Metalevel Information in Ontology-Based Applications

Thanh Tran Duc, Peter Haase
AIFB Institute,
University of Karlsruhe
D-76128 Karlsruhe, Germany

Boris Motik, Bernardo Cuenca Grau, Ian Horrocks
Computing Laboratory, University of Oxford
Wolfson Building, Parks Road,
Oxford, OX1 3QD, UK

Abstract

Applications of Semantic Web technologies often require the management of *metalevel* information—that is, information that provides additional detail about domain-level information, such as provenance or access rights policies. Existing OWL-based tools provide little or no support for the representation and management of metalevel information. To fill this gap, we propose a framework based on *metaviews*—ontologies that describe facts in the application domain. We have implemented our framework in the KAON2 reasoner, and have successfully applied it in a nontrivial scenario.

Introduction

Applications of the Web Ontology Language (OWL) abound in both academia and industry. By relying on OWL, applications can exploit the extensive body of research in ontology management and reasoning, as well as an advanced tool base supporting complex services such as query answering. OWL is often used for modeling the objects of an application domain. For example, in an application that manages corporate data about products, an OWL ontology might be used to capture the structure of the product catalog. Such a *domain ontology* can be used for data gathering and access.

Domain information is often accompanied by *metalevel* information that does not talk about the application domain, but describes the domain information itself. For example, an application might want to record the provenance and quality of domain information and to use them for filtering and ranking queries to domain information, such as retrieving "all information from a trusted source." Other possible uses of metalevel information include the provision of an access-rights model for domain information, and adding hints about the proper semantic meaning of ontology entities using *metaproperties* (Welty & Andersen 2005).

Several formalisms have been proposed that allow for a clear separation and a controlled interaction between domain and metalevel information. Modal logics have often been used to give a precise model-theoretic interpretation to modalities such as beliefs and agent knowledge (Halpern & Moses 1992). The metareasoning framework (Criscuolo, Giunchiglia, & Serafini 2002) distinguishes between object and metatheories and provides bridging rules for linking the statements at different levels. Context logics (Guha, McCool, & Fikes 2004) allow one to state that a proposition is true in some context. Such logics, however, are often rather complex and not practicable. At the other end of the spectrum, the Resource Description Framework (RDF) (Klyne & Carroll 2004) provides a rudimentary mechanism for the representation of metalevel information via *reification*; however, the semantics of reification has not been precisely defined. (Schueler *et al.* 2008) have extended RDF and SPARQL with constructs that enable explicit representation and querying of metalevel information, but it is unclear how to apply this approach to expressive languages such as OWL. In OWL, metalevel information about entities can be captured using annotations; however, these can be used in an OWL ontology only in a very restricted way. Furthermore, OWL provides no explicit support for metalevel information about axioms. Thus, OWL applications currently need to devise ad hoc mechanisms for solving problems related to metalevel information. Such mechanisms are often proprietary, so there is little or no standardized tool support.

In this paper we present a simple yet semantically sound framework for the management of metalevel information—one that can easily be integrated into existing ontology management systems and reasoners. Our framework is based on the observation that domain and metalevel information have distinct universes of discourse. We store the metalevel statements in the domain ontology using *axiom annotations*—ontology statements that are akin to comments in a programming language and that do not affect the semantics of the domain information. We give semantics to this information by translating the metalevel statements from the domain ontology into a *metaview*—an ontology that explicitly talks about the facts in the domain ontology. The domain ontology and its metaview are interpreted independently. We propose a query language MQL that can be used to integrate the information in the two ontologies in a controlled manner.

We have implemented our framework in the ontology management and reasoning system KAON2 (Motik & Sattler 2006). The implementation is relatively straightforward, which we take as evidence that our framework is lightweight and does not require extensive change to existing tools. To demonstrate the usefulness of our framework in practice, we have applied it in a nontrivial application scenario.

Our framework is based on the OWL 1.1 W3C Member Submission,[1] and we assume the reader to be familiar with its Functional-Style Syntax. OWL 1.1 is currently being evolved into a new W3C standard called OWL 2. Once this work is completed, aligning our framework with OWL 2 should be straightforward.

