Ontology Representation & Querying for Realizing Semantics-driven Applications

Boris Motik, Alexander Maedche, and Raphael Volz

FZI Research Center for Information Technologies at the University of Karlsruhe,
D-76131 Karlsruhe, Germany
{motik,maedche,volz}@fzi.de

Abstract. In recent years ontologies – shared conceptualizations of some domain – are increasingly seen as the key to further advances in automation of information processing. Although many approaches for representing and querying ontologies have already been devised, they haven't found their way into enterprise applications yet. Many of them offer a high degree of expressivity, but require complicated reasoning and query answering procedures, and often lack features needed in practical applications, such as constraints and meta-concept modeling. This is compounded by the lack of critical technical features in ontology management tools, such as scalability, reliability, concurrency and the support for ontology modularization. We present an approach for ontology representation and querying that balances some of the trade-offs to more easily integrate into existing enterprise information infrastructure. In particular, we focus on efficient evaluation of queries using existing information management infrastructure, such as relational databases. Finally, we describe a prototype implementation within KAON, the Karlsruhe Ontology and Semantic Web tool suite.

1 Introduction

The application of ontologies is increasingly seen as key to enabling semantics-driven information access. There are many applications of such an approach, e.g. automated information processing, information integration or knowledge management, to name just a few. Especially after Tim Berners-Lee coined the vision of the Semantic Web [6], where Web pages are annotated by ontology-based meta-data, the interest in ontology research increased, in hope of finding ways to off-load processing of large volumes of information from human users to autonomous agents.

Many ontology languages have been developed, each aiming to solve particular aspects of ontology modeling. Some of them, such as RDF(S) [30, 9], are simple languages offering elementary support for ontology modeling for the Semantic Web. There are other, more complex languages with roots in formal logic, with particular focus on advanced inference capabilities – mechanisms to automatically infer facts not explicitly present in the model. For example, the F-logic language [28] offers ontology modeling through an object-oriented extension of Horn logic. On the other hand, various classes of description logic languages (e.g. OIL [18] or OWL [38]) are mainly concerned with finding an appropriate subset of first-order logic with decidable and complete subsumption inference procedures.

Despite a large body of research in improving ontology management and reasoning, features standardized and widely adopted in the database community (such as scalability or transactions) must be adapted and re-implemented for logic-based systems. Support for ontology modularization is typically lacking in most systems. Logic-based approaches often focus primarily on the expressivity of the model, often neglecting the impact that this expressivity has on performance and ease of integration with other systems. Because of these problems, up until today there hasn't been a large number of successful enterprise applications of ontology technologies.

On the other hand, relational databases have been developed over the past 20 years to a maturity level unparalleled with that of ontology management systems, incorporating features critical for business applications, such as scalability, reliability and concurrency support. However, database technologies alone are not appropriate for handling semantic information. This is mainly due to the fact that the conceptual model of a domain (typically created using entity-relationship modeling [14]) for the actual implementation must be transformed into a logical model. After the transformation, the structure and the intent of the original model are not obvious. Therefore, the conceptual and the logical model tend to diverge. Further, operations that are natural within the conceptual model, such as navigation between objects or ontology querying, are not straightforward within the logical model.

Contribution of the paper. In this paper we present an approach for ontology representation and querying that tries to adapt the expressivity of the model with other requirements necessary for successful realization of business-wide applications. We adjust the expressiveness of traditional logic-based languages to sustain tractability, thus making realization of enterprise-wide ontology-based systems possible using existing well-established technologies, such as relational databases. Other critical features are modularization and modeling meta-concepts with well-defined semantics. Based on our ontology structure, we give an approach for conceptual querying of ontologies. We present the current status of the implementation of our approach within KAON [35] – Ontology and Semantic Web tool suite used as basis for our research and development.

Paper Structure. The rest of this paper is structured as follows. In section 2 we present the requirements based on which we derive our ontology representation and querying approach. Section 3 presents our ontology language in more detail, by giving a mathematical definition, denotational semantics and some examples. In section 4 we present our approach for answering queries over ontologies. In section 5 we present the current implementation status. In section 6 we contrast our work with related approaches. Finally, in section 7 we conclude the paper and give our directions for future research.

2 Requirements

In this section we discuss the requirements based on our observations we gathered while working on several research and industry projects. For example, Ontologging is focused on applying ontologies to knowledge management in hope of improving searching and navigation within a large document base. Harmonise tries to apply ontologies to provide

interoperability among enterprise systems in B2B scenarios, with tourism as the application field. Finally, Semantic Web enabled Web Services (SWWS) explores applying ontologies to improve and automate Web service-based interaction between businesses on the Internet.

Object-oriented Modeling Paradigm. In the last decade the object-oriented paradigm has become prevalent for conceptual modeling. A wide adoption of UML [20, 16] as syntax for object-oriented models has further increased its acceptance.

The object-oriented modeling paradigm owes its success largely to the fact that it is highly intuitive. Its constructs match well with the way people think about the domain they are modeling. Object-oriented models can easily be visualized, thus making understanding conceptual models much simpler. Hence, any successful ontology modeling approach should follow the object-oriented modeling paradigm.

Meta-concepts. When typing to create complex models of the real world, often it may be unclear whether some element should be represented as a concept or as an instance. An excellent example in [39] demonstrates problems with meta-concept modeling. While developing a semantics-driven image retrieval system, it was necessary to model the relationship between notions of species, ape types and particular ape individuals. Most natural conceptualization is to introduce the SPECIES concept (representing the set of all species), with instances such as APE. However, APE may be viewed as a set of all apes. It may be argued that APE may be modeled as a subconcept of SPECIES. However, if this is done, other irregularities arise. Since APE is a set of all apes, SPECIES, being a superconcept of APE, must contain all apes as their members, which is clearly wrong. Further, when talking about the APE species, there are many properties that may be attached to it, such as habitat, type of food etc. This is impossible to do if APE is a subconcept of SPECIES, since concepts cannot have instantiated property values.

There are other examples where meta-concept modeling is necessary:

- Ontology mapping is a process of mapping ontology entities in different ontologies in order to achieve interoperability between information in both ontologies. As described in [31], it is useful to represent ontology mapping as a meta-ontology that relates concepts of the ontologies being mapped.
- It is beneficial to represent ontology changes by instantiating a special evolution log ontology [32]. This (meta-)ontology consists of concepts reflecting various types of changes in an ontology.
- As described in [21], to guide ontology modeling decisions, it is beneficial to annotate ontology entities with meta-information reflecting fundamental properties of ontology constructs. To attach this meta-information to classes in an ontology, classes must be viewed as instances of meta-concepts that define these properties.

As already noted by some researchers [40, 44], support for modeling meta-concepts is important for adequate expression of complex knowledge models. There are logic languages, (e.g. HiLog [12]) with second-order syntax but with first-order semantics for which it has been shown that a sound and complete proof theory exists. Many running systems, such as Protege-2000 [36] incorporate this functionality into their knowledge model [36].

Modularization. It is a common engineering practice to extract a well-encapsulated body of information in a separate module, that can later be reused in different contexts. However, modularization of ontologies has some special requirements: both instances and schematic definitions may be subjected to modularization.

For example, a concept CONTINENT will have exactly seven instances. In order to include information about continents in some ontology, it is not sufficient to reuse only the CONTINENT concept – to be able to talk about particular continents, such as EUROPE, one must reuse the instances of CONTINENT as well. We consider modularization – on both ontologies and instances – to be an important aspect of reuse. Supporting modularization introduces some requirements on the ontology modeling language, but also on the systems implementing the language as well.

Lexical Information. Many applications extensively depend on lexical information about entities in an ontology, such as labels in different languages. Hence, consistent way of associating lexical information with ontology entities is mandatory. Lexical information can be thought of as meta-information about an ontology, since it talks about the ontology elements, so it makes sense to represent this information as just another type of meta-data. This has the benefit that the ontology and lexical information can be manipulated in the same way. For example, querying the ontology by the lexical information is done in the same way as the usual ontology queries.

Adaptable Inference Capabilities. Inference mechanisms for deduction of information not explicitly asserted is an important characteristic of ontology-based systems. However, systems with very general inference capabilities often do not take into account other needs, such as scalability and concurrency.

For example, in RDFS and OWL ontology languages it is possible to make some classes the domain or the range of some property. This statement can be interpreted as an axiom saying that for any property instance in the ontology, the source and target instances can be inferred to be members of the domain and target concepts, respectively. From our experience, this is not how users, especially those with strong background in database and object-oriented systems, typically think of domains and ranges. In OO and database systems, domains and ranges simply specify schema constraints that must be satisfied for the property to be instantiated in the first place. They don't infer new things, but guide the user in constructing the ontology by determining what can be explicitly said at all and provide guidance in what to do when the ontology changes.

Using domains and ranges as inference rules impacts the performance significantly – every property typically has at least one domain and range concept, which means that for every property there are two new inference rules that must be taken into account. If treating domains an ranges as schema constraints is enough, but only the inference rule semantics is provided, then the performance and the scalability of the system will suffer. Hence, for the successful application of ontologies it is necessary to be able to precisely control the inference capabilities according to the requirements at hand.

Conceptual Querying. Adequate query facilities are of critical importance for any ontology system. It is important that results of such queries reflect the original semantics of the model. For example, a typical query to a system managing knowledge about a

set of documents is to retrieve information about some documents and associated authors. If the query returns a relational-style answer (a table with all document-author pairs in each line), then the semantics of information is destroyed by the query: the model contains an n:m relationship, which is in the query result represented as a table with repeated rows. Hence, we stress the requirement for a query facility which doesn't destroy the semantics of the model.

