View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Infoscience - École polytechnique fédérale de Lausanne

Ontologies : A contribution to the DL/DB debate.

Nadine Cullot[1], Christine Parent[3], Stefano Spaccapietra[2], and Christelle Vangenot[2]

[1] LE2I Laboratory, University of Burgundy,
BP 47870, 21078 Dijon Cedex, France
nadine.cullot@u-bourgogne.fr,
[2] Database Laboratory, Swiss Federal Institute of Technology,
CH-1015 Lausanne, Switzerland
stefano.spaccapietra@epfl.ch,christelle.vangenot@epfl.ch
[3] University of Lausanne, CH-1015 Lausanne, Switzerland
christine.parent@unil.ch

Abstract. The move to global economy has emphasized the need for intelligent information sharing, and turned ontologies into a kernel issue for the next generation of semantic information services. The push towards an effective use of ontologies as a means to achieve semantic interoperability is, in our opinion, shifting the focus from purely taxonomic ontologies to more descriptive ontologies. These would namely provide agreed descriptions of the data structures representing the complex organization of objects and links of interest within the targeted domain. This paper analyzes the requirements for such descriptive ontologies, and contrasts the requirements to the functionality provided by some current representative approaches that have been proposed for ontology management. Selected approaches originate from research in artificial intelligence, knowledge representation and database conceptual modeling. The paper concludes that extending rich semantic data models with support for reasoning is an interesting alternative to extending description logics with data management functionality.

1 Introduction

Information sharing, rather than information processing, is what characterizes information technology in the 21st century. Consequently, ontologies gain increasing attention, as they appear as the most promising solution to enable information sharing both at a semantic level and in a machine-processable way. Ontologies by definition provide an encoded representation of a shared understanding of terms and concepts in a given domain, as agreed by a community of people. But ontologies are not all alike. At least three orthogonal criteria may be used to differentiate among them.

Ontology Focus. Wordnet (http://www.cogsci.princeton.edu/ wn/) is the most well-known representative of first-generation ontologies that basically provide

definitions of terms and are intended to be used as sophisticated thesauri. Their structure shows terms organized into a subsumption hierarchy (each term conveying the definition of a more specialized concept than its parent term), and linked by other relationships to express synonymy, composition, etc.... This kind of ontology, usually referred to as taxonomic ontologies, is useful in information-sharing infrastructures to provide a reference vocabulary for aligning names denoting data in different data sets. Other ontologies reach beyond terminology, defining conceptualizations that include the representation of properties of concepts and their interrelationships. These "descriptive" ontologies resemble database schemas, showing concepts interconnected by a variety of semantic associations to achieve a semantically rich representation of the intended domain. An example (out of the many existing ones) is ImMunoGeneTics, an international medical ontology (http://imgt.cines.fr). These ontologies are useful in information sharing to align existing data structures (not just terms) into an integrated description of the corresponding domain, or, in a top-down perspective, to provide patterns for the definition of new, specialized ontologies or database schemas.

Ontology Scope. The term scope here refers to the intended use of the ontology. Ontologies may be designed and used for purely explanatory purposes, i.e., as a service to enable the understanding of some domain. Such ontologies usually come without associated instances, as these are at a level of detail whose representation is most often not relevant. Ontologies may also serve as a means to actually support some data management services. Such ontologies have associated instances, stored in either a database or a semi-structured data set in e.g. a web server. In the latter case, the ontology plays the role of a database schema in guiding access to the data in the web server. In the former case, the ontology may either be used just to assist in the design of the database schema by providing background semantic information, or as an operational component in the information management architecture, providing access functionality that is additional to those typically provided by a DBMS. A characteristic example of additional functionality is the management of incomplete data. The split between explanation (non instantiated) and management (instantiated) ontologies is orthogonal to the split between taxonomic and descriptive ontologies. A taxonomic ontology serves a data management goal if it contains, for instance, references to databases where the user may find data corresponding to a given term or concept, thus facilitating information retrieval. Descriptive ontologies defined by some standardization body for a given application domain are explanatory. They hold generic abstractions of a domain, not aiming at managing a specific database. However, descriptive ontologies, looking very much alike a database schema, are natural candidates for being used for data management.

