A Conceptual Modeling Approach for Semantics-Driven Enterprise 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 automation of information processing. Although many approaches for representing and applying ontologies have already been devised, they haven't found their way into enterprise applications. In this paper we argue that ontology-based systems lack critical technical features, such as scalability, reliability, concurrency and integration with existing data sources, as well as the support for modularization and meta-concept modeling from the conceptual modeling perspective. We present a conceptual modeling approach that balances some of the trade-offs to more easily integrate into existing enterprise information infrastructure. Our approach is implemented within KAON, the Karlsruhe Ontology and Semantic Web tool suite.

1 Introduction

The application of ontologies[1] – shared conceptualizations of some domain – is increasingly seen as key to enable 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[2], where Web pages are annotated by ontology-based meta-data, the interest in ontology research increased, in hope of finding ways to off-load large-volume information processing from the human user to autonomous agents.

Many ontology languages have been developed, each aimed at solving particular aspects of conceptual modeling. Some of them, such as RDF(S) [20, 6], are simple languages offering elementary support for ontology modeling for the Semantic Web. There are other, more complex languages with roots in formal logic, focused around inference – ways to automatically infer facts not explicitly present in the model. For example, the F-logic language[18] (implemented, among others, in the OntoBroker system [10]) is based on an object-oriented extension of Prolog. On the other hand, various classes of description logic languages (e.g. OIL [12]) are mainly concerned with finding an appropriate subset of first-order logic with decidable and complete inference procedures (implemented, among others, in highly-optimized systems such as FaCT [16]). Despite

[1] An ontology is a conceptual model shared between autonomous agents in a specific domain.
[2] http://www.gca.org/attend/2000_conferences/XML_2000/knowledge.htm#lee

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. Relational databases have been developed over the past 20 years to a maturity level unparalleled with that of ontology-based systems, incorporating features critical for business applications, such as scalability, reliability and concurrency support.

Further, integration of existing information sources into ontology-based systems isn't easy. Databases cannot be replaced with ontology-based systems, because of the number of existing applications that depend on them. Hence, complicated and error-prone replication strategies are typically devised. Finally, ontology modularization is not provided in available systems. Because of these problems, up until today there hasn't been a large number of successful enterprise application of ontology technologies.

On the other hand, database technologies alone are not appropriate for handling information based on ontologies. This is mainly due to the fact that in databases systems first the conceptual model of the model is developed (e.g. using entity-relationship modeling [9]), but for actual implementation it is transformed into the 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, are not straightforward within the logical model.

Contribution of the paper. In this paper we present a conceptual modeling approach suitable for business-wide applications, designed based on the requirements analysis of several applications we are working on. We adjust the expressiveness of traditional logic-based languages to sustain tractability. As a side effect this makes realization of enterprise-wide ontology-based system using existing and well-established technologies, such as relational databases, possible. Other critical features are modularization and modeling meta-concepts with well-defined semantics. Our goal is to allow expressing and accessing conceptual models in a natural and easily understandable way, with a low gap between model conceptualization and its implementation in the system. We present the current status of the implementation of our approach within KAON[3] – Ontology and Semantic Web tool suite used as basis for our research and development. Finally, we discuss the possibilities of integration of ontology systems with existing information sources.

2 Requirements

In this section we discuss the requirements gathered during our work on various projects.

Unambiguous Semantics. The primary motivation for using ontologies over other modeling approaches is to enrich the information with semantics. For example, the notion that a class should be understood as a set of instances is defined by the semantics of an ontology modeling language.

[3] http://kaon.semanticweb.org/

The absence of clear a clear semantics may led to diverging interpretations of intended meaning, for example, with RDFS [13, 15, 7, 23] and UML [5]. Thus, a clear semantic description of an ontology modeling language is an important requirement.

Object-oriented Paradigm. In the last decade the object-oriented paradigm has become prevalent for conceptual modeling. A wide adoption of UML [14, 11] as syntax for object-oriented models has further increased its acceptance.