Technical Issues. In order to be applicable for real-world enterprise applications, our ontology representation approach must make it easy to fulfill following technical requirements:

- scalability – systems must be able to cope with large quantities of information,
- concurrency support – it must be possible for several users to use and change information at the same time,
- reliability – the system must under no circumstances loose or corrupt information,
- easy integration with existing data sources.

These requirements aren't trivial to fulfill, largely due to the fact that ontology management infrastructure hasn't reached the maturity of the relational databases. For example, many of existing tools are still file-oriented. This limits the size of ontologies that can be processed, as the whole ontology must be read into main memory. Further, the multi-user support and transactions are typically not present, so the whole infrastructure realizing these requirements must be created from scratch.

3 Ontology Representation

In this section we present the mathematical definition of our modeling language, followed by the presentation of denotational semantics in standard Tarski style [19]. Finally, we present an ontology example.

3.1 Mathematical Definition

We present our approach on an abstract, mathematical level that defines the structure of our models, which can be supported using several different syntaxes.

Definition 1 (OI-model Structure). *An OI-model (ontology-instance-model) structure is a tuple* $OIM := (E, INC)$ *where:*

- E *is the set of entities of the OI-model,*
- INC *is the set of included OI-models.*

An OI-model represents a self-contained unit of structured information that may be reused. An OI-model consists of entities and may include a set of other OI-models (represented through the set INC). Definition 5 lists the conditions that must be fulfilled when an OI-model includes another model.

Definition 2 (Ontology Structure). *An ontology structure of an OI-model is a structure* $O(OIM) := (C, P, R, S, T, INV, H_C, H_P, domain, range, mincard, maxcard)$*:*

- $C \subseteq E$ *is a set of concepts,*
- $P \subseteq E$ *is a set of properties,*
- $R \subseteq P$ *is a set of relational properties (properties from the set* $A = P \setminus R$ *are called attribute properties),*
- $S \subseteq R$ *is a subset of symmetric properties,*
- $T \subseteq R$ *is a subset of transitive properties,*
- $INV \subseteq R \times R$ *is a symmetric relation that relates inverse relational properties; if* $(p_1, p_2) \in INV$*, then* p_1 *is an inverse relational property of* p_2*,*
- $H_C \subseteq C \times C$ *is an acyclic relation called concept hierarchy; if* $(c_1, c_2) \in H_C$ *then* c_1 *is a subconcept of* c_2 *and* c_2 *is a superconcept of* c_1*,*
- $H_P \subseteq P \times P$ *is an acyclic relation called property hierarchy; if* $(p_1, p_2) \in H_P$ *then* p_1 *is a subproperty of* p_2 *and* p_2 *is a superproperty of* p_1*,*
- *function domain* $: P \rightarrow 2^C$ *gives the set of domain concepts for some property* $p \in P$*,*
- *function range* $: R \rightarrow 2^C$ *gives the set of range concepts for some relational property* $p \in R$*,*
- *function* $mincard : C \times P \rightarrow N_0$ *gives the minimum cardinality for each concept-property pair,*
- *function* $maxcard : C \times P \rightarrow (N_0 \cup \{\infty\})$ *gives the maximum cardinality for each concept-property pair.*

Each OI-model has an ontology structure associated with it. The ontology structure is a particular view of the OI-model, containing definitions specifying how instances should be constructed. It consists of concepts (to be interpreted as sets of elements) and properties (to be interpreted as relations between elements). Properties can have domain concepts, and relation properties can have range concepts, which constrain the types of instances to which the properties may be applied. If these constraints are not satisfied, then an ontology is inconsistent. Domain and range concepts define schema restrictions and are treated conjunctively – all of them must be fulfilled for each property instantiation. This has been done in order to maintain compatibility with various description logics dialects (e.g. OWL), which treat domains and ranges conjunctively. Relational properties may be marked as transitive and/or symmetric, and it is possible to say that two relational properties are inverse of each other. For each class-property pair it is possible to specify the minimum and maximum cardinalities, defining how many times a property can be instantiated for instances of that class. Concepts (properties) can be arranged in a hierarchy, as specified by the H_C (H_P) relation. This relation relates directly connected concepts (properties), whereas its reflexive transitive closure follows from the semantics, as defined in the next subsection.

Definition 3 (Instance Pool Structure). *An instance pool associated with an OI-model is a 4-tuple* $IP(OIM) := (I, L, instconc, instprop)$ *where:*

- $I \subseteq E$ *is a set of instances,*
- L *is a set of literal values,* $L \cap E = \emptyset$*,*
- *function instconc* $: C \rightarrow 2^I$ *relates a concept with a set of its instances,*
- *function instprop* $: P \times I \rightarrow 2^{I \cup L}$ *assigns to each property-instance pair a set of instances related through given property.*

Each OI-model has an instance pool associated with it. An instance pool is constructed by specifying instances of different concepts and by establishing property instantiation between instances. Property instantiations must follow the domain and range constraints, and must obey the cardinality constraints, as specified by the semantics.

Definition 4 (Root OI-model Structure). *Root OI-model is defined as a particular, well-known OI-model with structure ROIM* $:= (\{\text{KAON:ROOT}\}, \emptyset)$. KAON:ROOT *is the root concept, each other concept must subclass* KAON:ROOT *(it may do so indirectly).*

Each other OI-model must include ROIM and thus gain visibility to the root concept. Many knowledge representation languages contain the TOP concept which is a superconcept of all other concepts. This is also similar to many object-oriented languages – for example, in Java, every class extends java.lang.Object class.

Definition 5 (Modularization Constraints). *If OI-model OIM imports some other OI-model* OIM_1 *(with elements marked with subscript 1), that is, if* $OIM_1 \in INC(OIM)$*, then following modularization constraints must be satisfied:*

- $E_1 \subseteq E$, $C_1 \subseteq C$, $P_1 \subseteq P$, $R_1 \subseteq R$, $T_1 \subseteq T$, $INV_1 \subseteq INV$, $H_{C1} \subseteq H_C$, $H_{P1} \subseteq H_P$,
- $\forall p \in P_1 \;\; domain_1(p) \subseteq domain(p)$,
- $\forall p \in P_1 \;\; range_1(p) \subseteq range(p)$,
- $\forall p \in P_1, \forall c \in C_1 \;\; mincard_1(c,p) = mincard(c,p)$,
- $\forall p \in P_1, \forall c \in C_1 \;\; maxcard_1(c,p) = maxcard(c,p)$,
- $I_1 \subseteq I$, $L_1 \subseteq L$,
- $\forall c \in C_1 \;\; instconc_1(c) \subseteq instconc(c)$,
- $\forall p \in P_1, i \in I_1 \;\; instprop_1(p,i) \subseteq instprop(p,i)$.

In another words, if an OI-model imports some other OI-model, it contains all information from the included OI-model – no information may be lost. It is possible to add new definitions, but not possible to remove existing definitions (for example, it is possible to add a domain concept to a property in the included model, but it is not possible to remove one). Modularization constraints just specify structural consequences of importing an OI-model, independently from the implementation. For example, in some cases imported OI-models may be physically duplicated, whereas in other cases they may be linked to the importing model without making a copy.

Definition 6 (Meta-concepts and Meta-properties). *In order to introduce meta-concepts, the following constraint is stated:* $C \cap I$ *may, but does not need to be* $\emptyset$*. Also,* $P \cap I$ *may, but does not need to be* $\emptyset$*.*

The same element may be used as a concept and as an instance, or as a property and as an instance in the same OI-model. This has significant impact on the semantics of the model, which is described in section 3.2.

Definition 7 (Lexical OI-model Structure). *Lexical OI-model structure LOIM is a well-known OI-model with the structure presented in Figure 1*[1].

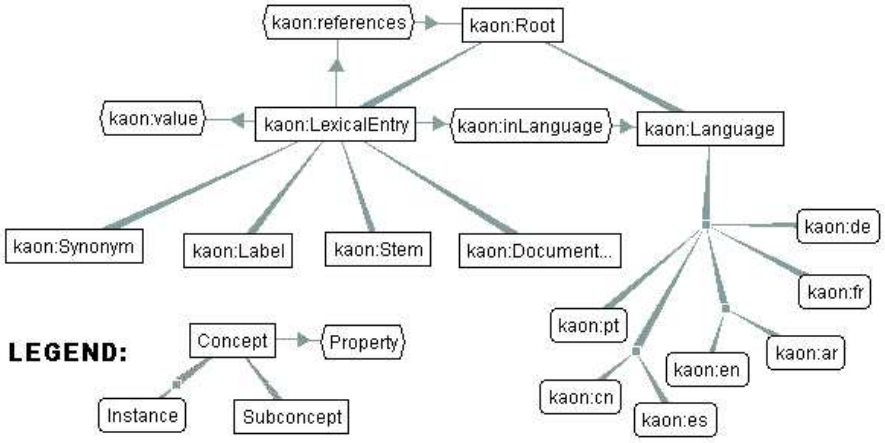

Fig. 1. Lexical OI-Model Structure

Lexical entries (instances of the KAON:LEXICALENTRY concept) reflect various lexical properties of ontology entities, such as a label, stem or textual documentation. There is an n : m relationship between lexical entries and instances, established by the KAON:REFERENCES property. Thus, the same lexical entry may be associated with several elements (e.g. jaguar label may be associated with an instance representing a Jaguar car or a jaguar cat). The value of the lexical entry is given by property KAON:VALUE, whereas the language of the value is specified through the KAON:INLANGUAGE property. Concept KAON:LANGUAGE represents the set of all languages, and its instances are defined by the ISO standard 639. The lexical structure is not closed, as it is the case in most languages. On the contrary, it is possible to define other type of lexical information by providing additional subclasses of the KAON:LEXICALENTRY concept.