Application Scenario

Our application scenario is based on a use case from a European research project. In order to keep up with its competitors, one of the project partners—a well-known European car manufacturer—maintains a Competitive Analysis Department, in which analysts produce monthly reports about competitors' new cars and improvements to existing models. Product managers use such reports to determine which cars are likely to adversely affect manufacturer's market position.

The Department employs a range of techniques for gathering the data necessary to produce these reports. For example, a lot of data is manually gathered and entered into various databases. In order to deal with the large volumes of information now available online, the Department also uses text analysis tools that periodically scan online information sources; these tools typically provide quality indicators such as confidence in data accuracy. Example 1 show the type of information that is gathered during the acquisition process.

Example 1 *Mazda2 consumes 5.3 l/100km. (origin: Motorbox; agent: text analysis; confidence: 0.8)*

In order to generate the monthly report, the analysts run various queries over the databases containing the gathered data. They often want to filter out "uncertain" information, and so they might ask the following query.

Example 2 *Retrieve all pieces of information about Mazda car models together with their associated confidence levels.*

Furthermore, since fuel prices are high, fuel consumption is a critical factor in the report. Hence, the analysts might want to verify the information sources that provide information about models with low fuel consumption.

Example 3 *Retrieve all information sources stating some information about car models with low consumption.*

Finally, the analysts might want to retrieve information about competitors' low-consumption cars; since this represents critical marketing information, only high quality information sources should be considered.

Example 4 *Retrieve all car models with low consumption, but consider only high-quality information sources.*

Large amounts of data are needed in order to produce accurate reports, so a high degree of automation is crucial to the Department's success. This is to be achieved using a common conceptual model that can be used for end-to-end description of the data in the system. Such a conceptual model should be multifaceted. It should provide a common vocabulary that can capture common knowledge about the modeled domain, as in the following example.

[1] `http://www.w3.org/Submission/2006/10/`

Example 5 *Each car model has a nominal consumption measured in l/100km. Car models whose consumption is lower than 6 l/100km have low consumption.*

The conceptual model should also represent analysts' background knowledge about the information gathering and report generation processes, as in the following example.

Example 6 *Motorbox is a low-quality, and Newstreet is a high-quality information source. Information that is manually entered is of high-quality.*

Finally, the conceptual model should allow the application to store the actual information gathered in the process, such as that presented in Example 1, and should support queries such as the ones in Examples 2–4. Ontologies are nowadays commonly used in applications for conceptual modeling, so it is natural to apply them in this scenario.

The Metaview Approach

We now present a framework that can be used to realize scenarios such as the one described in the previous section.

Domain- and Metalevel Information

Using ontology-based technologies in our scenario is not straightforward. From Examples 1, 5, and 6, we can see that our application needs to represent two different types of information. The facts that "the consumption of Mazda2 is 5.3 l/100km" and that "low consumption cars consume less than 6 l/100km" describe the application domain; we call such information *domain-level*. The fact that the consumption of Mazda2 has been discovered on Motorbox describes not the application domain, but the domain-level statements themselves; we call such information *metalevel*.

The distinction between domain- and metalevel information is crucial in our scenario. One might argue that metalevel information can be represented by simply extending the domain-level ontology appropriately. For example, the information from Example 1 can be represented using a ternary relation connecting the car model, the consumption, and the information source. Such a modeling style, however, is quite cumbersome because it interacts adversely with reasoning; for example, it is unclear how the information about the origin of the fact should influence the conclusion that Mazda2 is not a low consumption car. Moreover, RDF and OWL support only binary relations. Hence, separating the two types of information is useful for both conceptual and practical reasons. To do so, we need an approach for representing and relating domain- and metalevel information.