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 them easily understandable. Thus, any successful conceptual modeling approach should incorporate the object-oriented paradigm.

Meta-concepts. In real-world conceptual models, it is often unclear whether some element should be represented as a concept or as an instance. An excellent example in [25] 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, apes and particular apes. Most natural conceptualization is to say that there is a concept SPECIES (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 properties. 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 [21], 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 ontology [22]. Evolution (meta-)ontology consists of concepts reflecting various types of changes in an ontology.

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 conceptual models 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 include the CONTINENT concept – one must include its instances as well to be able to talk about particular continents, such as Europe. The shared nature of ontology implies reuse. We consider modularization – on both ontologies and instances – to be an important aspect of reuse.

Lexical Information. Many applications, such as semantics-driven web content management, extensively depend on lexical information about entities in an ontology, such as labels for ontology entities in different languages. Hence, consistent way of associating lexical information with ontology entities is mandatory.

Root Concept. Although not a semantically critical issue, from the methodological point of view, the necessity to explicitly classify a concept into the hierarchy forces the modelers to think about the proper characterization of the concept in the hierarchy. Further, including a root concept from which all other concepts are derived results in more elegant and manageable conceptual models. For example, a property is made applicable to instances of any concept by specifying the root concept as the property domain. If there were no root concept, the case with no domain should be treated specially, thus resulting in a slightly more complicated system. Many systems – e.g. all description logic systems or Protege-2000 – include some form of a root concept.

Light-weight Inferences. 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.

Based on our experience, we observed that while rules in conceptual modeling are important, they are often used in some well-defined patterns. Hence, instead of building a system based on a general reasoning paradigm, we introduce the notion of light-weight inferences based on axiom patterns [26] – predefined types of rules that sustain scalability and tractability. The list of axiom patterns is currently limited to common patterns in ontology structure, such as symmetric, transitive and inverse relations.

Technical Issues . In order to be applicable for real-world enterprise applications, our conceptual modeling approach must make it easy fulfill technical requirements such as scalability, concurrency support, reliability and easy integration with existing data sources.

3 Conceptual Modeling Approach

In this section we present the mathematical definition of our modeling language. Next the denotational semantics is presented in standard Tarski style. Finally, several examples are presented.

3.1 Mathematical Definition

We present our approach on an abstract, mathematical level that defines the structure of our models[4]. We may support this structure with several different syntaxes. Further, in enterprise systems the structure of model is much more important, since this structure must be reflected in the systems for information storage and retrieval.

[4] The definition is based on previous work as described in [4]

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 IO-models,*
- *INC is the set of included OI-models.*

An OI-model represents a self-contained unit of structured information that may be reused. Elements in an OI-model are entities. An OI-model 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 associated with an OI-model is a 10-tuple* $O(OIM) := (C, P, S, T, INV, H_C, H_P, domain, range, mincard, maxcard)$ *where:*

- $C \subseteq E$ *is a set of concepts,*
- $P \subseteq E$ *is a set of properties,*
- $S \subseteq P$ *is a subset of symmetric properties,*
- $T \subseteq P$ *is a subset of transitive properties,*
- $INV \subseteq P \times P$ *is a symmetric relation that relates inverse properties, if* $(p_1, p_2) \in INV$*, then* p_1 *is an inverse 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*,* 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*,* p_2 *is a superproperty of* p_1*,*
- *function domain* $: P \rightarrow 2^C \setminus \{\emptyset\}$ *gives the set of domain concepts for some property* $p \in P$*,*
- *function range* $: P \rightarrow (2^C \setminus \{\emptyset\}) \cup \{ L \}$ *gives the set of range concepts for some property* $p \in P$*,*
- *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, consisting of a set definitions regulating how instances should be constructed. An ontology consists of concepts (sets of elements) and properties (specification how objects may be connected). Each property must have at least one domain concept, while its range may either be a literal, or a set of at least one concept. Domain and range concept restrictions are treated conjunctively – all of them must be fulfilled for each property instantiation. Some properties may be marked as transitive, and it is possible to say that two 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 may be specified for instances of that class. Concepts and 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 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 $: C \to 2^I$ relates a concept with a set of its instances,*
- *partial function instprop $: P \times I \to 2^{I \cup L}$ assigns to each property-instance pair a set of instances related through given property.*

Each IO-model has an instance pool associated with it. 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.