Ontology Context. Ontologies traditionally convey a single, monolithic conceptualization, supposedly stating the truth, i.e., how the described domain should be understood. But many different conceptualizations may exist for the same real word, each one defining a view shared by some user community. Contextual ontologies have been proposed to support alternative views of the world.

They provide definitions and descriptions that are context-dependent, thus supporting use from inhomogeneous user communities. The advantage of a contextual ontology, versus the alternative to have independent ontologies (or a hierarchy of ontologies), is that in a contextual ontology it is very easy to support navigation among contexts, i.e. dynamically moving from one context to another. Another advantage is to have a multi-context vision of the world available as a single consistent whole. This is particularly important when updating ontologies is considered and update propagation from one context to another is desirable.

While the traditional use of ontologies is more on the taxonomic and explanatory side, there is a tangible push to move towards descriptive, and even management-oriented ontologies, in order to use them as a key component in data integration frameworks. In such frameworks ontologies do not refer anymore to arbitrary, abstract perceptions of the real world. They describe some defined subset of the real world that is actually represented in the stored data. The increase in similarity between ontologies and database schemas [1] arises a legitimate question about the appropriateness and effectiveness of using a conceptual modeling approach (that has proven to be the best to elaborate a semantically rich description of the data in a database) to describe the conceptualization that is the subject of an ontology. In other terms, could the conceptual modeling know-how provide an interesting alternative to traditional description logic (DL) based approaches? This debate is still open. Some strong proposals from the database [2] and knowledge representation communities [3] already challenge the corpus of work more directly framed in the context of description logic and its reasoning capabilities (DAML+OIL [4]). The purpose of this paper is to contribute to the debate a detailed analysis of requirements for the description and use of what we believe will become the leading ontology framework, i.e. descriptive and, in the longer term, management ontologies. The paper also analyzes differences and complementarities between proposed approaches to ontology description and use. We conclude that enhancing conceptual models with reasoning support may be the best way to make ontologies operational in data integration frameworks.

The next section briefly introduces selected related work and representative proposals for ontology management. It also introduces the conceptual model we propose. Its facilities are illustrated using a simplified example (borrowed from [2]) of a scientific conference ontology. The same example is used throughout the paper. Section 3 discusses ontology requirements, focusing on differences with traditional database requirements. The highlighted issues are detailed in the sequel of the paper. Section 4 focuses on the comparison of the data models. Section 5 deals with how instances are handled. Section 6 analyzes constraint specification and consistency checking. Query languages are discussed in section 7. Some additional features for ontologies are described in section 8, before the concluding remarks.

2 Related work

Interactions between the ontology and conceptual modeling domains is a topic of rapidly growing interest for both communities, as witnessed by looking, for instance, at the number of ontology-oriented contributions at the last ER conference on Conceptual Modeling. The present workshop on Semantic Web and Databases is another clear sign of interest into these interrelationships.

A well-worth-reading starting point when targeting a comparison of models for ontologies is the paper by D. McGuinness [5], which provides interesting insights into a historical perspective of description logic developments, explaining how emerging applications such as those on the web have motivated the use of description logics.

A. Borgida and R.J. Brachman [6] adopt a conceptual modeling perspective to discuss ontology modeling issues, looking in particular at object-based aspects and DL. They highlight some weaknesses of DL in representing structured values, some kinds of constraints, and some forms of "inheritance" (for materialization) and meta-information for conceptual modeling. Conversely, they identify strengths of DL as its specific features to specify primitive and defined concepts, necessary and sufficient conditions, and its reasoning tools. We pursue the same comparative analysis by addressing more specifically ontology requirements and complementing the comparison of the database conceptual models and description logic approaches for modeling with two other issues: Instances handling and querying.