A careful reader may have noted that LOIM defines the KAON:REFERENCES property to have the KAON:ROOT concept as the domain. In another words, this means that each instance of KAON:ROOT may have a lexical entry. This excludes concepts from having lexical entries – concepts are subclasses, and not instances of root. However, it is possible to view each concept as an instance of some other concept (e.g. the KAON:ROOT concept), and thus to associate a lexical value with it.

3.2 Denotational Semantics

In this subsection we give meaning to OI-models through denotational semantics in the spirit of description logics. The main distinction of our definition lies in the support for meta-concept modeling, the semantics of domains and ranges and treatment of cardinalities. These distinctions are discussed after the formal presentation of the semantics.

Definition 8 (OI-model Interpretation). *An interpretation of an OI-model OIM is a structure* $I = (\triangle^I, \triangle_D, E^I, L^I, C^I, P^I)$ *where:*

[1] Instead a formal definition, we present a graphical view of LOIM because we consider it to be more informative.

- $\triangle^I$ *is the set of interpretations of objects,*
- $\triangle_D$ *is the concrete domain for data types,* $\triangle^I \cap \triangle_D = \emptyset$*,*
- $E^I : E \rightarrow \triangle^I$ *is an entity interpretation function that maps each entity to a single element in a domain,*
- $L^I : L \rightarrow \triangle_D$ *is a literal interpretation function that maps each literal to an element of the concrete domain,*
- $C^I : \triangle^I \rightarrow 2^{\triangle^I}$ *is a concept interpretation function by treating concepts as subsets of the domain,*
- $P^I : \triangle^I \rightarrow 2^{\triangle^I \times (\triangle^I \cup \triangle_D)}$ *is a property interpretation function by treating properties as relations on the domain.*

An interpretation is a model of OIM if it satisfies the following properties:

- $C^I(E^I(\textit{kaon:Root})) = \triangle^I$*,*
- $\forall c, i \;\; i \in instconc(c) \Rightarrow E^I(i) \in C^I(E^I(c))$*,*
- $\forall c_1, c_2 \;\; (c_1, c_2) \in H_C \Rightarrow C^I(E^I(c_1)) \subseteq C^I(E^I(c_2))$*,*
- $\forall p, i, i_1 \;\; i_1 \in instprop(p, i) \wedge i_1 \in E \Rightarrow (E^I(i), E^I(i_1)) \in P^I(E^I(p))$*,*
- $\forall p, i, x \;\; x \in instprop(p, i) \wedge x \in L \Rightarrow (E^I(i), L^I(x)) \in P^I(E^I(p))$*,*
- $\forall p, x, y \;\; p \in R \wedge (x, y) \in P^I(E^I(p)) \Rightarrow y \in \triangle^I$*,*
- $\forall p, x, y \;\; p \in P \setminus R \wedge (x, y) \in P^I(E^I(p)) \Rightarrow y \in \triangle_D$*,*
- $\forall p_1, p_2 \;\; (p_1, p_2) \in H_P \Rightarrow P^I(E^I(p_1)) \subseteq P^I(E^I(p_2))$*,*
- $\forall s \;\; s \in S \Rightarrow P^I(E^I(s))$ *is a symmetric relation,*
- $\forall p, ip \;\; (p, \textit{ip}) \in \textit{INV} \Rightarrow P^I(E^I(ip))$ *is an inverse relation of* $P^I(E^I(p))$*,*
- $\forall t \;\; t \in T \Rightarrow P^I(E^I(t))$ *is a transitive relation,*
- $\forall p, c, i \;\; c \in \textit{domain}(p) \wedge (\exists x \;\; (E^I(i), x) \in P^I(E^I(p))) \wedge E^I(i) \notin C^I(E^I(c)) \Rightarrow$ *ontology is inconsistent,*
- $\forall p, c, i \;\; c \in \textit{range}(p) \wedge (\exists x \;\; (x, E^I(i)) \in P^I(E^I(p))) \wedge E^I(i) \notin C^I(E^I(c)) \Rightarrow$ *ontology is inconsistent,*
- $\forall p, c, i \;\; E^I(i) \in C^I(E^I(c)) \wedge \textit{mincard}(c, p) > |\{\, y \mid (E^I(i), y) \in P^I(E^I(p))\}| \Rightarrow$ *ontology is inconsistent,*
- $\forall p, c, i \;\; E^I(i) \in C^I(E^I(c)) \wedge \textit{maxcard}(c, p) < |\{\, y \mid (E^I(i), y) \in P^I(E^I(p))\}| \Rightarrow$ *ontology is inconsistent.*

OIM is unsatisfiable it is doesn't have a model. Following definitions say what can be inferred from an OI-model:

- $H_C^* \subseteq C \times C$ *is the reflexive transitive closure of the concept hierarchy if:*
  in all models $C^I(E^I(c_1)) \subseteq C^I(E^I(c_2)) \Leftrightarrow (c_1, c_2) \in H_C^*$*,*
- $H_P^* \subseteq P \times P$ *is the reflexive transitive closure of the property hierarchy if:*
  in all models $P^I(E^I(p_1)) \subseteq P^I(E^I(p_2)) \Leftrightarrow (p_1, p_2) \in H_P^*$*,*
- $\textit{instconc}^* : C \rightarrow 2^I$ *represents inferred information about instances of a concept if:*
  in all models $E^I(i) \in C^I(E^I(c)) \Leftrightarrow i \in \textit{instconc}^*(c)$*,*
- $\textit{instprop}^* : P \times I \rightarrow 2^{I \cup L}$ *represents the inferred information about instances if:*
  in all models $i_2 \in \textit{instprop}^*(p, i_1) \Leftrightarrow (E^I(i_1), E^I(i_2)) \in P^I(E^I(p)) \wedge$
  in all models $l \in \textit{instprop}^*(p, i) \Leftrightarrow (E^I(i), L^I(l)) \in P^I(E^I(p))$*,*
- $\textit{domain}^*(p) = \bigcup_{(p,p_1) \in H_P^*} \textit{domain}(p_1)$ *denotes all domain concepts of a property,*
- $\textit{range}^*(p) = \bigcup_{(p,p_1) \in H_P^*} \textit{range}(p_1)$ *denotes all range concepts of a property,*

Meta-modeling. In usual presentation of first-order semantics, the sets of concept, property and instance symbols are disjoint. The interpretation assigns an element of the domain set to each instance, a subset of the domain set to each concept and a relation on the domain set to each property symbol.

We, on the other hand, allow for each symbol to be interpreted as a concept, a property or an instance, and thus obtain a first-order logic with second-order flavor. In [12] it has been proven that such logic has a sound and complete proof theory, which justifies its usage for ontology modeling. Hence, function E^I associates a domain object with each entity symbol, and functions C^I and P^I provide concept and property interpretations to these objects.

Returning to the example presented in section 2, information about species, ape species and apes may be modeled as in the Figure 2. In this model element APE plays a dual role. Once it is treated as a concept, in which it has the semantics of a set, and one can talk about the members of the set, such as APE1. However, the same object may be treated as an instance of the (meta-)concept SPECIES, thus allowing information such as the type of food to be attached to it. Both interpretations of the element SPECIES are connected by the dotted line, which represents the so called spanning object relationship between the concept and the instance interpretation of the APE symbol.

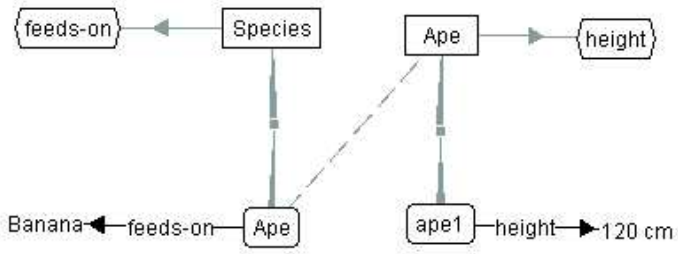

Fig. 2. Meta-Modeling Example

In [44] the problems of considering concepts as instances are well explained. The proposed solution is to isolate different domains of discourse. What is a concept in one domain, may become an instance in a higher-level domain. Elements from two domains are related through so called spanning objects. Our approach builds on that, however, without explicit isolation of domains of discourse. This has subtle consequences on how an OI-model should be interpreted. It is not allowed to ask "what does entity e represent". Instead, one must ask a more specific question: "what does e represent if it is considered as either a concept, a property or an instance". Thus, the interpreter must filter out a particular view of the model – it is not possible to consider multiple interpretations simultaneously. However, it is possible to move from one interpretation to another – if something is viewed as a concept, it is possible to switch to a different view and to look at the same thing as an instance.

Our approach is clearly different from that presented in [37], where the initial RDFS semantics has been criticized for its infinite meta-modeling architecture that may un-

der some circumstances cause Russell's paradox. That paper proposes a fixed four-layer meta-modeling architecture called RDFS(FA) that introduces a strict separation between concepts and instances. Concepts are part of the ontology layer, whereas instances are part of the instance layer.