Metalevel Information in RDF

Before presenting our approach, we overview the capabilities for representing metalevel information in RDF—the simplest of the Semantic Web languages. RDF graphs consist of triples of the form $\langle s\ p\ o \rangle$, where s is the *subject*, p is the *predicate*, and o is the *object*; subjects, predicates and objects are all called *resources*. For example, the domain-level statement from Example 1 can be represented by triple (1). Metalevel information about RDF resources can be encoded using standard triples. For metalevel information

about RDF statements, RDF provides a mechanism called *reification*. To represent information about triple (1), the triple can be *reified*—that is, it can be given identity and turned itself into triples. Triple (1) can thus be represented by a new resource *a:t1* and described by triples (2). Information about (1) can then be stated by attaching it to *a:t1*, as it is shown in triple (3).

(1) ⟨*a:Mazda2 a:cons "5.3"*⟩

(2) ⟨*a:t1 rdf:type rdf:Statement*⟩, ⟨*a:t1 rdf:subject a:Mazda2*⟩, ⟨*a:t1 rdf:predicate a:cons*⟩, ⟨*a:t1 rdf:object "5.3"*⟩

(3) ⟨*a:t1 a:origin a:Motorbox*⟩

The main drawback of RDF reification is that the relationship between the original and the reified triples is unclear. For example, it is unclear whether (1) implies (2) or vice versa. It is also unclear whether the infrastructure that realizes the management of RDF ontologies should generate (2) from (1) automatically, or whether the reified triples should be entered manually. Hence, reification has not been used extensively in practice and is not supported in OWL DL.

Framework Overview

Our framework for the management of domain- and metalevel information is schematically shown in Figure 1. It is inspired by RDF reification, but without the ambiguities in the semantics. It is based on an observation that the domain- and metalevel information talk about different universes of discourse: the domain information may, for example, talk about vehicles and vehicle information sources, while the metalevel information talks about domain statements themselves. We therefore use distinct ontologies to capture domain- and metalevel information; we call the former the *domain ontology* and the latter the *metaview*.

Keeping the domain and metalevel information in physically separate ontologies would make information maintenance difficult. Thus, we use annotations to add metalevel information to the domain ontology O in a way which allows us to extract the metaview from O whenever needed. By using annotations, we ensure that metalevel information in O does not affect the formal meaning of O. We define an operator μ that transforms the domain ontology O into its metaview $\mu(O)$. Thus, the metaview can be though of as a "virtual" ontology that is generated from O on demand. The metaview is, however, an OWL ontology, thus providing precise semantics to the metalevel information.

The ontology $\mu(O)$ contains the reified facts of O, represented as instances of O_{mo}—a metaontology that models the structure of OWL 1.1 axioms; furthermore, $\mu(O)$ contains assertions about these facts that are extracted from their annotations in O. Thus, $\mu(O)$ might contain a description of the fact in Example 1: it would represent the structure of the fact "Mazda2 consumes 5.3 l/100km", and the statement that this fact was derived from Motorbox using text analysis with a confidence of 0.8.

Knowledge about the general structure of the metalevel information used in the application can be described in a separate application-specific ontology O_{ml}. This ontology might contain statements such as those in Example 6—that is, that Newstreet and manual entry are both high-quality

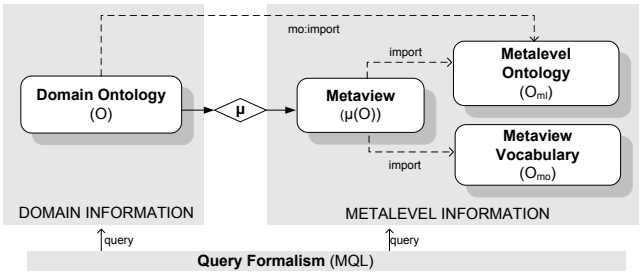

Figure 1: The Metaview Framework

information sources, while Motorbox is a low-quality information source.

The operator μ ensures that each metaview ontology $\mu(O)$ imports O_{mo} via the standard OWL 1.1 import mechanism. To make O_{ml} available to $\mu(O)$, our framework provides a special annotation *mo:imports* that can be added to O and that is mapped by μ to an imports statement in $\mu(O)$. Thus, by adding Annotation(*mo:import* O_{ml}) to O, we ensure that $\mu(O)$ imports O_{ml} and thus gains access to the relevant metalevel domain descriptions. Note that although O_{ml} is application specific, it describes the structure of the metalevel domain, and can thus be expected to change much less frequently than the "data" in $\mu(O)$.

