, A. D. and Wolff, B. (2002). A proposal for a formal ocl semantics in isabelle/hol. In *TPHOLs '02: Proceedings of the 15th International Conference on Theorem Proving in Higher Order Logics*, pages 99–114, London, UK. Springer-Verlag.

[Brucker and Wolff, 2006] Brucker, A. D. and Wolff, B. (2006). The HOL-OCL book. Technical Report 525, ETH Zurich.

[Chandra and Merlin, 1977] Chandra, A. K. and Merlin, P. M. (1977). Optimal implementation of conjunctive queries in relational data bases. In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90, New York, NY, USA. ACM.

[Dong et al., 1999] Dong, G., Libkin, L., Su, J., and Wong, L. (1999). Maintaining the Transitive Closure of Graphs in SQL. *In Int. J. Information Technology*, 5.

[Donini et al., 1998] Donini, F. M., Lenzerini, M., Nardi, D., Nutt, W., and Schaerf, A. (1998). An epistemic operator for description logics. *Artif. Intell.*, 100(1-2):225–274.

[Ebert et al., 2002] Ebert, J., Kullbach, B., Riediger, V., and Winter, A. (2002). GUPRO-Generic Understanding of Programs. *Electronic Notes in Theoretical Computer Science*, 72(2):59–68.

[Grimm and Motik, 2005] Grimm, S. and Motik, B. (2005). Closed world reasoning in the semantic web through epistemic operators. In *Proceedings of Workshop on OWL Experiences and Directions - OWLED 2005, November 11-12, Galway, Ireland*.

[KAON2, 2005] KAON2 (2005). http://kaon2.semanticweb.org/.

[Kline and Kline, 2000] Kline, K. and Kline, D. (2000). *SQL in a Nutshell*. O'Reilly.

[Miller et al., 2001] Miller, J., Mukerji, J., et al. (2001). Model Driven Architecture (MDA). *Object Management Group, Draft Specification ormsc/2001-07-01, July*, 9.

[Motik et al., 2007] Motik, B., Horrocks, I., and Sattler, U. (2007). Bridging the gap between owl and relational databases. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 807–816, New York, NY, USA. ACM.

[OMG, 2005] OMG (2005). *Object Constraint Language Specification, version 2.0*. Object Modeling Group.

[OMG, 2008] OMG (2008). *Ontology Definition Metamodel*. Object Modeling Group.

[Pan et al., 2005] Pan, J. Z., Horrocks, I., and Schreiber, G. (2005). OWL FA: A Metamodeling Extension of OWL DL. In *Proceeding of the International workshop on OWL: Experience and Directions (OWL-ED2005)*.

[Parreiras and Walter, 2008] Parreiras, F. S. and Walter, T. (2008). Report on the combined metamodel. Deliverable ICT216691/UoKL/WP1-D1.1/D/PU/a1, University of Koblenz-Landau. EU FP7 STREP MOST Project number ICT-2008-216691.

[Pérez et al., 2006] Pérez, J., Arenas, M., and Gutierrez, C. (2006). Semantics and complexity of sparql. In *In Proc. of International Semantic Web Conference (ISWC2006)*, pages 30–43.

[Polleres et al., 2007] Polleres, A., Scharffe, F., and Schindlauer, R. (2007). Sparql++ for mapping between rdf vocabularies. In Meersman, R. and Tari, Z., editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I*, volume 4803 of *Lecture Notes in Computer Science*, pages 878–896. Springer.

[Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). Sparql query language for rdf. W3c recommendation, W3C.

[Richters, 2002] Richters, M. (2002). *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universität Bremen.

[Roe et al., 2003] Roe, D., Broda, K., and Russo, A. (2003). Mapping UML Models incorporating OCL Constraints into Object-Z. Technical report.

[Schmitt, 2001] Schmitt, P. H. (2001). A model theoretic semantics for ocl. In *In Proc. of IJCAR Workshop on Precise Modelling and Deduction for Object-oriented Software Development (PMD.*

[Shalloway and Trott, 2001] Shalloway, A. and Trott, J. R. (2001). *Design patterns explained: a new perspective on object-oriented design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Silva Parreiras, 2008] Silva Parreiras, F. (2008). Marrying ontologies and model driven engineering technical spaces: The twouse approach. In Mylopoulos, J., Cabot, J., and Guizzardi, G., editors, *In the PhD Workshop of the 27th International Conference on Conceptual Modeling (ER2008), 20-23 October, Barcelona, Spain.*

[Silva Parreiras et al., 2008a] Silva Parreiras, F., Staab, S., Schenk, S., and Winter, A. (2008a). Model driven specification of ontology translations. In Lia, Q., Spaccapietra, S., and Yu, E., editors, *Conceptual Modeling - ER 2008, 27th International Conference on Conceptual Modeling, Barcelona, Spain, October 23-26, 2008, Proceedings*, volume 5231 of *Lecture Notes in Computer Science*. Springer.