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

Each other OI-model must include ROIM and thus gain visibility to the root concept. This is similar to object-oriented languages approaches – 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 are marked with subscript 1), that is, if OIM$_1$ $\in$ INC(OIM) must satisfy following modularization constraints:*

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

If an OI-model imports some other OI-model, it contains all information – no information may be lost. Modularization constraints just specify structural consequences of importing an OI-model. This is independent from the implementation – imported OI-models may be physically duplicated, whereas in other cases they may be linked.

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.

Definition 7 (Lexical OI-model Structure). *Lexical OI-model structure LOIM is a well-known OI-model with the structure matching that presented in the Figure 1*[5].

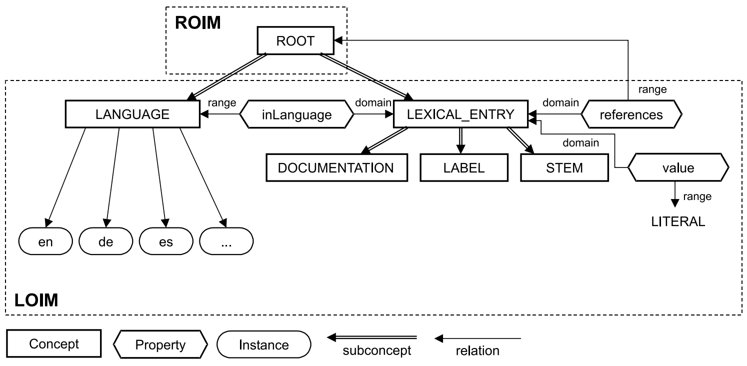

Fig. 1. Lexical OI-Model Structure

Lexical entries (instances of the LEXICAL_ENTRY 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 property $references$. 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 $value$, whereas the language of the value is specified through the $inLanguage$ property. Concept LANGUAGE represents the set of all languages, and its instances are defined by the ISO standard 639.

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

3.2 Denotational Semantics

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

- *$\triangle^I$ is the domain set,*
- *$C^I : C \rightarrow 2^{\triangle^I}$ is a concept interpretation function that maps each concept to a subset of the domain set,*
- *$L^I : L \rightarrow \triangle^I$ is a literal interpretation function that maps each literal to a single element of a domain,*
- *$I^I : I \rightarrow \triangle^I$ is an instance interpretation function that maps each instance to a single element in a domain,*
- *$P^I : P \rightarrow 2^{\triangle^I \times \triangle^I}$ is a property interpretation function that maps each property into a relation over the domain set.*

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

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

- $\forall c \in C, i \in I \;\; i \in instconc(c) \Rightarrow I^I(i) \in C^I(c)$,
- $\forall c_1, c_2 \in C \;\; (c_1, c_2) \in H_C \Rightarrow C^I(c_1) \subseteq C^I(c_2)$,
- $\forall c \in C \;\; C^I(c) \subseteq C^I(ROOT)$,
- $\forall p_1, p_2 \in P \;\; (p_1, p_2) \in H_P \Rightarrow P^I(p_1) \subseteq P^I(p_2) \wedge$
  $\bigcup_{c_1 \in domain(p_1)} C^I(c_1) \subseteq \bigcup_{c_2 \in domain(p_2)} C^I(c_2) \wedge$
  $\bigcup_{c_1 \in range(p_1)} C^I(c_1) \subseteq \bigcup_{c_2 \in range(p_2)} C^I(c_2)$,
- $\forall p \in P, i \in I \;\; instprop(p,i)$ *is defined* $\Rightarrow \forall c \in domain(p) \;\; I^I(i) \in C^I(c)$,
- $\forall p \in P, i_1, i_2 \in I \;\; i_2 \in instprop(p, i_1) \Rightarrow range(p) \neq L \wedge$
  $(I^I(i_1), I^I(i_2)) \in P^I(p) \wedge \forall c \in range(p) \;\; I^I(i_2) \in C^I(c)$,
- $\forall p \in P, i \in I, l \in L \;\; l \in instprop(p, i) \Rightarrow range(p) = L \wedge$
  $(I^I(i), L^I(l)) \in P^I(p)$,
- $\forall p \in P, c \in C \;\; mincard(c,p) \leq |\{\, y \mid (x,y) \in P^I(p) \wedge x \in C^I(c)\}| \leq maxcard(c,p)$,
- $\forall s \in S \;\; P^I(s)$ *is a symmetric relation,*
- $\forall p, ip \in P \;\; (p, ip) \in INV \Rightarrow P^I(ip)$ *is an inverse relation of* $P^I(p)$,
- $\forall t \in T \;\; P^I(t)$ *is a transitive relation.*