Ontologies O and $\mu(O)$ do not import each other, so there is no semantic interaction between them. Typical applications will, however, often need to integrate information from a domain ontology and its metaview; for example, they might want to answer questions such as the one presented in Example 4. In order to achieve this we propose MQL—a language that allows for querying both domain- and metalevel information and integrating the query answers.

Representing Metalevel Information in OWL 1.1

The domain-level information can be represented straightforwardly using an OWL 1.1 domain ontology O. For example, the statements from Example 5 correspond naturally to the following axioms.

(4) SubClassOf(*a:CarModel* DataSomeValuesFrom(*a:cons xsd:float*))

(5) SubClassOf(DataSomeValuesFrom(*a:cons* DatatypeRestriction(*xsd:float* maxExclusive "6")) *a:LowConsCarModel*)

Similarly, the description of the metalevel domain can be represented as an OWL 1.1 ontology O_{ml}. Hence, Example 6 can be represented by axioms (6)–(8).

(6) ClassAssertion(*a:Motorbox a:LowQualityIS*)

(7) ClassAssertion(*a:Newstreet a:HighQualityIS*)

(8) SubClassOf(ObjectHasValue(*a:agent a:Manual*) ObjectSomeValuesFrom(*a:origin a:HighQualityIS*))

The metalevel information about the domain-level facts is stored in the domain ontology O using OWL 1.1 *annotations*. The existing OWL standard already allows for annotations on ontology entities: concepts, properties, and individuals can have information attached to them that is akin to

comments in programming languages, and OWL 1.1 extends this idea to axioms and even ontologies as well. Annotations in OWL 1.1 have the form (9) or (10), depending on whether the value of the annotation is an individual or a constant.

(9) Annotation(*annotationURI individual*)

(10) Annotation(*annotationURI constant*)

Thus, to represent statements from Example 1, we add to O the following axiom.

(11)
DataPropertyAssertion(
Annotation(*a:origin a:Motorbox*)
Annotation(*a:agent a:TextAnalysis*)
Annotation(*a:conf* "0.8")
a:cons a:Mazda "5.3")

In this way, domain- and metalevel information is managed physically in one place, but metalevel information has no semantic effect on domain-level information.

Semantics of Metalevel Information

The metaview ontology $\mu(O)$ consists of (1) a reference to the description of the metalevel domain, (2) a reified representation of the axioms of O, (3) the logical interpretation of the metalevel information stored in O. Due to space constraints, we defer the definition of μ to an online appendix.[2]

For part (1) of $\mu(O)$, whenever O contains an ontology annotation of the form Annotation(*mo:import* O_{ml}), the ontology $\mu(O)$ imports O_{ml} using the standard OWL 1.1 import mechanism.

For part (2) of $\mu(O)$, we have created a metaontology O_{mo} that captures the structure of OWL 1.1 ontologies,[3] and have defined μ such that each metaontology imports O_{mo}. This ontology is independent of the domain, and its purpose is to describe OWL 1.1 axioms. For example, a data property assertion is described in O_{mo} using the following axiom (*mo:* is the namespace prefix used in O_{mo}), which states that a data property assertion is a kind of fact and that it has exactly one property, one source, and one target that are a data property, an individual, and a constant, respectively.

SubClassOf(*mo:DataPropertyAssertion*
ObjectIntersectionOf(*mo:Fact*
ObjectExactCardinality(1 *mo:property mo:DataProperty*)
ObjectExactCardinality(1 *mo:source mo:Individual*)
DataExactCardinality(1 *mo:target rdfs:Literal*)))

Each axiom α of O is assigned by μ a unique *representative individual* μ_α, and α is then translated into assertions by following the structure defined by O_{mo}. For example, assertion (11) is assigned the individual $\mu_{(11)}$, and is then translated into assertions (12)–(14).