Our approach to defining semantics is more similar to the RDFS-MT [23], which avoids the Russell's paradox in a similar way as we do – the same name can be interpreted in many ways, but these interpretations still are isolated from one another. However, contrary to RDFS-MT, we still separate the modeling primitives, such as subconcept or subproperty relations into an ontology language layer. In another words, the RDFS:SUBCONCEPTOF property is not available within the model, but exists in the ontology language layer. This approach has been chosen to avoid ambiguities when modeling primitives themselves are redefined (e.g. what semantics does a subproperty of RDFS:SUBCONCEPTOF have?). In effect, our approach is similar to RDFS-FA, but with only three layers – the RDFS-FA ontology and instance layer have been merged into one OI-model layer and the multiple interpretations of some symbol are allowed.

Domains and Ranges. Our definition of domains and ranges differs from that of RDFS and OWL. In these languages, domain and range specifications are axioms specifying sufficient conditions for an instance to be a member of some class. For example, for the ontology from Figure 3, although A is not explicitly stated to be an instance of DOMAIN, because it has PROPERTY instantiated and because PROPERTY has DOMAIN as domain, it can be inferred that A is an instance of DOMAIN. The semantics of this approach would be captured by replacing the domain and range restrictions with following conditions:

- $\forall p, c, i \;\; c \in \text{domain}(p) \wedge (\exists x \;\; (E^I(i), x) \in P^I(E^I(p))) \Rightarrow E^I(i) \in C^I(E^I(c))$,
- $\forall p, c, i \;\; c \in \text{range}(p) \wedge (\exists x \;\; (x, E^I(i)) \in P^I(E^I(p))) \Rightarrow E^I(i) \in C^I(E^I(c))$.

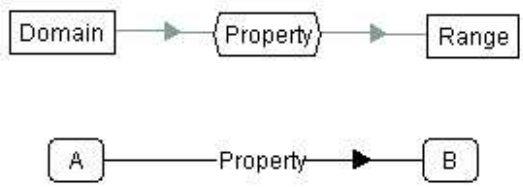

Fig. 3. Treatment of Domains and Ranges

From our experience, while sometimes such inferencing may indeed be useful, often it is not needed, or even desired in closed environments, such as e.g. presented by most knowledge management applications. Most users without a formal background in logic, but with strong background in databases and object-oriented systems, intuitively expect domains and ranges to specify the constraints that must be fulfilled while populating ontology instances. In another words, unless A is known to be an instance of DOMAIN,

then PROPERTY cannot be instantiated for A in the first place, or the ontology becomes inconsistent. This approach has the following benefits:

- Treating domains and ranges as constraints makes it possible to guide the user in the process of providing information about instances. It is easy to compute the set of properties that can be applied to an instance, and then to ask the user to provide values for them. On the other hand, if domains and ranges are treated as axioms, any property can be applied to any instance, which makes it difficult to constrain user's input.
- Similar problems occur when evolving the ontology. E.g., if B is removed from the extension of the concept RANGE, it can be computed that the PROPERTY between A and B must be removed. On the other hand, if domains and ranges are axioms, then it is not clear how to change the ontology so that it still makes sense.
- Treating domains and ranges as axioms introduces significant performance overhead in query answering. For example, to compute the extension of some concept, it is not sufficient to take the union of the extension of all subconcepts – one must examine a larger part of the instance pool to see which instances may be classified under the concept according to the domain and range axioms. Therefore, if only the constraint semantics is needed, the system will suffer from unnecessary performance overhead.

For ontologies that are consistent under the constraint semantics, interpretations under both definitions of the semantics match. This may be easily seen, since if ontology isn't inconsistent, the negated conditions for instance membership can be moved from the left to the right side of the implication, by which the semantics becomes identical to the axiom semantics. Hence, the choice of semantics doesn't change the model theory or allowed entailments for (the common case of) ontologies consistent under constraint semantics.

Cardinalities. In our ontology modeling approach we treat cardinalities as constraints regulating the number of property instances that may be specified for instances of each concept. This is different from OWL and other description logic languages, where cardinalities are axioms specifying that instances with particular number of property instances to some concept can be inferred to be instances of some concept. We find that constraining the number of property instances that are allowed for some instance is extremely useful for guiding the user in providing ontology instances. By knowing how many property instances can be provided for instances of some concept, the user can be asked to provide the appropriate number of values. Similar arguments as in the case of domain and range semantics apply here as well.

3.3 Example

Next we present an example OI-model. A common problem in knowledge management systems is to model documents classified into various topic hierarchies. With each document we associate with it an author. Being an author is just a role of a person, and a single person may have other roles at the same time (e.g. it may be a researcher). This domain may be conceptualized as follows:

- Documents are modeled as instances of the DOCUMENT concept.
- Topics are modeled as instances of the TOPIC concept.
- There is a SUBTOPIC transitive property specifying that a topic is a subtopic of another topic.
- There is a HAS-AUTHOR property between a document and an author. Each author has a property HAS-WRITTEN that is inverse to HAS-TOPIC.
- There is a HAS-TOPIC property between a document and a topic.
- Persons are modeled as instances of the PERSON concept.
- For each role there are subconcepts of PERSON. We present two: RESEARCHER and AUTHOR.

The hierarchy of topics is self-contained unit of information, so it is reasonable to assume there will be a need to reuse it. Therefore, it will be factored out into a separate OI-model called TOPICS. Similarly, the personnel information will be separated into human resources (HR) OI-model, and the document information will be separated into DOCUMENT OI-model. Within a company it doesn't make sense to reuse just the ontology of HR – all information about people is centralized in one place. Therefore, HR OI-model will contain both ontology definitions as well as instances. Similar arguments may be made for other OI-models. Figure 4 shows these OI-model graphically.

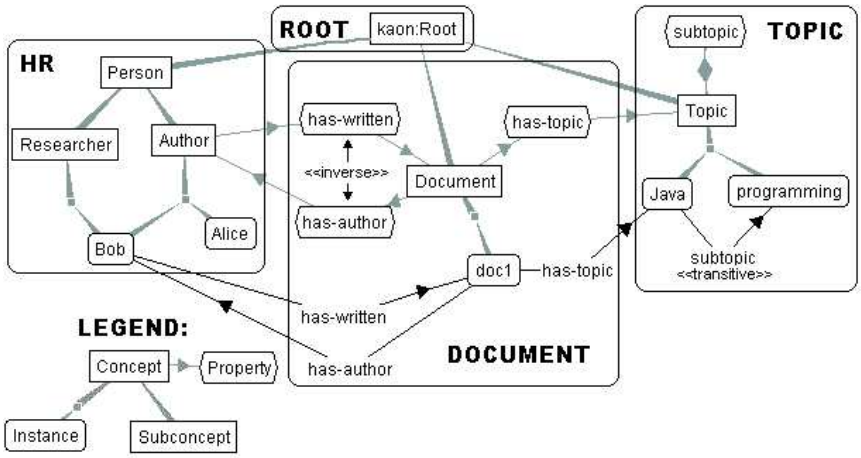

Fig. 4. Example Ontology

The semantic interpretation of the example is as follows. All elements modeled as concepts may be interpreted as sets: DOCUMENT is a set of all documents, PERSON is the set of all persons and AUTHOR is a subset of PERSON representing those persons that are authors as well. We may interpret BOB as a member of sets PERSON and AUTHOR. DOCUMENT as a member of set KAON:ROOT. TOPIC is a set whose instances are individual topics, such as JAVA. Subtopic relationship is modeled using a transitive property. Inverse properties allows navigation in both directions.

4 Ontology Querying

Apart from representing domains with low conceptual gap, a way of querying ontologies is needed. Querying is closely related to inference – computing information which is not explicitly part of the model, but logically follows from the model. The query and inference capabilities available depends heavily on the ontology modeling approach. In this section we discuss some existing approaches to ontology querying and inference, and present our approach.

4.1 Approaches to Ontology Querying

We base our approach for ontology querying on the observation that a query can be seen as a function which, when applied to a data model, instantiates some results. As discussed in [43], it is beneficial if the domain and range models of queries are the same. In such way, queries can be composed functionally, giving the user a considerable degree of freedom.

In the case of relational data model and relational algebra, this condition is fully satisfied, as the domain and range of algebra queries are both relational models, thus allowing the user to combine queries arbitrarily (e.g. through nesting). On the other hand, the relational model doesn't directly represent the semantics of modeled information and is thus not applicable to ontology modeling.

In other cases, e.g. in F-Logic [28], the domain of queries is the ontology model, but the range is a list of variable bindings for which the query is satisfied in the model. We see several drawbacks to this approach. For one, F-Logic queries can't be composed functionally, resulting in a more complex query language. Further, F-Logic queries destroy the semantics of the information. As an example let's consider an ontology consisting of people linked to the documents that they've written. A query selecting all persons with their corresponding documents, written in F-Logic as "FORALL P,D $\leftarrow$ P:Person[hasWritten $\twoheadrightarrow$ D]", may produce the result as in the table 1.

P	D
John	Report1
John	Tutorial2
Bob	Calculation1

Table 1. Result of an Example F-Logic Query

We may observe that John has written two documents, but the query result reflects this fact by repeating the value John multiple times for P. The explicit information that there is an n:m relationship between persons and documents has been lost in the query process, and is now available only by analyzing the intent of the query. This relationship may be reconstructed from the query result by collecting all values of D for which P is the same. In our view this makes the interpretation of the query external to the query itself, which doesn't follow the general desire to represent the semantics of information directly.

4.2 Conceptual Querying

In order to avoid problems stated in the previous subsection, in our approach we focus on conceptual querying based on the observation that the basic ontology elements are concepts and properties, so to remain within the model, queries should generate concepts and properties as their results. This is a natural extension of the ontology model definition – concepts are sets of instances and queries are set of things that match some conditions. We may naturally unify these two constructs and simply say that concepts can either be primitive, in which case they have explicitly defined extension (as described in the section 3), or can be defined intensionally, by specifying a condition constraining instance membership. In such spirit the enumeration of concept's instances (primitive or otherwise) becomes the fundamental primitive for retrieving information from ontologies.