The comparative analysis of Entity Relationship (ER) models and DL proposed in [7] develops the transformation of ER schemas into knowledge bases. The chosen description logic, $\mathcal{DLR}$ [8], is a generalization of description logic for n-ary relations. The authors argue that the semantics of ER models can be captured by $\mathcal{DLR}$. They also address querying issues, showing that DL queries can only return subsets of existing objects, while database queries may also create new objects. Desirable extensions of standard DL queries are discussed. Our work also focuses on the comparison of conceptual models and DL. We complement their analysis by adopting the reverse viewpoint: We aim at identifying desirable extensions of conceptual models to better address ontological issues. We actually follow the path lead by R. Meersman [1], who argues that methods and techniques originally developed for database conceptual modeling and large databases management could be relevant for ontologies.

His DOGMA project is one of those we explicitly discuss in this paper. Indeed, to substantiate our analysis on trends in ontology management, we have looked in detail at proposals that we felt were good representatives of the alternative approaches from the research communities in artificial intelligence, knowledge representation, and databases.

Artificial Intelligence Approach and Reasoning. Description logics and their associated inference techniques have been extensively used as formal theories on which several ontology languages have been defined. We have chosen RACER [9] to represent this research area, because it has a wide range of applicability as it includes instance management facilities. RACER is a description logic rea-

soning system based on the $\mathcal{SHIQ}$ logic [10], [11]. RACER separates the formal description of the ontology schema (denoted as the TBox) from the description of individuals (in the ABox). RACER modeling constructs include:

- *Concepts.* They are atomic types defined by their names. Logical expressions may be attached to them, thus allowing designers to define [4]: a) subsumption hierarchies, e.g. *(implies Author Person)*, b) constraints associated to a concept, e.g. *(implies Author (at-least 1 Writes))*, c) stand-alone constraints, e.g. *(disjoint Person Committee Review Paper Topic)*, and d) a new concept, using a logical assertion, e.g. *(equivalent PCAuthor (and Author Reviewer))*. Concepts defined by a logical assertion are called "defined concepts", as opposed to the other ones called "primitive concepts".
- *Roles.* They define binary relationships between a domain concept and a range concept, e.g., *roles (Writes: domain Author :range Paper)*. Roles played by a concept can be qualified using quantified restrictions *(some, all)* and numeric restrictions *(at-most, at-least, exactly)*. Roles may be transitive, symmetric, and functional. They can have an inverse as well as super-roles.
- *Domain of values.* Integer and real named domains of values can be defined. But attributes, i.e. binary links from a concept to a domain of values, cannot be defined within the schema. Attribute values are dynamically defined and associated to instances of concepts. Lastly, RACER allows users to define in an intentional way, i.e. by a logical expression, characteristics of instances. For example, one can state that paper p100 has been written by exactly one author: *(instance p100 (and Paper (exactly 1 WrittenBy)))*.

Artificial Intelligence Approach and Knowledge Representation. We chose KAON [3] as a representative proposal transferring a knowledge representation know-how into the ontology domain. KAON is an ontology and semantic web framework allowing the design and management of ontologies. It includes an ontology modeling language based on RDF(S) with some proprietary extension and a conceptual query language. KAON supports modularization through the recursive definition of sub-models. Each sub-model has (similarly to RACER) two components:

- An ontology structure, holding definitions of concepts, oriented binary relationships between concepts, and attributes. Relationships may be symmetric, transitive and have an inverse. Minimum and maximum cardinality constraints for relationships and attributes may be specified. Concepts and relationships can be arranged in two distinct generalization hierarchies.
- An instance pool, holding concepts and relationship instances and attribute values. Specific to KAON is the possibility to have spanning objects, i.e. a real world entity being represented both as a concept and as an instance.

Database Approach. DOGMA [2] is an ontology engineering framework based on the ORM (Object-Role-Modeling) conceptual model[12]. ORM is a binary relationship data model. DOGMA splits the ontology into two parts:

[4] Examples refer to the Conference ontology illustrated in Figures 2 and 3 in RACER (and in Figure 1 in MADS).

- The ontology base, holding the data structure. Its definitions may be contextualized using a context name.
- A set of ontological commitments. A commitment is a set of integrity constraints (e.g., definition of identifiers, cardinalities) that govern the ontology for its use in a specific application. The idea is that the ontology base holds generic knowledge about a domain, while its association to a commitment set specializes the ontology for a given application within the domain.