(12) ObjectPropertyAssertion($\mu_{(11)}$ *mo:property a:cons*)

(13) ObjectPropertyAssertion($\mu_{(11)}$ *mo:source a:Mazda*)

(14) DataPropertyAssertion($\mu_{(11)}$ *mo:target* "5.3")

Part (3) of $\mu(O)$ is generated by translating annotations into assertions: if an axiom α contains an annotation of the form (9) or (10), the annotation is translated into an assertion (15) or (16), respectively.

[2] `http://owlodm.ontoware.org/mu.pdf`
[3] `http://owlodm.ontoware.org/OWL1.1`

(15) ObjectPropertyAssertion(μ_α *annotationURI individual*)

(16) DataPropertyAssertion(μ_α *annotationURI constant*)

For example, the annotations of (11) are translated into assertions (17)–(19). Thus, the metalevel information in O is interpreted in $\mu(O)$ as domain level information.

(17) ObjectPropertyAssertion($\mu_{(11)}$ *a:origin a:Motorbox*)

(18) ObjectPropertyAssertion($\mu_{(11)}$ *a:agent a:TextAnalysis*)

(19) DataPropertyAssertion($\mu_{(11)}$ *a:conf* "0.8")

Querying Metalevel Information

We now present a query language called MQL, which allows for querying an ontology and its metaview in an integrated way. The main goal in MQL is to enable the integration of the domain ontology with its metaview, so we assume that the queries over a single ontology (either the domain ontology or the metaview) can be answered by some query language $\mathcal{L}_Q$. In practice, $\mathcal{L}_Q$ might be, e.g., the language of conjunctive queries over OWL ontologies (Calvanese, Giacomo, & Lenzerini 1998), which we can process using well-known algorithms (Glimm *et al.* 2007). With $Q(\vec{x})$ we denote a query of $\mathcal{L}_Q$, where $\vec{x}$ is the vector of *distinguished variables*—that is, the variables that should be substituted with individuals to obtain an answer. We write $O \models Q(\vec{a})$ to denote that $\vec{a}$ is an answer to $Q(\vec{x})$ in O.

Definition 1 *Let $\mathcal{L}_Q$ be a language for querying a single OWL ontology. The syntax of MQL queries is defined as follows, with the restriction that each* ***mql-query*** *must be domain independent (Abiteboul, Hull, & Vianu 1995).*

$\mathcal{L}_Q$***-query*** $\leftarrow$ *a query of* $\mathcal{L}_Q$ *of the form* $Q(\vec{x})$
O-spec $\leftarrow$ *ontology specification of the form* O
mql-atom $\leftarrow$ $\mathcal{L}_Q$***-query***@[***mql-query*** | ***O-spec***]
mql-query $\leftarrow$ ***mql-atom*** | $x_1 = x_2$ | $\neg$ ***mql-query*** |
$\exists x$: ***mql-query*** | ***mql-query*** $\wedge$ ***mql-query***

We define the answer $\mathsf{ans}(\varphi)$ *of an MQL query φ using the following induction.*

- *If* $\varphi = Q(\vec{x})@[O]$, *then* $\vec{a} \in \mathsf{ans}(\varphi)$ *iff* $O \models Q(\vec{a})$.
- *If* $\varphi = Q(\vec{x})@[\psi]$, *then* $\vec{a} \in \mathsf{ans}(\varphi)$ *iff* $\omega \models Q(\vec{a})$ *for* $\omega = \{\alpha \mid \mu_\alpha \in \mathsf{ans}(\psi)\}$.
- $\mathsf{ans}(\varphi)$ *is defined as usual (Abiteboul, Hull, & Vianu 1995) if the top-level connective of* φ *is* $=$, $\neg$, $\exists$, *or* $\wedge$.

The main difference of MQL to standard relational query languages is in the definition of atoms. Atoms of the form $\varphi = Q@[O]$ are evaluated by simply answering the $\mathcal{L}_Q$-query Q over the ontology O. Atoms of the form $\varphi = Q@[\psi]$ where ψ is a nested MQL query are evaluated as follows. First, ψ is evaluated recursively. Next, for each individual $x \in \mathsf{ans}(\psi)$, if x was assigned by μ to some axiom α (i.e., if $x = \mu_\alpha$), then α is added to the set ω; otherwise, x is ignored. After this process, ω contains the axioms selected by the nested query ψ. Finally, the $\mathcal{L}_Q$-query Q is answered in the axiom set ω. Since MQL queries are domain independent, they can be evaluated recursively similarly to standard first-order relational queries (Abiteboul, Hull, & Vianu 1995), by just changing the way in which atoms are processed. Based on this observation, we were able to implement the metaview framework and MQL on top of the KAON2 reasoner with only moderate effort.