Our approach to querying ontologies is significantly influenced by description logics [7], a family of subsets of first-order logic. Central to the description logics paradigm is specifying concepts not only by enumerating individual instances, but by specifying necessary and/or sufficient conditions for instance membership. There are some differences, though. First, in description logics the focus is on inferring concept subsumption, i.e. deciding which concept is more specific than some other concept by examining only the definition of the concept. To obtain a sound and complete subsumption inference procedures, the expressivity of the logics is sacrificed. In our case the focus is on efficient computation of concept's extension, while the soundness and completeness of concept subsumption inference becomes of secondary importance. While description logics provide excellent ways of specifying structural conditions, they don't provide enough power for filtering. For example, in description logics it is not possible to form queries such as "select all persons that are older then their spouses".

Next we present our query approach in more detail. The queries may be concept or property expressions. Table 2 specifies the basic concept and property expressions, which can be combined into more complicated expressions. Enumerating a concept expression results with a set of member instances, whereas enumerating a property expression results with a set of pairs. The semantics of concept and property expressions is given in model-theoretic way, building on the semantics presented in subsection 3.2. We extend the function E^I to assign elements of the domain not only to entities, but to concept and property expressions as well.

Some primitives from table 2 may be expressed using other primitives. However, we chose to include them directly in the definition of our language because they benefit user's understanding of the language. Primitives for property navigation, concept cross product and application of functions to concept and property members are unique to our approach and typically do not appear in description logics. Next we give query examples of several queries (the characters [], <> and !! delimit the URIs of concepts, properties and instances, respectively).

The following is a typical description logic query which retrieves all persons working on a research project:

```
[#Person]  AND  SOME(<#worksOn>,[#ResearchProject])
```

The following query allows easy navigation across properties and retrieves all persons working on a project sponsored by the EU:

Expression	Semantics
c AND d	$C^I(E^I(c)) \cap C^I(E^I(d))$
c OR d	$C^I(E^I(c)) \cup C^I(E^I(d))$
NOT c	$\triangle^I \setminus C^I(E^I(c))$
SOME(p,c)	$\{x \mid \exists y \ \ (x,y) \in P^I(E^I(p)) \wedge y \in C^I(E^I(c))\}$
ALL(p,c)	$\{x \mid \exists y \ \ (x,y) \in P^I(E^I(p)) \Rightarrow y \in C^I(E^I(c))\}$
MUST_BE(p,c)	SOME(p,c) AND ALL(p,c)
{ i }	$\{E^I(i)\}$
FROM(p)	$\{x \mid \exists y \ \ (x,y) \in P^I(E^I(p))\}$
TO(p)	$\{y \mid \exists x \ \ (x,y) \in P^I(E^I(p))\}$
f(c)	$\{f(x) \mid x \in C^I(E^I(c))\}$
INVERSE(p)	$\{(y,x) \mid (x,y) \in P^I(E^I(p))\}$
CROSS(c,d)	$\{(x,y) \mid x \in C^I(E^I(c)) \wedge y \in C^I(E^I(d))\}$
p IN:1 c	$\{(x,y) \mid (x,y) \in P^I(E^I(p)) \wedge x \in C^I(E^I(c))\}$
p IN:2 c	$\{(x,y) \mid (x,y) \in P^I(E^I(p)) \wedge y \in C^I(E^I(c))\}$
p.r	$\{(x,z) \mid (x,y) \in P^I(E^I(p)) \wedge (y,z) \in P^I(E^I(r))\}$
f:1(p)	$\{(f(x),y) \mid (x,y) \in P^I(E^I(p))\}$
f:2(p)	$\{(x,f(y)) \mid (x,y) \in P^I(E^I(p))\}$

Table 2. Query Syntax and Semantics

```
[#Person]  AND  SOME(<#worksOn>.<#sponsored>={!#EU!})
```

The following query applies a function to the elements of a concept to retrieve the subconcepts of the PROJECT concept. An interesting point about this query is that it is supposed to return the set of concepts as result. However, concepts can contain only instances, so a trick is applied. The query starts with a set of instances, interprets them as concepts (by traversing to the spanning concept), retrieves the subconcepts and returns the spanning instances of these concepts:

```
SUBCONCEPTS({!#Project!})
```

Schema queries can be easily chained. E.g., retrieving all concepts two levels under the PROJECT concept can be done in this way (SUBCONCEPTS^ function returns only direct subconcepts):

```
SUBCONCEPTS^(SUBCONCEPTS^({!#Project!}))
```

4.3 Navigation as a Complement for Queries

It is obvious that presented query language itself can't be used to express any query. For example, retrieving all documents with their authors and topics can't be done in one query, because our queries can return only sets of items, not sets of tuples. However, this deficiency of the query langauge is compensated with the capability for navigation within the model. In our view the role of the query language is to provide entry points into the model, while retrieval of all ancillary information to each entry point is obtained using navigational primitives.

Retrieving all documents, their authors and topics can then be performed in the following way: First the list of relevant documents using the query language is obtained.

Each document is then processed in a loop, where for each document the set of authors is retrieved by traversing the HASAUTHOR property and the set of topics is retrieved by traversing the HASTOPIC property. By doing that, the primitives for information retrieval follow closely the semantic structure of the ontology, so the semantics of the underlying information remains preserved.

However, there is a significant performance drawback in the specified approach. If there are n documents, then there will be $2 * n + 1$ query requests issued (one for retrieving all documents and the two queries per document), which will clearly be a major cause of performance problems. Further, the order in which the information is retrieved is predefined by the order in which the navigational primitives are issued. This prevents the usage of various query optimization strategies that are considered to be the key factor for the success of relational databases systems. There seems to be a mismatch between the desire to retrieve as much information as needed on one side (to reduce communication overhead and to allow for optimizations) and to retrieve information using navigation on the underlying model (in order not to destroy the model's semantics).

To avoid such problems, we apply the following approach. Our queries can still be treated as concepts and return individuals or individual pairs at most. However, with each main query we allow specifying several additional queries. There queries aren't part of the query result, but they are used as hints specifying which information is needed as well. For example, selecting all documents, their authors and topics can be done like this:

```
[#Document]
   [<#hasAuthor> IN:1 this]
   [<#hasTopic> IN:1 this]
```

The second and the third line specify that HASAUTHOR and HASTOPIC properties with their first element equal to elements in the main query are of interest. Such additional queries can be treated as nested queries and query unnesting [17] algorithms can be applied. Thus all necessary information can be retrieved, while allowing the database to apply any optimization strategy. The results of the query must be nested and can be accessed using navigational primitives.

5 Implementation

In this section we present how our ontology modeling and querying approach is applied to KAON [35] – a platform for developing and deploying semantics-driven enterprise applications. First we present KAON API – the central part of the system – and discuss its role within the overall KAON architecture. Next we discuss possible database schemata for storing ontologies.

5.1 KAON API within KAON Architecture

The manipulation of the ontologies in KAON is performed using KAON API – a set of interfaces offering access and manipulation of ontologies and instances. A UML view of the KAON API is shown in figure 5.

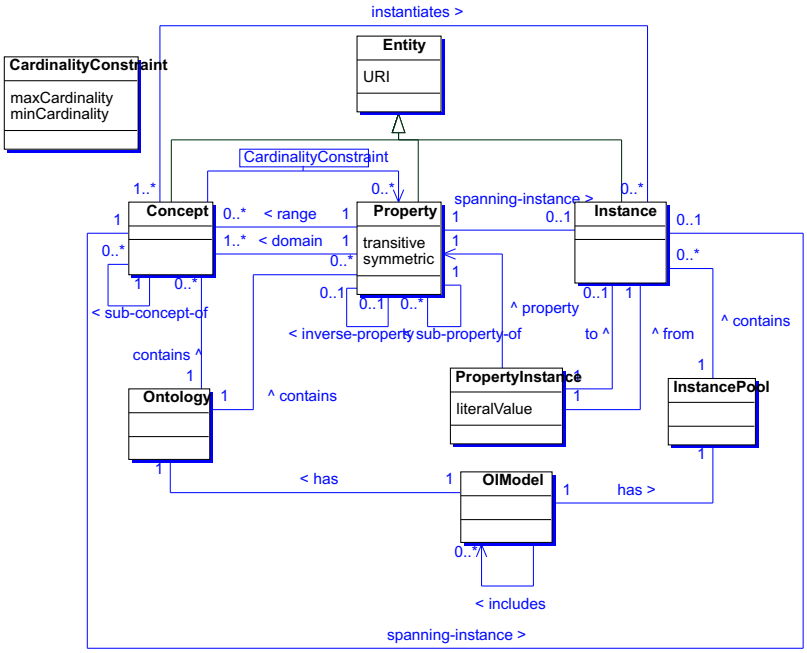

Fig. 5. UML View of KAON API

The API closely follows the definitions presented in 3.1. For example, an OI-model from definition 1 is represented as an OIModel object, that may include other OI-models according to modularization constraints from definition 5. Ontology and instance pool objects are associated with each OI-model. As per definition 2, an ontology consists of concepts (represented as Concept objects) and properties (represented as Property objects). A property has domain and range restrictions, as well as cardinality constraints. It may be an inverse of some other property, and may be marked as being transitive or symmetric. As per definition 3, an instance pool consist of instances (represented through Instance objects) that may be linked with other instances through property instances (represented as PropertyInstance objects). Each concept or property may have an instance associated with it through a spanning object.