Our approach to ontology modeling also belongs to the database inspired track. While DOGMA (as description logics and KAON) organizes the world as a collection of object tokens associated to properties and interrelated by binary relationships, we favor a more synthetic view, as supported by complex object data models (e.g., UML, extended ER models, semantic models). MADS [13] is such a data modeling framework. MADS is a spatio-temporal conceptual model that handles complex objects (i.e., objects with a multi-level attribute structure, where an attribute can be composed of other attributes), n-ary relationships with attributes, generalization hierarchies, multi-instantiation, as well as spatial, temporal and contextual features (context is materialized by stamping definitions, values and instances to express for which context they are relevant [14]. Both object and relationship types are first class constructs. MADS has associated data manipulation languages. The MADS framework includes a visual schema editor, a visual query editor and the associated mappings onto existing DBMS. It provides users with an integrated environment where they can work at the conceptual level for both designing and querying the database. Figure 1 uses traditional ER diagrammatic techniques to show a MADS data structure (without space, time, and contextual features) for our running example about activities and contributors of a scientific conference.

3 Ontology Requirements

This section holds introductory discussions of the four major components of an ontology management approach: How the conceptualization is described, how associated instances are managed, how reasoning is performed, and how data is queried. The discussion points at similarities and differences between ontology requirements and requirements for traditional databases. Sections 4 to 7 look in more detail into each issue.

Data Modeling. As we believe future ontologies will be descriptive rather than purely taxononmic, we assume the conceptualization includes the definition of relevant data structures. For instance, Figure 1 can be interpreted as illustrating an ontology data structure for management of conference reviews. Representing the knowledge that "papers are assigned to reviewers" as a data structure showing a relationship type linking the two complex object types defining papers and reviewers, is semantically richer than embedding the same semantics in the separate definition of three terms (paper, reviewer, assignment) in a taxonomic ontology. On the contrary, binary data models à la DOGMA, KAON, and description logics may provide a good solution for taxonomic ontologies: Concepts

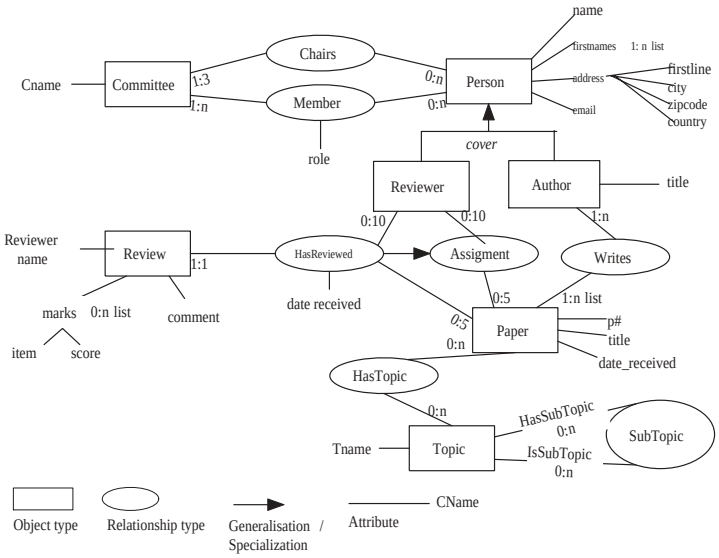

Fig. 1. MADS schema for the scientific conference ontology.

may represent terms and binary links may represent the classic taxonomic relationships, e.g., synonym, homonym, hyponym, holonym. Defining a semantic data structure is also an extremely efficient support for visualizing in an intelligent and intuitive way how the domain of interest is articulated into its many concepts. The capability to visualize the structure of a domain has always been one of the best selling arguments for conceptual models and their acceptance by users. It also has been concerns of knowledge representation systems [15]. Of course, this argument is irrelevant if the ontology is automatically built using some emergent semantic technique, or if the ontology is explored by agents only.