Table 1: Typical MQL Query Patterns

Query Type	Pattern
Domain	$\varphi(x) = Q_D(x)@[O]$
Metalevel	$\varphi(x) = Q_M(x)@[\mu(O)]$
Filter	$\varphi(x) = Q_D(x)@[Q_M(y)@[\,\mu(O)]\,]$
Join	$\varphi(x,y) = Q_M(x,y)@[\mu(O)] \wedge Q_D(x)@[O]$

MQL is a very expressive language; however, we expect that most queries used in practice will follow the patterns shown in Table 1.[4]

The *domain* and the *metalevel* query patterns retrieve either only domain or metalevel information without any interaction between the two. Query (20) is an example of the former query pattern, and it retrieves all instances of *a:MazdaModel* in O. Query (21) is an example of the latter query pattern, and it retrieves all facts in the metaview together with their confidence values.

(20) $\varphi_1(x) = \textit{a:MazdaModel}(x)@[O]$

(21) $\varphi_2(x,y) = (\ \textit{mo:Fact}(x) \wedge \textit{a:conf}(x,y)\ )@[\mu(O)]$

The *filter* query pattern consists of a domain query Q_D that is answered over a subset of the domain ontology; this subset is defined by a nested query Q_M over the metaview. The query from Example 4 is such a query and it can be expressed in MQL as follows. The $\mathcal{L}_Q$-query (22) selects all facts from high-quality information sources, and the $\mathcal{L}_Q$-query (23) selects all cars with low consumption. The MQL query (24) thus selects all axioms that are of high quality, and (25) selects all car models with low consumption from that set. To evaluate φ_3, we recursively evaluate ψ and then evaluate Q_2 over the set of axioms that correspond to the representative individuals contained in the result of ψ.

(22) $Q_1(x) = \exists y : \textit{mo:Axiom}(x) \wedge \textit{a:origin}(x,y) \wedge \textit{a:HighQualityIS}(y)$

(23) $Q_2(z) = \textit{a:LowConsCarModel}(z)$

(24) $\psi(w) = Q_1(w)@[\mu(O)]$

(25) $\varphi_3(z) = Q_2(z)@[\psi]$

The *join* query pattern consists of a metalevel query Q_M whose results are filtered by a join with a domain-level query Q_D. The query from Example 2 is such a query and it can be expressed in MQL as follows. The $\mathcal{L}_Q$-query (26) selects the facts, the entities that the fact is about (encoded using the *mo:source* property), and their confidence values. The $\mathcal{L}_Q$-query (27) selects all Mazda car models. The MQL query (28) thus evaluates $Q_3(x,y,z)$ over $\mu(O)$, evaluates $Q_4(y)$ over O, joins the results on y, and finally projects y out.

(26) $Q_3(x,y,z) = \textit{mo:Fact}(x) \wedge \textit{mo:source}(x,y) \wedge \textit{a:conf}(x,z)$

(27) $Q_4(y) = \textit{a:MazdaModel}(y)$

(28) $\varphi_4(x,z) = \exists y : (\ \ Q_3(x,y,z)@[\mu(O)] \wedge Q_4(y)@[O]\ )$

MQL is closely related to EQL-Lite($\mathcal{Q}$) (Calvanese *et al.* 2007). EQL-Lite($\mathcal{Q}$) is parameterized with a first-order query language $\mathcal{Q}$. Queries of EQL-Lite($\mathcal{Q}$) are first-order formulas over atoms of the form $\mathbf{K}\,\psi$ for ψ a $\mathcal{Q}$-query; the semantics of $\mathbf{K}\,\psi$ is essentially equivalent to the MQL-atom $\psi@[O]$. MQL can thus be seen as a generalization of EQL-Lite($\mathcal{Q}$), in which MQL-atoms can query more than one ontology and/or "dynamic" axiom sets generated via nested queries to the metaview.