OIM is unsatisfiable it is doesn't have a model. Following information can be inferred from OIM, since it is true in all models of OIM:

- $H_C^* \subseteq C \times C$ *is the transitive closure of the concept hierarchy and is defined by* $C^I(c_1) \subseteq C^I(c_2) \Leftrightarrow (c_1, c_2) \in H_C^*(c_1, c_2)$,
- $H_P^* \subseteq P \times P$ *is the transitive closure of the property hierarchy and is defined by* $P^I(p_1) \subseteq P^I(p_2) \Leftrightarrow (p_1, p_2) \in H_P^*(p_1, p_2)$,
- $instprop^* : P \times I \rightarrow 2^{I \cup L}$ *is a partial function assigning to each property-instance pair a set of related instances or literals and is defined by*
  $i_2 \in instprop^*(p, i_1) \Leftrightarrow (I^I(i_1), I^I(i_2)) \in P^I(p)$ *and*
  $l \in instprop^*(p, i) \Leftrightarrow (I^I(i), L^I(l)) \in P^I(p)$.

This definition of semantics has the following interesting consequences:

- H_C, H_P and instprop represent explicit (ground) information in an OI-model. From this ground information and an interpretation, it is possible to infer H_C^*, H_P^* and instprop*, thus supporting the light-weight inference requirement.
- A subproperty of some property may add additional domain and range restrictions, but cannot remove existing ones.
- An interpretation is not associated with entities, but with concepts, instances and properties. As stated by the definition 6, it is possible for a single entity to be a concept and an instance at the same time. However, such an entity has two different interpretations – once an interpretation as a set (through function C^I), and once as an instance (through function I^I). These interpretations map concept resp. instance to completely different domain objects. The notion of spanning object unifies these different interpretations.

Definition 9 (Spanning Object). *Under interpretation I, for each entity $e \in E$ the spanning object is defined as a triple $SO(e) := (C^I(e), P^I(e), I^I(e))$ that combines different interpretations of the entity e.*

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 spanning object.

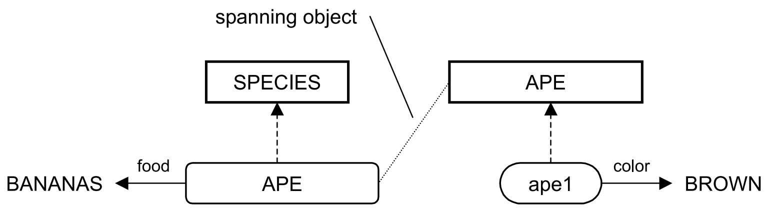

Fig. 2. Spanning Object Example

In [28] the problems of considering concepts as instances are well explained. A solution proposed in the paper is to isolate different domains of discourse. What is an 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 in my model. 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. Before interpreting a model, 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.

This approach is similar to the approach from F-Logic. In F-Logic it is possible to use the same symbol to denote the concept and an instance. Two occurrences are mutually independent, unless an explicit rule is created.