Aiming at expressiveness of concepts for the representation and definition of data structures, powerful conceptual data models naturally appear as the best candidate. They have been purposely and carefully developed to enable building representations that are as rich and as close as possible to human perception. They have proven to be quite successful with users. The same benefits can be expected in using them to build ontologies. A number of researchers [16], [17], [1] have already argued in favor of "highly intuitive" ontology models with a "frame-like look and feel" or "database schema" alike. We support this viewpoint. Nevertheless, ontologies may require even higher expressiveness than conceptual models, as, beyond modeling, they aim at supporting reasoning on the description of the domain of interest. As will be shown hereinafter, this requires extending current conceptual data models with some additional features.

Instance Handling. Instances always exist in a database (except during the design phase), the main purpose of a DBMS being to provide efficient services for storing and handling the instances. As seen in section 1, instances do not

necessarily exist in an ontology. While in a database framework there is a clear separation between the schema (metadata) and the instances (data), and the schema definition is completed before instances are created, this separation is not always enforced in ontological frameworks, where instances may be created anytime. The database approach is normative, in the sense that the database schema defines how the world is, and instances are accepted only if they fully comply with the definitions and constraints stated in the schema. The ontology approach is only partly normative, as it accepts instances as long as they do not explicitly contradict the knowledge already in the ontology, without requiring that all expected data being present. When an inference mechanism finds that an instance should hold a characteristic that is not present, the ontology assumes that the instance does hold it. In other words, databases work with a closed-world assumption, while ontology systems apply an open-world assumption.

Reasoning. The ontology world seems to follow a collaborative approach, where the conceptualization at hand may continuously evolve through updates from a community of users, without a normative policy or sequence ruling the process. For example, the specification of a concept (e.g., the *Paper* object type in the conference example) may be changed anytime, irrespectively of the fact that the ontology holds instances of the concept. In the ontology approach, the specification of a concept defines the condition that its instances must verify. At any time instances are classified in concepts according to these specifications. If the specification of the concept or the characteristics of the instance are modified the classification of the instance is automatically updated. This flexibility is enabled by the existence of powerful reasoning mechanisms. An even higher flexibility is provided in ontology approaches that support so-called spanning objects [18] i.e. objects that are both at the instance and at the type level (these objects are both source and target of instance-of links). For example in [18], an Ape may be an instance of the Species type and a type for ape objects. Although theoretically possible by introducing a meta-schema level, spanning objects are not supported by database technology, which for pragmatic reasons limits its interest to the two basic levels, schema and data.

Queries. Querying in databases is used to retrieve data. Queries are expressed on the schema, which is supposed to be known to users that want to formulate a query. Ontology users are more prone to start their search for data by wondering about what information is actually held by the system. These users (or agents) will first query the ontology schema, to identify what relevant information exists, and then proceed to query the data to extract the desired information from the underlying databases.

4 Data Modeling

In the previous section, we have argued that, due to strong similarity between descriptive ontologies and database schemas, conceptual data models are good candidates for ontology modeling. In this section we analyze differences between constructs in conceptual models and in current ontology proposals.

Object Structure. Ontology models, as we have seen, adopt a binary (also termed functional) approach. Objects are mere tokens (i.e., objects with only an identity and no value) that gain their semantics through binary relationships with other objects or value domains. The known disadvantage of the approach is that a real world entity is scattered into its most elementary pieces and the vision of the thing as a whole is lost. Conceptual models, like MADS, that support complex (NF2like) object structures can represent each real world entity as a single object. This greatly reduces the complexity of the schema. Figures 1, 2 and 3 show that, even for an over-simplified example, the difference in readability is important. The MADS diagram in Figure 1 only needs 7 object types, while the equivalent DL diagram needs 27 objects types. The latter also doubles the number of relationship types if its inverse roles have to be represented. The gain in semantic expressiveness induced by complex objects is worth the additional challenge in implementation.

Object Identity. There is a general agreement that object instances should have a unique object identity. Originally, object identity is system defined and not visible to users. Some ontology approaches (including KAON and RACER) leave it up to users to define the identity of each object. In our opinion, this policy hardly scales up to the very large sets of instances that may be expected in future ontologies.