An Application Based on Metaviews

Based on the KAON2's implementation of the metaview framework, we have implemented a *knowledge browser*[5] that supports users in the previously mentioned scenario. Roughly speaking, the browser first asks the user to specify the *task* he is trying to accomplish. Then, the browser implements *data filtering*: it presents the user with a view over the data that matches with the specified task.

The data filtering process is described by a *filtering policy*—a declarative description of which data is relevant for which task. The policy is modeled as an ontology O_{po},[6] which corresponds to the metalevel ontology O_{ml} in Figure 1. The ontology extends the well-known top-level Suggested Upper Merged Ontology (SUMO),[7] and its axioms can be separated into two main parts.

Part (*i*) of O_{po} describes various information sources, information extraction methods, and the quality of information that they produce. Information sources are represented by the *po:InformationProvider* concept, human information extractors are represented by the *po:Employee* concept, and software information extractors are represented by the *po:SoftwareProgram* concept; all these are subconcepts of *sumo:Agent.* Information sources and information extractors are associated with the facts in the domain ontology using the *po:origin* and *po:extractor* properties, respectively. These concepts and properties are used to define more complex concepts. For example, the concept *po:TrustedAgent* is defined to contain all agents that are considered trustworthy according to the company's policy. Similarly, the concept *po:HighQualityFact* is defined to contain "all facts that have been extracted by a trusted agent, that come from a trusted information source, and for which the agent specified a confidence value higher than 0.8."

Part (*ii*) of O_{po} describes the task that the user currently working with the browser is trying to accomplish. Unlike Part (*i*) which is general and does not change frequently, Part (*ii*) of O_{po} changes whenever the user completes one task and moves to the next one: the old task description in O_{po} is then replaced with a new one. The task is described by means of the identity of the user that is working with the application, the type of task that the user is trying to accomplish, and the lists of information sources and information extractors that are relevant for the task. Tasks are modeled as instances of the *sumo:Process* concept, and users are associated with the tasks using the *po:actor* property. Finally, information sources and information extractors are associated with the task using the *sumo:instrument* property, which is used in SUMO to denote that a process (i.e.,

[4] In the absence of a standardized syntax for conjunctive queries over OWL ontologies, we use the common notation in which concepts and properties are unary and binary predicates, respectively.

[5] `http://ontoware.org/projects/xxplore/`

[6] `http://xxplore.ontoware.org/policy`

[7] `http://www.ontologyportal.org/`

a task) is accomplished by means of some other entity. Axioms (29)–(32) show a description of a calendar forecasting task: *po:Marina* is the user working on the task, which is being accomplished using the *po:Motorbox* information source and the *po:Pronto* software extractor—a tool for extracting ontology relations from text. Note that the latter three individuals are described in O_{po}.

(29) ClassAssertion(*po:CalenderForecasting po:Process*)

(30) ObjectPropertyAssertion(*po:actor po:CalenderForecasting po:Marina*)

(31) ObjectPropertyAssertion(*po:instrument po:CalenderForecasting po:Motorbox*)

(32) ObjectPropertyAssertion(*po:instrument po:CalenderForecasting po:Pronto*)

To construct the view containing the facts relevant to the current task, the knowledge browser uses then the MQL query $q = Q_5(x)@[\mu(O)]$, where Q_5 is defined as follows.

$$\begin{aligned} Q_5(x) = {} & \textit{po:HighQualityAxiom}(x) \wedge \textit{po:Process}(y) \wedge \\ & \textit{po:origin}(x, v) \wedge \textit{po:TrustedAgent}(v) \wedge \\ & \textit{po:extractor}(x, w) \wedge \textit{po:TrustedAgent}(w) \wedge \\ & \textit{po:instrument}(y, v) \wedge \textit{po:instrument}(y, w) \end{aligned} \tag{33}$$

Intuitively speaking, Q_5 selects all facts that were produced by the information source and extracted by an agent both of which are relevant for the task and are trusted according to company's policy. The query q then evaluates Q_5 over the metaview $\mu(O)$ in order to perform information filtering. The result of q can be used either directly for information browsing, or it can be used to answer domain-level queries.