In [24] RDFS has been criticized for its infinite meta-modeling architecture that may under some circumstances cause Russell's paradox. A fixed four-layer meta-modeling architecture called RDFS(FA) has been proposed that introduces a strict separation between concepts and instances. Concepts are part of the ontology layer, whereas instances are part of the instance layer.

In our proposal we are less restrictive, as each entity may be assigned several different interpretations, but the modeling architecture is fixed to three layers (meta-model, ontology, instances). Thus, with the added flexibility in creation of models similar to that of RDFS, the Russell's paradox cannot happen.

3.3 Example

In this section we present an example OI-model. A common problem in knowledge management systems is to model documents classified into various topic hierarchies. We base the conceptualization of the domain on the DOCUMENT concept whose instances represent individual documents. Topics are modeled as instances of the TOPIC concept and the subtopic transitive property specifies that a topic is a subtopic of another topic. The has-author property between a document and an author determines the author of a document, and the has-written property is inverse to it. The has-topic property between a document and a topic determines the topic of the document. Individual persons are members of the PERSON concept. Two roles – RESEARCHER and AUTHOR – are subconcepts of the PERSON concept.

The hierarchy of topics is self-contained unit of information, so to allow its reuse, it is factored out into a separate OI-model called TOPICS. Similarly the personnel information is separated into OI-model called HR, and the document information is separated into OI-model called DOCUMENT. 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.

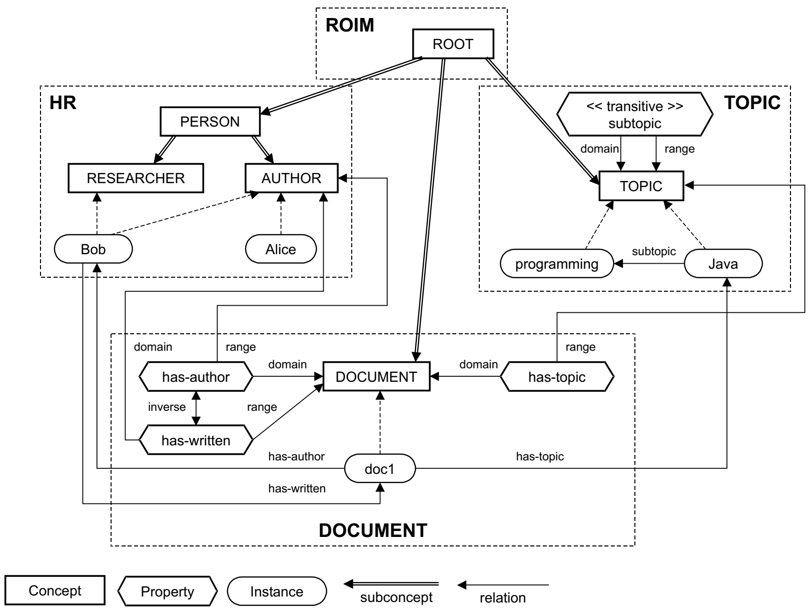

Fig. 3. Example Domain Ontology

Figure 3 shows the information graphically. In this figure the boxes represent concepts, rounded boxes represent instances, and hexagonal boxes represent property definitions. Please note that property instances are members of the OI-model that contains their label (e.g. has-author between doc1 and Bob is member of the DOCUMENT OI-model). Transitivity of the subtopic property has been specified by the ≪ transitive ≫ stereotype.

4 Implementation

In this section we present how our conceptual modeling approach is applied to KAON – a platform for developing and deploying semantics-driven enterprise applications.

4.1 KAON API within KAON Architecture

Our conceptual modeling approach defines the data model of KAON. The manipulation of the data model is performed through 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 4.