[Silva Parreiras et al.,] Silva Parreiras, F., Staab, S., and Winter, A. TwoUse: Integrating UML models and OWL ontologies. Technical Report 16/2007, University of Koblenz-Landau. Available at `http://isweb.uni-koblenz.de/Projects/twouse/tr162007.pdf`.

[Silva Parreiras et al., 2008b] Silva Parreiras, F., Staab, S., and Winter, A. (2008b). Improving design patterns by description logics: A use case with abstract factory and strategy. In Kühne, T., Reisig, W., and Steimann, F., editors, *Modellierung 2008, 12.-14. März 2008, Berlin*, number 127 in LNI. GI.

[Sirin and Parsia, 2007] Sirin, E. and Parsia, B. (2007). Sparql-dl: Sparql query for owl-dl. In Golbreich, C., Kalyanpur, A., and Parsia, B., editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Smith, 2000] Smith, G. (2000). *The Object-Z specification language*. Kluwer Academic Publishers, Norwell, MA, USA.

[T and Su, 2000] T, G. D. and Su, J. (2000). Incremental maintenance of recursive views using relational calculus/sql. *SIGMOD Record*, 29:2000.

[Vardi, 1982] Vardi, M. Y. (1982). The complexity of relational query languages (extended abstract). In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146, New York, NY, USA. ACM.

[Vrandecic et al., 2006] Vrandecic, D., Volker, J., Haase, P., Tran, D. T., and Cimiano, P. (2006). A metamodel for annotations of ontology elements in owl dl. In Sure, Y., Brockmans, S., and Jung, J., editors, *Proceedings of the 2nd Workshop on Ontologies and Meta-Modeling*, Karlsruhe, Germany. GI Gesellschaft fÃ$\frac{1}{4}$r Informatik.

[Zhao et al., 2009] Zhao, Y., Pan, J. Z., Thomas, E., Jekjantuk, N., Ren, Y., Gröner, G., Utz, W., and Ke, P. (2009). Initial prototype of language transformations. Deliverable ICT216691/UoKL/WP3-D3.2/D/PU/b1, University of Aberdeen. EU FP7 STREP MOST Project number ICT-2008-216691.

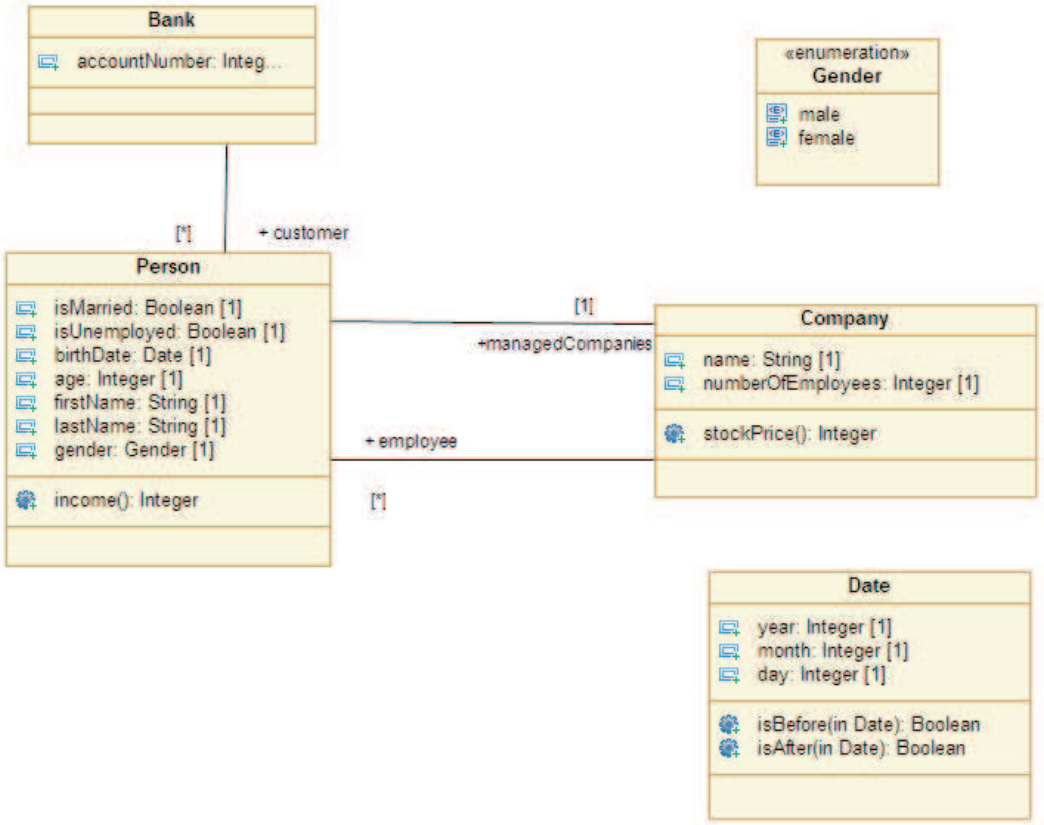

Figure 16: Domain to illustrate OCL-DL.