Generalization Hierarchy. Is-a links, with population inclusion semantics and property inheritance, and generalization hierarchies (or lattices) are standard constructs in both ontology languages (where the term subsumption is often preferred to the term is-a) and conceptual models. Notice that rules for generalization hierarchies in conceptual models may differ significantly from object-oriented models rules. MADS, for instance, allows an object instance to dynamically gain (or loose) membership in (or from) other classes. MADS also supports multi-instantiation, i.e. a real world entity can be represented by several instances belonging to different classes. For example, a person can be both an author and a reviewer. A generalization hierarchy may similarly be defined on relationship types. DL models follow a similar approach. However, they have different default assumptions. In KAON and DL models (e.g. RACER), by default any two concepts may contain common instances. In conceptual models, like MADS, by default two object (or relationship) classes with no common ancestor in the generalization hierarchy (but the root) are disjoint. With the permissive approach of DL, non-careful users may unwillingly create unwanted multi-instantiations that are automatically deduced by the inference engine from their assertions.

Defined constructs, views and derivations. The main goal of ontologies, supporting precise definitions of concepts in relation to other concepts, is fulfilled by the possibility to define concepts using an intentional formula. For example, based on the Conference ontology, one may want to define new concepts such as *PCPaper* (to represent papers submitted by at least one member of a committee), *ChairPerson* (persons chairing a committee), and *SwissAuthor* (authors from Switzerland). In DL these defined concepts are managed exactly in the