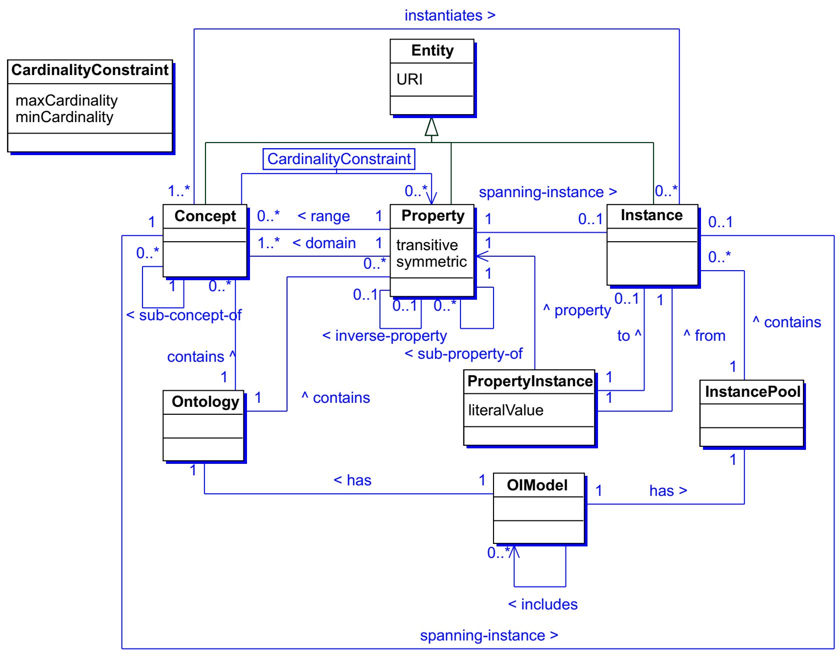

Fig. 4. UML View of KAON API

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.

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:

Implementation optimized for ontology engineering. A separate implementation of KAON API 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 next subsection.

Implementation for RDF repository access. An implementation of KAON API based on RDF API[6] 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 RDF API implementation exists. KAON RDF Server is such a repository that enables persistence and management of RDF models.

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 [27]. 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 OI-model is generated. When accessed, the model will translate the request into native database queries, thus fulfilling most requests directly within the database itself. Similarly, the OI-model will will translate ontology update requests to a series of updates to the underlying database. In such way the persistence of ontology information is obtained, while reusing well-known database mechanisms such as transactional integrity.

4.2 Database Schemas for Storing Conceptual Models

In previous section we discussed how KAON API isolates applications from actual data sources by offering various API implementations. In this section we describe possible database organizations for these API implementations.

[6] We adapted and reengineered Sergey Melnik's RDF API for our purposes, see http://www-db.stanford.edu/ melnik/rdf/api.html.

Generic Schema for Ontology Engineering. Ontology engineering is a cooperative consensus building process, during which several ontology modelers experiment with different modeling possibilities. As a result, concept adding, 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 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 5, is used. The schema is a straightforward translation of the definitions 1, 2 and 3 into relational model.

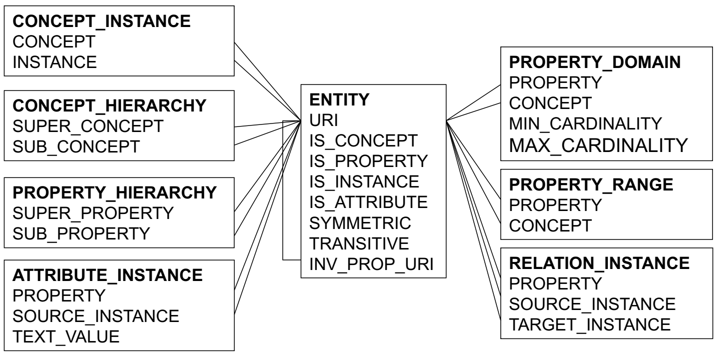

Fig. 5. Generic KAON Schema

The schema is organized around the ENTITY 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) [17] in one significant point. In the RQL implementation, to each concept an unary table is assigned whose rows identify the instances of some concept. 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 are 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 critical.

Schema Optimized for Instance Storage. While it is possible to use the generic schema for run-time ontology storage and access as well, typically this will result in suboptimal performance. This is especially true for accessing instance data – joins are required for accessing a value of an instance property. After ontology engineering is finished and if it may be assumed that the ontology will not to change significantly, an optimized schema for storing concept extensions may be automatically generated. Such schema can then be accessed using the KAON API implementation for accessing any database.