KAON API is embedded within the KAON conceptual architecture consisting of three layers as described in [8]. Building on top of KAON API, various tools have been realized within the Application & Services layer, such as OI-modeler for ontology and meta-data engineering and KAON Portal for (semi-)automatic generation of Web portals from ontologies have been realized. These tools are responsible for providing the user interface.

KAON API is realized on top of the Data and Remote Services layer that is responsible for realizing typical business-related requirements, such as persistence, reliability, transaction and concurrency support. The layer is realized within an EJB application server and uses relational databases for persistence. Apart from providing abstractions for accessing ontologies, KAON API also decouples actual sources of ontology data by

offering different API implementations for various data sources. Following API implementations may be used:

Engineering Server. Engineering server is an implementation of the KAON API that may be used for ontology engineering. This implementation provides efficient implementation of operations that are common during ontology engineering, such as concept adding and removal in a transactional way. The schema for this API implementation is described in the following subsection.

Implementation for RDF repository access. An implementation of KAON API based on RDF API[2] may be used for accessing RDF repositories. This implementation is primarily useful for accessing in-memory RDF models under local, autonomous operation mode. However, it may be used for accessing any RDF repository for which an RDF API implementation exists. KAON RDF Server is such a repository that enables persistence and management of RDF models and is described in more detail in [42].

Implementation for accessing any database. An implementation of KAON API may be used to lift existing databases to the ontology level. To achieve that, one must specify a set of mappings from some relational schema to the chosen ontology, according to principles described in [41]. E.g. it is possible to say that tuples of some relation make up a set of instances of some concept, and to map foreign key relationships into instance relationships. After translations are specified, a virtual OI-model is generated, which will on each access translate the request into native database queries. In such way the persistence of ontology information is obtained, while reusing well-known database mechanisms such as transactional integrity.

This architecture fulfills requirements stated in section 2. We consider it suitable for enterprise-wide application because of the following reasons:

- Well-known technologies are used for ontology persistence and management, such as EJB servers and relational databases.
- These technologies already realize many of the needed requirements: transactions, concurrent processing and data integrity comes ”for free”.
- Because of the structure of ontologies, majority of queries may be executed by the underlying databases, thus ensuring scalability.

5.2 Persisting Ontologies

In this section we present our approach for persisting ontologies in engineering server. The name 'engineering server' comes from the fact that the persistence component is optimized for management of ontologies during the ontology engineering process. Ontology engineering is a cooperative consensus building process, during which many ontology modelers experiment with different modeling possibilities. As a result, addition, removal, merging and splitting of concepts are operations whose performance is most critical. Further, since several people are working on the same data set at once, it

[2] We reengineered Sergey Melnik's RDF API: http://www-db.stanford.edu/ melnik/rdf/api.html

is important to support transactional ontology modification. At the same time, accessing efficiently large data sets is not so critical during ontology engineering. In order to support such requirements, a generic database schema, presented in Figure 6, is used. The schema is a straightforward translation of the definitions 1, 2 and 3 into relational model.

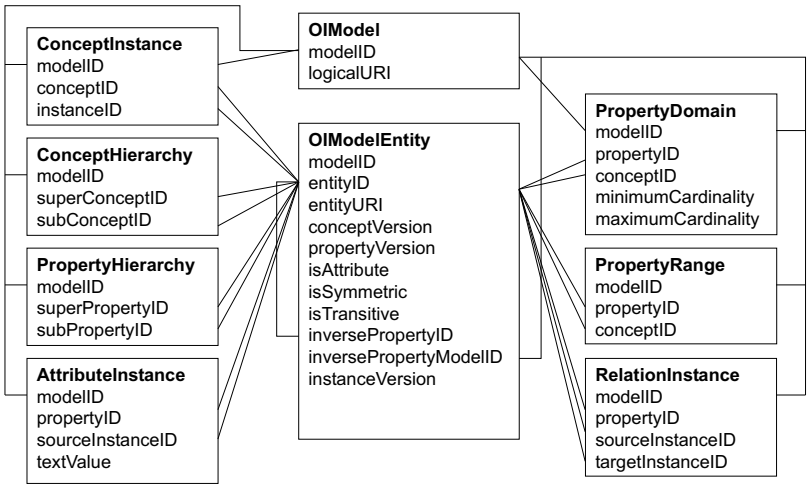

Fig. 6. Generic KAON Schema

The schema is organized around the OIModelEntity table, whose single row represents a concept, instance and property attached to a single URI. This structure has been chosen due to the presence of meta-class modeling – by keeping all information within one table it is possible to access all information about an entity at once.

Our schema design differs from the schema of RDF Query Language (RQL) [26] of the RDFSuite toolkit [2] in one significant point. In the RQL implementation, a unary table is assigned to each concept, where rows identify the concept's instances. Clearly, such schema will offer much better performance, but has a significant drawback – adding (removing) a concept requires creating (removing) a table in the database. Such operations are quite expensive and may not performed within a transaction. However, our goal is to design a system supporting users in cooperative ontology engineering, where creation and removal of concepts is quite common and must be transactional, while run-time performance is not that critical.

5.3 Evaluating Concept Queries

As mentioned in section 4, efficient evaluation of extensions of intentionally defined concepts is of central importance in our approach for ontology querying. Most description logic reasoners (e.g. FaCT [24]) use tableaux reasoning [3] for proving whether an instance is a member of a concept. Such an inference procedure is needed because in

description logics any concept can be instantiated. For example, in description logics it is possible to specify that i is an instance of the concept A OR B. In this case, it is not known whether i is in A or in B (or perhaps in both), but if A OR B is subconcept of some other concept C, then it can be inferred that i is in C. To implement such inferences, a system providing disjunctive and incomplete reasoning is needed.

Our case is much simpler, since currently only primitive concepts can be instantiated. Hence, a statement from the previous paragraph can't be made in the first place. Therefore, we may choose more traditional techniques for evaluating queries, which build on the rich experience of deductive database systems. Such evaluation strategies are much more efficient, and lend themselves well to implementation on top of relational database systems.

To evaluate queries we perform a translation of concept and property expressions into datalog programs, which we then evaluate bottom-up [1] using a semi-naive evaluation strategy. In order to restrict the amount of information retrieved from the extensional database only to relevant facts, we perform a generalized magic sets transformation of the program against the query asked [5]. Briefly, magic sets transformation simulates top-down through bottom-up computation, by applying sideways information passing, which propagates bindings of variables between predicates. Further, the magic program is extended with magic predicates containing information about the facts which are relevant to the query. Particular care has to be taken in case negation is used, since it is known that magic sets transformation of stratified programs can result in unstratified programs. In this case results from [27] are used to compute the well-founded model of the magic program, which has been proven to be equal to the minimal model of the original stratified program. By imposing some restrictions on the sideways information passing strategy, it is even possible to compute the well-founded model of a non-stratified datalog program using magic sets.

$\mu[c \text{ AND } d](x) \equiv \mu[c](x), \mu[d](x)$
$\mu[c \text{ OR } d](x) \equiv p(x)$ with two additional rules $p(x) \leftarrow \mu[c](x).$, $p(x) \leftarrow \mu[d](x).$
$\mu[\text{NOT } c](x) \equiv \mu[\text{kaon:Root}](x), \neg\mu[c](x)$
$\mu[\text{SOME}(p, c)](x) \equiv \mu[p](x, y), \mu[c](y)$
$\mu[\text{ALL}(p, c)](x) \equiv \mu[\text{kaon:Root}](x), \neg p(x)$ with additional rule $p(x) \leftarrow \mu[p](x, y), \neg\mu[c](y).$
$\mu[\{i\}](x) \equiv x = i$
$\mu[\text{FROM}(p)](x) \equiv \mu[p](x, y)$
$\mu[\text{TO}(p)](x) \equiv \mu[p](y, x)$
$\mu[\text{f}(c)](x) \equiv \mu[c](y), f(x, y)$
$\mu[\text{INVERSE}(p)](x, y) \equiv \mu[p](y, x)$
$\mu[\text{CROSS}(c, d)](x, y) \equiv \mu[c](x), \mu[d](y)$
$\mu[p \text{ IN:1 } c](x, y) \equiv \mu[p](x, y), \mu[c](x)$
$\mu[p \text{ IN:2 } c](x, y) \equiv \mu[p](x, y), \mu[c](y)$
$\mu[p.r](x, y) \equiv \mu[p](x, z), \mu[r](z, y)$
$\mu[\text{f:1}(p)](x, y) \equiv \mu[p](z, y), f(x, z)$
$\mu[\text{f:2}(p)](x, y) \equiv \mu[p](x, z), f(y, z)$

Table 3. Translating Queries to Datalog

Extending the work from [22], table 3 defines the transformation operator μ providing a transformation for concept and property expressions.