Validation of our Framework

To validate the practical usefulness of our framework, we analyzed the actual information needs of our project partner. In particular, we interviewed eight company's employees, all of whom were familiar with traditional database user interfaces and the concept of querying. The interviewees were first made acquainted with the domain and the metalevel ontologies, and then they were asked to provide as many (*i*) domain-level concepts called *types*, such as *a:Model*, *a:News*, and *a:Forecast*, that determine the objects being queried, and (*ii*) concepts and properties that can restrict the types called *selectors*. For example, the domain-level selector *car:Maker* can be applied to the type *a:Model* to retrieve all models by a certain manufacturer; similarly, the metalevel selector *po:extractor* can be applied to the type *a:News* to retrieve all news obtained through a certain extractor. In total, 3 types and 36 selectors were identified. We analyzed 100 "sensible" combinations of types and selectors, for which we extracted an intuitive description of what the combination should mean w.r.t. the domain and the policy ontologies. Then, we translated each description into a *query schema*—an expression containing placeholders for the selectors' values—that realizes the intuitive semantics of the combination. We thus obtained 59 domain-level queries, 33 filter queries, and 8 join queries. We did not encounter a combination of types and selectors whose intuitive meaning cannot be transformed into MQL. Thus, our framework seems to cover the information needs of our use case.

We evaluated the performance of our implementation. Our main goal was just to see whether our framework is implementable at all, and not to conduct an exhaustive performance evaluation. Using a standard PC, we measured the time needed to evaluate 24 query schemas in which placeholders were replaced with values supplied by the users. The domain ontology contained 110 axioms, and the metaview ontology inherited 227 axioms from O_{mo}. We used four data sets containing 50K, 100K, 500K, and 1M assertions, respectively. The longest times to evaluate a query were 0.5s, 1.2s, 2.4s, and 8.8s, respectively, while the average times were 0.2s, 4s, 1.5s, and 3.7s. These results suggest that our implementation can handle nontrivial data sets.

Conclusion

We presented a framework for the representation and integration of domain- and metalevel information in the Semantic Web. Even without relying on a complex logical formalism, our framework is capable of satisfying the information needs of a nontrivial practical scenario. At the same time, our framework provides a well-understood and precise semantics to the metalevel information. This semantics is based on the standard semantics of OWL, thus allowing the framework to be implemented using existing technologies.

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison Wesley.

Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. EQL-Lite: Effective First-Order Query Processing in Description Logics. In *Proc. IJACI 2007*, 274–279.

Calvanese, D.; Giacomo, G. D.; and Lenzerini, M. 1998. On the Decidability of Query Containment under Constraints. In *Proc. PODS '98*, 149–158.

Criscuolo, G.; Giunchiglia, F.; and Serafini, L. 2002. A Foundation for Metareasoning Part I: The Proof Theory. *J. Log. Comput.* 12(1):167–208.

Glimm, B.; Horrocks, I.; Lutz, C.; and Sattler, U. 2007. Conjunctive Query Answering for the Description Logic $\mathcal{SHIQ}$. In *Proc. IJCAI 2007*, 399–405.

Guha, R. V.; McCool, R.; and Fikes, R. 2004. Contexts for the Semantic Web. In *Proc. ISWC 2004*, 32–46.

Halpern, J. Y., and Moses, Y. 1992. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence* 54(3):319–379.

Klyne, G., and Carroll, J. J. 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax. `http://www.w3.org/TR/rdf-concepts/`.

Motik, B., and Sattler, U. 2006. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In *Proc. LPAR 2006*, 227–241.

Schueler, B.; Sizov, S.; Staab, S.; and Tran, T. 2008. Querying for Meta Knowledge. In *Proc. WWW 2008*.

Welty, C., and Andersen, W. 2005. Towards ontoclean 2.0: A framework for rigidity. *Applied Ontology* 1(1):107–116.