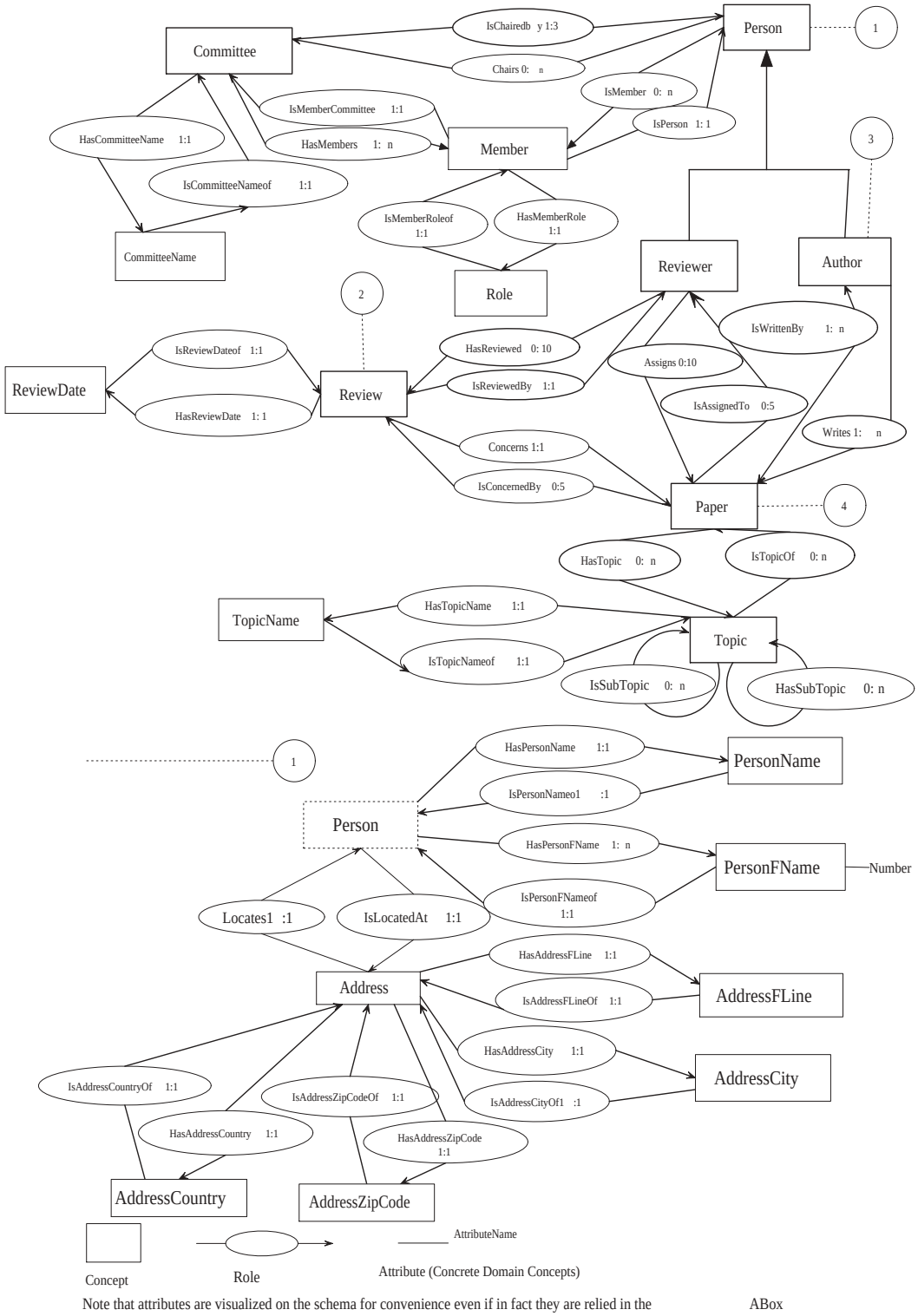

Fig. 2. Scientific Conference Ontology Snapshot with DL formalism (Part1)

same way as the other concepts (the primitive ones), which means, among other things, that they belong to the same generalization hierarchy.

Databases, interested in redundancy-free schemas and data, are not prone to support defined constructs. Nevertheless, some DBMS provide a somehow similar functionality through views and derived constructs. Views are relations (or object classes) defined by a query. Their goal is to provide users with another presentation of a subset of the database in order to make user querying easier. Views do not belong to the database schema. They form an external level that acts as an interface for the users. In terms of supporting derivation mechanisms, some conceptual models allow designers to associate to an object class (or relationship class or attribute) an expression that automatically generates the instances of the class (or the values of the attribute). The designer also explicitly defines the structure of the derived class and its position in the generalization hierarchy. Only the identities of the instances (the values, in case of an attribute) are automatically inferred. An example of a derived construct is a cluster of is-a links defined by a classifying attribute whose value determines which sub-class each instance belongs to. Another example is MADS support of derived topological relationships. For instance, a derived inclusion relationship *On* may be defined to link two spatial objects classes *Parcel* and *House*: Each time the geometry of a house is inside the geometry of a parcel, the system automatically generates an instance of the *On* relationship that links the house and the parcel.

Let us compare these three mechanisms using the following criteria:

- *Which modeling constructs can be defined, derived or can be a view?* Description logics support defined concepts and roles. Both relational and object-oriented DBMS support views for their main construct only: tables or object classes. MADS supports derived spatial and temporal relationships and derived attributes. Defined object classes could be implemented by queries whose resulting type would be added to the database schema. Description logics, RACER, and KAON also support some kind of derived relationships: Instances of transitive, symmetric, and inverse roles are automatically inferred. DOGMA supports symmetric cyclic relationships.
- *What is the status of the defined, derived construct, or view?* In DL there is no difference between defined and primitive concepts except their definition. In databases the specificity of a derived construct is that it cannot be instantiated directly by users. View is a special construct that does not belong to the set of constructs of the data model.
- *How powerful is the defining formula, derivation expression, or the query defining a view?* As the data model of DL is based upon token objects, DL formulas have to define identities only, while in databases derivation expressions and queries for views have to define identities with the associated structured values. Another difference is that DL formulas define only sets of existing instances, while queries can be either object preserving (defining new representations for existing objects) or object generating (creating new objects with new oids). Examples of such queries for the Conference

ontology are, respectively: *PCPaper*, the set of papers written by at least a member of a committee, and the new relationship class *ReviewerAuthor* that links each author to each of his/her reviewers. Borgida [19] showed that DL formulas have a limited power compared to databases query languages. They are equivalent to first order logic with 3 variables. For instance, it is easy to write a query that finds the papers that have been assigned to one of their authors (an error of assignment), while in DL it is impossible to write the equivalent formula. As for derivation expressions, they vary according to the data model. Often they are predefined and therefore are expressions with limited power.