The process of optimized schema generation is inspired by well-known approaches for mapping objects to relational databases (e.g. [1]), and may be performed through following steps:

- For each concept c, create a table c_{EXT} for the extension of c and include a primary key equal to the instance URI.
- For each property p and $c \in$ domain(p), if maxcard$(c, p) \leq t$, where t is some predefined threshold, add attributes $p_1, ..., p_{maxcard}$ to c_{EXT} for storing values of the property. Attributes $p_1, ..., p_{mincard}$ mark as not nullable, others mark as nullable. If range$(p) \neq LITERAL$, create a foreign key constraint from each p_i to t_{EXT}, for each $t \in$ range(p).
- If maxcard$(c, p) > t$, where t is some predefined threshold, create a table p_{EXT} for the extension of p. Create a foreign key constraint from SOURCE_URI to s_{EXT} for each $s \in$ domain(p). If range$(p) = LITERAL$, include an attribute VALUE, otherwise an attribute TARGET_URI with a foreign key constraint to t_{EXT} for each $t \in$ range(p).
- If concept c is a subconcept of a, then create a foreign key constraint on URI from c_{EXT} to a_{EXT}.

Figure 6 shows the schema obtained through by applying the algorithm to example in Figure 3. For illustration purposes we assume that a single document can have at most one topic.

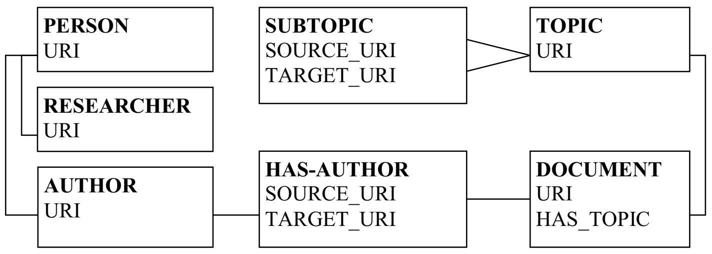

Fig. 6. Optimized KAON Schema

5 Related Work

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

Entity-Relationship and Relational Modeling. Entity-relationship models are a tool for conceptual design of databases. Before implementation ER 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 perspective must be kept in synchrony.

In our approach we isolate the mapping of the conceptual model to the logical one into a separate step, thus making the logical model hidden from the ontology user. The users of the ontology should use ontology in an ontology-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. [24]) 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 with elements of the ontology are possible.

UML. There have been proposals for using UML as an ontology modeling language (e.g. [8]). 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 [19]). 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. The class of an object is determined at the point when object is created.

In our conceptual 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 are true for that object.

Frame-based Languages. The ideas of object-oriented modeling paradigm on ontology modeling has resulted in creation of so called frame-based knowledge modeling languages, and F-logic [18] is one example. In F-logic, the knowledge is structured into frames (analogous to classes) that have different value slots (analogous to attributes). 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 not explicitly present information.

Although F-logic is an expressive modeling language, the application of axioms makes F-logic often intractable and therefore unpractical for many of the aforementioned applications. Implementing inference procedures for Horn logic requires special data structures that are incompatible with data structures typically found in relational databases. In comparison, our conceptual model approach introduces the notion of lightweight inferences, which are intended to be easily implementable with existing systems.

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 [3]). Although description logics are founded on a well-research theory, as mentioned in [2], they have proven to be difficult to use due to the non-intuitive style of modeling. 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[7]. 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 is exists, but is not intuitive – we do not specify exactly with which instance of t1 is d1 related, but to say that it is related to "some" instance of t1. In the syntax of CLASSIC[8] description logic system, this is expressed as

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

OIL. Another important ontology modeling language is OIL [12]. This language combines the intuitive notions of frames, clear semantics of description logics and the serialization syntax of RDF embedded within a layered architecture[9]. 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. Although it supports ontology modularization, it doesn't have a consistent strategy for management of lexical information.