ConceptHierarchy_t$(X, Y) \leftarrow$ ConceptHierarchy(X, Y).
ConceptHierarchy_t$(X, Y) \leftarrow$ ConceptHierarchy(X, Z), ConceptHierarchy_t(Z, Y).
PropertyHierarchy_t$(X, Y) \leftarrow$ PropertyHierarchy(X, Y).
PropertyHierarchy_t$(X, Y) \leftarrow$ PropertyHierarchy(X, Z), PropertyHierarchy_t(Z, Y).
ConceptInstance_a$(C, I) \leftarrow$ ConceptInstance(C, I).
ConceptInstance_a$(C, I) \leftarrow$ ConceptHierarchy_t(C, SC), ConceptInstance(SC, I).
AttributeInstance_a$(P, S, V) \leftarrow$ AttributeInstance(P, S, V).
AttributeInstance_a$(P, S, V) \leftarrow$ PropertyHierarchy_t(P, SP), AttributeInstance(SP, S, V).
RelationInstance_a$(P, S, T) \leftarrow$ RelationInstance(P, S, T).
RelationInstance_a$(P, S, T) \leftarrow$ PropertyHierarchy_t(P, SP), RelationInstance_a(SP, S, T).
RelationInstance_a$(P, S, T) \leftarrow$ OIModelEntity$(P, ATTR, SIM, TRANS, IP)$, RelationInstance_a$(IP, T, S)$.
RelationInstance_a$(P, S, T) \leftarrow$ OIModelEntity$(P, ATTR,$ true$, TRANS, IP)$, RelationInstance(P, T, S).
RelationInstance_a$(P, S, T) \leftarrow$ OIModelEntity$(P, ATTR, SIM,$ true$, IP)$, RelationInstance_a(P, S, M), RelationInstance_a(P, M, T).
PropertyInstance_a$(P, S, NULL, V) \leftarrow$ AttributeInstance_a(P, S, V).
PropertyInstance_a$(P, S, T, NULL) \leftarrow$ RelationInstance_a(P, S, T).

Table 4. Default Axiomatization of Engineering Server

In case engineering server is used as the underlying storage model for primitive concepts and properties, we represent each table of the server as an extensional database predicate. To capture the semantics of the model specified in subsection 3.2, we use the default axiomatization as specified in the table 4.

μ[concept-URI]$(x) \equiv$ ConceptInstance_a(concept-URI, x)
μ[primitive-relation-URI]$(x, y) \equiv$ RelationInstance_a(primitive-relation-URI, x, y)
μ[primitive-attribute-URI]$(x, y) \equiv$ AttributeInstance_a(primitive-attribute-URI, x, y)

Table 5. Translating Primitive Concepts and Properties to Datalog

Based on such an axiomatization, we complement the rules from table 3 by the rules in table 5, which define how to translate into datalog the references to primitive concepts and properties.

6 Related Work

In this section we discuss how our approach differs from other ontology and conceptual modeling approaches and tools available.

Entity-Relationship and Relational Modeling. In database community the entity-relationship (ER) modeling has established itself as the prevailing conceptual modeling paradigm. ER models consist of entity sets that define a set of attributes for each entity in the set and of relations of any arity between entities. Under recent extensions it is even possible to model subclassing between entities as well. Entity-relationship modeling is just a tool for conceptual databases design. Before implementation these models are transformed into a logical model – nowadays this is the relational model. Evolving and implementing such models is more complex than if only one paradigm were used, since the conceptual and logical models must be kept in synchrony.

In our approach the logical model is hidden from the user. The users may use an ontology in a natural way, while enjoying the benefits of relational database technology for information storage and management.

RDF and RDFS. It has been argued by many (e.g. [37]) that the definition of RDFS is very confusing, because RDFS modeling primitives are used to define the RDFS language itself. Currently there is no specification for modularization of RDF models. The handling of lexical information is very messy – languages can be attached to RDF model using XML attributes. Further, only 1:m relationships between lexical entries and ontology elements are possible.

RQL. Several query languages have been developed for RDF(S), with RDF Query Language (RQL) [26] being one of the most well-known. RQL basically extends the functional approach of object query language (OQL) [11] with capabilities for querying schema as well as instances. This is achieved by creating a type system on top of the RDF model. Thus, each RDF object is given a certain type, and types for collections of objects of other types have been added. Each RQL query primitive can be viewed as a function which takes an object of certain type and creates a new object of some type. To return multiple values, RQL introduces the tuple type as a sequence of values. Many RQL queries result in a collection of tuples, so they destroy the semantics, as discussed in section 4.

UML. There have been proposals for using UML as an ontology modeling language (e.g. [13]). However, the reader may note that there are significant, but subtle, differences in standard object-oriented and ontology modeling. Classes of an object-oriented model are assigned responsibilities, which are encapsulated as methods. Often fictitious classes are introduced to encapsulate some responsibility (also known as pure fabrications [29]). On the other hand, ontologies don't contain methods so responsibility analysis is not important.

The fact that UML is targeted for modeling of software systems containing methods is far-reaching. For instance, an object cannot be an instance of more than one class and, if an object is created as an instance of some class, it is impossible for it to later become

an instance of some other class – an object's class membership is determined at the point when object is created.

In our modeling paradigm, however, these statements do not hold. Membership of an object in a class is often interpreted as statement that some unary predicate is true for this object. If properties of an object change as time passes, then the object should be reclassified. Membership of an object in different classes at the same time has the notion that some different aspects of that object are true simultaneously.

Frame-based Languages. The application of object-oriented modeling paradigm to ontology modeling has resulted in so called frame-based knowledge modeling languages, and F-logic [28] is one example. In F-logic, the knowledge is structured into classes having value slots. Frames may be instantiated, in which case slots are filled with values. F-logic introduces other advanced modeling constructs, such as expressing meta-statements about classes. Also, it is possible to define Horn-logic rules for inferring new information not explicitly present in the model.

Although F-Logic offers a comprehensive ontology model, the queries to F-Logic models result in the set of variable bindings, which aren't within the model. Hence, the semantics of the data is obscured when the model is queried.

Description Logics. A large body of research has been devoted to a subclass of logic-based languages, called description logics (a good overview is presented in [7]). In description logics an ontology consists of individuals (representing elements from the universe of discourse), roles (representing binary relations between individuals) and concepts (sets of elements from the universe of discourse). Concepts may either be primitive or specified through descriptions. Concept descriptions are generated by combining other concept definitions using constructors to specify constraints on the membership of individuals in the concept being defined. The primary inference procedure in description logics is that of subsumption – determining whether one concept description subsumes another one (whether all individuals of the subsumed concept must be individuals of the subsuming concept, regardless of how the instances are specified). Further, it is possible to enumerate all individuals of a concept. A large body of research has examined which concept constructors may be combined, while still allowing for a decidable and tractable inference procedure.

This led to a well-research theory, however, as mentioned in [4], description logics have proven to be difficult to use due to the non-intuitive modeling style. Basically this is due to the mismatch between with predominant object-oriented way of thinking. For example, a common knowledge management problem is to model a set of documents that have topics chosen from a topic hierarchy. A typical solution is to create a DOCUMENT concept acting as a set of individual documents. However, modeling a topic hierarchy is not so straightforward, since it is not clear whether topics should be modeled as concepts or as individuals.

In object oriented systems it is not possible to relate instances of classes with classes. Therefore, a possible approach is to model all topics as members of the concept TOPIC, and to introduce subtopic transitive relation between topic instances. To an experienced object-oriented modeler, this solution will be intuitive.

On the other hand, in description logic systems, since topics are arranged in a hierarchy, the preferred modeling solution is to arrange all topics in a concept hierarchy to rely on the subsumption semantics of description logics[3]. Thus, each topic will be a subtopic of its superconcepts. However, two problems arise:

- If topics are sets, what are the instances of this set? Most users think of topics as fixed entities, and not as (empty and abstract) sets.
- How to relate some document, e.g. d1, to a particular topic, e.g. t1? Documents will typically have a role has-topic that should be created between d1 and topic instance. But, t1 is a concept, and there are no instances of t1. The solution exists, but is not intuitive – we do not specify exactly with which instance of t1 d1 is related, but to say that it is related to "some" instance of t1. In the syntax of CLASSIC [34] description logic system, this is expressed as

$$(createIndividual\ d1\ (some\ has-topic\ t1)).$$

Further, description logics typically don't support inclusion of lexical information into the model. Because of their focus on the sound and complete inference procedures, the power of filtering query results is limited. In description logics it is usually not possible to realize constraints – e.g. domains and ranges are there always treated as axioms (a proposal of overcoming this limitation by using an epistemic operation has been proposed in [15] but to the best knowledge of authors it hasn't been implemented in practice). Finally, to implement description logics fully, very advanced reasoning techniques, such as disjunctive and incomplete reasoning are required. This hinders realization of systems that would fulfill requirements from the section 2.

The appropriate query language for description logics is still under discussion. In [25] an approach for extending description logics with conjunctive querying has been presented. The approach is based on query roll-up, where a conjunctive query over description logics concept is transformed into equivalent description logics boolean query. However, as the authors mention, an efficient algorithm for evaluating such queries is still missing.

OIL. An important ontology modeling language is OIL [18]. This language has tried to combine the intuitive notions of frames, clear semantics of description logics and the serialization syntax of RDF. It proposes a layered architecture. Core OIL defines constructs that equal in expressivity to RDF schema without the support for reification. Standard OIL contains constructs for ontology creation that are based on description logics. However, the syntax of the language hides the description logics background and presents a system that seems to have a more frame-based "feel". However, standard OIL doesn't support creation of instances. Instance OIL defines constructs for creation of instances, whereas heavy OIL is supposed to include the full power of description logics (it hasn't been defined yet).

Despite its apparent frame-based flavor, OIL is in fact a description logic system, thus our comments about description logics apply to OIL as well. OIL does support modularization of ontologies. However, it doesn't have a consistent strategy for management of lexical information.

[3] Thanks to Ian Horrocks for discussion on this topic.

RDFSuite. RDFSuite [2] is a set of tools for management of RDF information developed by the ICS-FORTH institute. It includes a validating parser for RDF, a database system for storing RDF and the RDF Query Language (RQL) – a functional query language for querying RDF repositories.