Annex A: Examples of Equivalent Queries OCL-DL and SPARQL-FA

Figure 16 depicts the context of the OCL expressions described below. The correspondent OWL ontology follows the example queries.

```
context Company
  self.numberOfEmployees > 50

  into

PREFIX : <http://www.example.org/example#>
PREFIX op: <http://www.w3.org/2005/xpath-functions>
SELECT ?numberOfEmployees
WHERE  {
   ?self numberOfEmployess ?x .
   FILTER op:numeric-less-than(?x 50)
}
```

```
context Company
self.employee->select(age > 50)->notEmpty()

into

PREFIX : <http://www.example.org/example#>
PREFIX op: <http://www.w3.org/2005/xpath-functions>
ASK
WHERE {
  ?self employee ?i_Person .
  ?i_Person age ?v_age .
  FILTER op:numeric-less-than(?v_age 50)
}
```

```
context Company
self.employee.birthDate

into

PREFIX : <http://www.example.org/example#>
SELECT ?birthDate
WHERE {
  ?self employee ?v_employee
  ?v_employee birthDate ?birthDate
}
```

```
context Company
  inv: self.employee->exists( forename = 'Jack' )

into

PREFIX : <http://www.example.org/example#>
ASK
WHERE {
  ?self employee ?e .
  ?e forname "Jack"
}
```

```
context Company
  inv: self.employee->forAll( age <= 65)

into
```

```
PREFIX : <http://www.example.org/example#>
ASK
WHERE{
  ?self employee ?e1 .
  OPTIONAL {
     ?e2 age ?a .
     FILTER (?a <= 65)
     . FILTER (?e1 = ?e2)
  }
  FILTER (!bound(?e2))
} is false

}
```

Ontology used in the example in N3 notation.

```
@prefix :         <http://www.example.org/example#> .
@prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl2xml: <http://www.w3.org/2006/12/owl2-xml#> .
@prefix owl:      <http://www.w3.org/2002/07/owl#> .
@prefix xsd:      <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix example: <http://www.example.org/example#> .

:age
       a        owl:DatatypeProperty ;
       rdfs:domain :Person .

:Date
       a        owl:Class ;
       rdfs:subClassOf
                [ a        owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :year
                ] ;
       rdfs:subClassOf
                [ a        owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :day
                ] ;
       rdfs:subClassOf
                [ a        owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :month
                ] .

:Bank
       a        owl:Class ;
       rdfs:subClassOf
                [ a        owl:Restriction ;
```

```
                owl:cardinality "1"^^xsd:nonNegativeInteger ;
                owl:onProperty :accountNumber
              ] .

:gender
      a       owl:DatatypeProperty ;
      rdfs:domain :Person .

:accountNumber
      a       owl:DatatypeProperty ;
      rdfs:domain :Bank .

:manager
      a       owl:ObjectProperty ;
      rdfs:domain :Company ;
      rdfs:range :Person .

:managedCompanies
      a       owl:ObjectProperty ;
      rdfs:domain :Person ;
      rdfs:range :Company .

:year
      a       owl:DatatypeProperty ;
      rdfs:domain :Date .

:employee
      a       owl:ObjectProperty ;
      rdfs:domain :Company ;
      rdfs:range :Person .

:month
      a       owl:DatatypeProperty ;
      rdfs:domain :Date .

:name
      a       owl:DatatypeProperty ;
      rdfs:domain :Company ;
      rdfs:range xsd:string .

:employer
      a       owl:ObjectProperty ;
      rdfs:domain :Person ;
      rdfs:range :Company .

:isMarried
      a       owl:DatatypeProperty ;
      rdfs:domain :Person .

:customer
      a       owl:ObjectProperty ;
```

```
        rdfs:domain :Bank ;
        rdfs:range :Person .


:birthDate
        a       owl:ObjectProperty ;
        rdfs:domain :Person ;
        rdfs:range :Date .

:Person
        a       owl:Class ;
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :managedCompanies
                ] ;
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :gender
                ] ;
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :lastName
                ] ;
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :isUnemployed
                ] ;
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:maxCardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :bank
                ] ;
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :isMarried
                ] ;
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :age
                ] ;
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :firstName
                ] ;
```

```
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :birthDate
                ] .

owl:Thing
        a       owl:Class .

:Company
        a       owl:Class ;
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :numberOfEmployees
                ] ;
        rdfs:subClassOf
                [ a       owl:Restriction ;
                  owl:cardinality "1"^^xsd:nonNegativeInteger ;
                  owl:onProperty :name
                ] .

:bank
        a       owl:ObjectProperty ;
        rdfs:domain :Person ;
        rdfs:range :Bank .

:lastName
        a       owl:DatatypeProperty ;
        rdfs:domain :Person ;
        rdfs:range xsd:string .

:numberOfEmployees
        a       owl:DatatypeProperty ;
        rdfs:domain :Company .

:isUnemployed
        a       owl:DatatypeProperty ;
        rdfs:domain :Person .

:day
        a       owl:DatatypeProperty ;
        rdfs:domain :Date .

:firstName
        a       owl:DatatypeProperty ;
        rdfs:domain :Person ;
        rdfs:range xsd:string .
```