6 Conclusion

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

[7] Thanks to Ian Horrocks for discussion on this topic.
[8] http://www.bell-labs.com/project/classic/
[9] http://oil.semanticweb.org/

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, concurrency and support for integration with existing data sources. 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. Finally, we have presented the current status of the implementation in the form of the KAON API – an API for management of ontologies and instances. Finally, we have shown how our conceptual modeling approach fulfills the requirements for integration with existing data sources.

In future, a query language for conceptual models is needed, perhaps based on paradigms found in description logics. Next, we want to investigate how to extend the set of axiom patterns (e.g. by allowing some form of free Horn-logic rules), by keeping required performance levels. Finally, we want to test the adequacy of the conceptual modeling approach and its implementation by testing its suitability and performance in our target applications.

References

1. S. W. Ambler. Mapping objects to relational databases: What you need to know and why. http://www-106.ibm.com/developerworks/library/mapping-to-rdb/, July 2002.
2. 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.
3. 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.
4. 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.
5. R. Breu, U. Hinkel, C. Hofmann, C. Klein, B. Paech, B. Rumpe, and V. Thurner. Towards a Formalization of the Unified Modeling Language. In *Proceedings of ECOOP'97 - Object-Oriented Programming, 11th European Conference*, LNCS 1241, Finland, June 1997. Springer Verlag.
6. D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema/.
7. W. Conen and R. Klapsing. A Logical Interpretation of RDF. In *Journal of Electronic Transactions on Artificial Intelligence (ETAI), Area: The Semantic Web (SEWEB)*, volume 5, 2000.
8. S. Cranefield and M. Purvis. UML as an ontology modelling language. In *Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.
9. 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.

10. S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In *Database Semantics - Semantic Issues in Multimedia Systems, Proceedings TC2/WG 2.6 8th Working Conference on Database Semantics (DS-8)*, Rotorua, New Zealand, January 1999.
11. A. Evans and A. Clark. Foundations of the unified modeling language. In *In 2nd Northern Formal Methods Workshop, Ilkley, electronic Workshops in Computing*. Springer-Verlag, 1998.
12. 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.
13. R. Fikes and D. McGuinness. An Axiomatic Semantics for RDF, RDF-S, and DAML+OIL (March 2001), http://www.w3.org/TR/daml+oil-axioms.
14. M. Fowler and K. Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (2nd Edition)*. Addison-Wesley Pub Co., August 1999.
15. P. Hayes. RDF Model Theory, http://www.w3.org/TR/rdf-mt/.
16. I. Horrocks. FaCT and iFaCT. In *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 133–135, 1999.
17. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. In *Proceedings of The Eleventh International World Wide Web Conference (WWW'02)*, Hawaii, May 2002.
18. 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.
19. C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition)*. Prentice Hall, July 2001.
20. O. Lassila and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification, http://www.w3.org/TR/REC-rdf-syntax/.
21. 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.
22. 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.
23. W. Nejdl, H. Dhraief, and M. Wolpers. O-Telos-RDF: A Resource Description Format with Enhanced Meta-Modeling Functionalities based on O-Telos. In *Workshop on Knowledge Markup and Semantic Annotation at the First International Conference on Knowledge Capture (K-CAP'2001)*, Victoria, B.C., Canada, October 2001.
24. J. Pan and I. Horrocks. Metamodeling architecture of web ontology languages. In *Proceedings of the Semantic Web Working Symposium*, pages 131–149, July 2001.
25. G. Schreiber. Some challenge problems for the Web Ontology Language, http://www.cs.man.ac.uk/ horrocks/OntoWeb/SIG/challenge-problems.pdf.
26. S. Staab, M. Erdmann, and A. Maedche. Engineering Ontologies using Semantic Patterns. In *Proceedings of the IJCAI-2001 Workshop on E-Business & Intelligent Web*, Seattle, August 2001.
27. 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.
28. C. A. Welty and D. A. Ferrucci. What's in an instance? Technical report, RPI Computer Science, 1994.