Our approach differs form the RDFSuite in several aspects. First, the ontology modeling paradigm explained in this paper is richer than that of the RDFSuite, as it includes symmetric, inverse and transitive properties, whereas the RDFSuite is limited to constructs of RDF(S). Second, for manipulation of ontology data KAON API is used, which provides an object-oriented view of the ontology information, whereas RDFSuite relies on queries for accessing the information. The same API is used to isolate applications from actual data sources, thus allowing storage of ontology information not only in relational databases. Finally, the implementation of RQL doesn't support transactional updates for ontology engineering – creation of concepts involves creating tables, which cannot be executed within a transaction.

Sesame. Sesame [10] is an architecture for storing and querying RDF data developed by the Free University of Amsterdam and Aidministrator. It consists of an RDF parser, RQL query module and an administration module for managing RDF models. The differences between our approach and Sesame are similar to those between our approach and RDFSuite. Sesame also supports RDF(S) modeling paradigm but doesn't provide the API for object-oriented access and modification of information.

Jena. Jena [33] is the set of tools for RDF manipulation developed by HP Labs. It implements a repository for RDF(S) and includes RDF Data Query Language (RDQL) implementation for querying. On top of that, it offers support for persistence of DAML ontologies, but with only very basic inferencing capabilities. An API for manipulation of raw RDF information is included, as well as for manipulation of DAML ontologies. However, there is no API for manipulation of RDF(S) ontologies. Further, RDQL query language is based on graph pattern matching, thus preventing the user from issuing recursive queries (e.g. for selecting all subconcepts of some concept).

7 Conclusion

In this paper we present an approach for ontology modeling and querying, currently being developed within KAON. Our main motivation is to come up with an approach that can be used to build scalable enterprise-wide ontology-based application using existing, well established technologies.

In the paper we argue that existing approaches for conceptual modeling lack some critical features, making them unsuitable for application within enterprise systems. From the technical point of view, these features include scalability, reliability and concurrency. From the conceptual modeling point of view, existing approaches lack the support for modularization and concept meta-modeling.

Based on the motivating usage scenarios, a set of requirements has been elicited. A mathematical definition of the ontology language has been provided, along with a denotational semantics. Next an approach for ontology querying adapting and extending

the description logic paradigm has been presented. We have presented the current status of the implementation in the form of the KAON API – an API for management of ontologies and instances, along with the algorithms for evaluating ontology queries.

In future we shall focus on two important aspects of ontology modeling. First, we plan to investigate whether our approach for query execution can be extended to handle more description logics constructs. In particular we are interested in allowing the explicit instantiation of defined concepts while still being able to compute concept extensions efficiently. Second, in order to provide capabilities for modeling constraints and to increase the expressivity of our model, we are interested in finding ways to combine description logics with rules, but without significant performance degradation.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *2nd International Workshop on the Semantic Web (SemWeb'01), in conjunction with Tenth International World Wide Web Conference (WWW10),*, pages 1–13, Hong Kong, May 2001.
3. F. Baader and U. Sattler. Tableau Algorithms for Description Logics. In R. Dyckhoff, editor, *Proc. of the Int'l Conf. on Automated Reasoning with Tableaux and Related Methods (Tableaux 2000)*, volume 1847, pages 1–18, St Andrews, Scotland, UK, 2000. Springer-Verlag.
4. S. Bechhofer, C. Goble, and I. Horrocks. DAML+OIL is not Enough. In *SWWS-1, Semantic Web working symposium*, Stanford (CA), July 29th-August 1st 2001.
5. C. Beeri and R. Ramakrishnan. On the power of magic. In *Proc. ACM SIGACTSIGMOD-SIGART Symposium on Principles of Database Systems*, pages 269–284, 1987.
6. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.
7. A. Borgida. Description logics are not just for the FLIGHTLESS-BIRDS: A new look at the utility and foundations of description logics. Technical Report DCS-TR-295, Department of Computer Science, Rutgers University, 1992.
8. E. Bozak, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, R. Studer, G. Stumme, Y. Sure, S. Staab, L. Stojanovic, N. Stojanovic, J. Tane, and V. Zacharias. KAON - Towards An Infrastructure for Semantics-based E-Services. In *Proceedings of the 3rd International Conference on Electronic Commerce and Web Technologies - EC-Web 2002*, Aix-En-Provence, France, 2002. Springer-Verlag.
9. D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema/.
10. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. In D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors, *Semantics for the WWW*.
11. R. G. G. Cattell, D. K. Barry, R. Catell, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.
12. W. Chen, M. Kifer, and D. S. Warren. HiLog: A Foundation for Higher-Order Logic Programming. In *Journal of Logic Programming*, volume 15, pages 187–230, 1993.
13. S. Cranefield and M. Purvis. UML as an ontology modelling language. 1999.

14. C. Davis, S. Jajodia, P. Ng, and Eds. R. Yeh. Entity-Relationship Approach to Software Engineering. In *Proceedings of the International Conference on Entity-Relationship Approach*, North-Holland, 1983.
15. F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt. Adding epistemic operators to concept languages. In *Proc. of the Third International Conference on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 342–353. Morgan Kaufmann, 1992.
16. A. Evans and A. Clark. Foundations of the unified modeling language. Springer-Verlag, 1998.
17. L. Fegaras. Query Unnesting in Object-Oriented Databases. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 49–60, Seattle, Washington, USA, June 1998. ACM Press.
18. D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a Nutshell. In *Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000)*, pages 1–16. Springer-Verlag, October 2000.
19. M. Fitting. *First-Order Logic and Automated Theorem Proving, 2nd Edition*. Springer Verlag, 1996.
20. M. Fowler and K. Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (2nd Edition)*. Addison-Wesley Pub Co., August 1999.
21. A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Understanding top-level ontological distinctions. In *Proceedings of IJCAI 2001 workshop on Ontologies and Information Sharing*, 2001.
22. B. Grossof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of WWW 2003*, Budapest, Hungary, May 2003.
23. P. Hayes. RDF Model Theory, http://www.w3.org/TR/rdf-mt/.
24. I. Horrocks. FaCT and iFaCT. In *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 133–135, 1999.
25. I. Horrocks and S. Tessaris. Querying the semantic web: a formal approach. In *Proc. of the 2002 International Semantic Web Conference (ISWC 2002)*, Sardinia, Italy, June 2002. Springer-Verlag.
26. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. In *11th International Conference on the WWW*, Hawaii, 2002.
27. D. B. Kemp, D. Srivastava, and P. J. Stuckey. Magic Sets and Bottom-Up Evaluation of Well-Founded Models. In Vijay Saraswat and Kazunori Ueda, editors, *Proceedings of the 1991 International Symposium on Logic Programming*, pages 337–354, San Diego, USA, 1991. The MIT Press.
28. M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42:741–843, July 1995.
29. C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition)*. Prentice Hall, July 2001.
30. O. Lassila and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification, http://www.w3.org/TR/REC-rdf-syntax/.
31. A. Maedche, B. Motik, N. Silva, and R. Volz. MAFRA — An Ontology MApping FRAmework in the Context of the Semantic Web. In *Workshop on Ontology Transformation at ECAI - 2002*, Lyon, France, July 2002.
32. A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. Managing Multiple Ontologies and Ontology Evolution in Ontologging. In *Proceedings of the Conference on Intelligent*

Information Processing, World Computer Congress 2002, Montreal, Canada, 2002. Kluwer Academic Publishers.
33. B. McBride. Jena: Implementing the RDF Model and Syntax Specification. http://www-uk.hpl.hp.com/people/bwm/papers/20001221-paper/.
34. D. L. McGuinness and J. R. Wright. An Industrial Strength Description Logic-based Configurator Platform. *IEEE Intelligent Systems*, 13(4):69–77, July/August 1998.
35. B. Motik, A. Maedche, and R. Volz. A Conceptual Modeling Approach for Semantics-driven Enterprise Applications. In *Proc. 1st Int'l Conf. on Ontologies, Databases and Application of Semantics (ODBASE-2002)*, October 2002.
36. N. F. Noy, R. W. Fergerson, and M. A. Musen. The knowledge model of Protege-2000: Combining interoperability and flexibility. In *2th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000)*, Juan-les-Pins, France, 2000.
37. J. Pan and I. Horrocks. Metamodeling architecture of web ontology languages. In *Proceedings of the Semantic Web Working Symposium*, pages 131–149, July 2001.
38. P. F. Patel-Schneider, P. Hayes, I. Horrocks, and F. van Harmelen. Web Ontology Language (OWL) Abstract Syntax and Semantics. http://www.w3.org/TR/owl-semantics/, November 2002.
39. G. Schreiber. Some challenge problems for the Web Ontology Language, http://www.cs.-man.ac.uk/ horrocks/OntoWeb/SIG/challenge-problems.pdf.
40. G. Schreiber. The Web is not well-formed. *IEEE Intelligent Systems*, pages 79–80, March/April 2002.
41. N. Stojanovic, L.Stojanovic, and R. Volz. A reverse engineering approach for migrating data-intensive web sites to the Semantic Web. In *Proceedings of the Conference on Intelligent Information Processing, World Computer Congress*, Montreal, Canada, 2002. Kluwer Academic Publishers.
42. R. Volz, B. Motik, and A. Maedche. Poster at 1st International Semantic Web Conference ISWC. Sardinia, Italy, June 2002.
43. R. Volz, D. Oberle, and R. Studer. Views for light-weight web ontologies. In *Proceedings of ACM Symposium of Applied Computing (SAC)*, Melbourne, Florida, USA, March 2003.
44. C. A. Welty and D. A. Ferrucci. What's in an instance? Technical report, RPI Computer Science, 1994.