In conclusion, one could roughly say that conceptual models are better at designing primitive concepts because they can describe more complex structures, closer to the real world, and because they support appealing visual diagrams and design tools. [5] They also support derived constructs whose instances can be automatically inferred. But, contrarily to models based on DL, they do not support defined constructs that designers can define by a logical formula without knowing where they will fit in the generalization hierarchy or even knowing the generalization hierarchy.

5 Instance Handling

Ontologies may include instances, as databases routinely do, as part of their domain of interest. To realistically manage large sets of instances, storage and transaction management mechanisms that support security, concurrency, reliability, query optimization, and scalability are needed. As this is exactly what a DBMS provides, DOGMA, KAON, and MADS delegate such services to an underlying DBMS. KAON, for example, stores ontology instances in a relational database [18]. DOGMA, KAON, and MADS are built as a layer in between users and the DBMS, providing an ontological or conceptual modeling perspective on the data. RACER, instead, uses a proprietary file system, which limits portability and does not provide all of the above services. Instance management includes manipulation facilities such as insertion, deletion and updating. They should be accessible both via some user-oriented assertional language (à la SQL, for example) and via some API providing one instance at a time access. Both types of DML are fully supported by a DBMS. RACER provides only elementary facilities. Attribute values as treated as objects, which requires three operations to define a value for a simple attribute of an instance: (1) creation of an object-value, (2) assigning the value to the object-value, and (3) linking this object-value to the instance. DOGMA, KAON, and MADS offer (or plan to offer) a conceptual DML that corresponds to the data modeling paradigm they

[5] In order to achieve readibility and understandability by users, visual diagrams purposely limit their expressive power to a subset of the concepts in the conceptual model. They are complemented with textual specifications that complete the description of the schema.

use. Users can, as in RACER, load and manipulate instances that obey the rules of the ontology, without having to consider the representation format used to store instances.

Ontologies, however, need additional manipulation facilities. DBMS users are expected to know the schema and issue manipulation requests that conform to the schema and the associated constraints. For example, to insert a new instance users have to explicitly specify its type and to provide its value and links. Moreover the value and links have to obey the format and constraints defined for this type. Description logics assume users (humans or agents) may be only partly (if at all) aware of the schema (concepts and role definitions). A DL schema hence acts like a set of sufficient conditions that define the membership of the instances to concepts (or roles). Therefore, DL systems allow users to insert a new instance giving only an intensional definition (i.e., a logical formula) characterizing the target concept. The reasoner then computes the concepts the instance belongs to. Finding the most specific concepts an individual belongs to is called *realization*. For example, RACER supports the definition of an instance by specifying its properties *instance Mary (some Writes Paper)* and realization allows finding its most specific concepts. The system will deduce that the instance *Mary* is an instance of *Author*. Realization may be much more complex than in this very simple example.

Database systems are not meant to provide such looseness in instance manipulation. As already stated, they are normative. They follow the closed world assumption, stating that only information that is present in the database (or derivable by explicitly defined derivation rules) is valid. Consequently, they do not need sophisticated reasoners to infer additional information. DL systems naturally adhere to the open world assumption, which assumes that present data is just the explicitly known subset of the valid data, and more valid data may be inferred by sophisticated reasoning. Thus, if an assertion implies a deduced fact that is consistent with all known assertions and instances, then the fact is assumed to be true even if it is not present in the instance set. Otherwise stated, an insertion in DL is always treated as the insertion of incomplete information. For example, a database will accept the insertion of the above instance *Mary* in *Author* only if it comes with the insertion of (at least) one instance of the *Writes* relationship involving *Mary*. RACER accepts the insertion of the single *Mary* instance, deducing that the paper written by *Mary* is presently unknown (see [20], for more on this discussion). In addition, *Recognition* is performed when an already existing instance of a concept acquires (or loses) a characteristics and therefore gets (or loses) a new instantiation in another concept. For example, on the insertion of a new role instance linking a *Person* instance p to a *Paper* instance, RACER infers that p is also an instance of *Author*.

6 Constraints

In the database world, constraints significantly enrich data description. They state rules that apply in the real world of interest (e.g., one paper has at least

one author) and rules that define the conditions for real world phenomena to be or not to be of interest for the intended application (e.g., a committee is not registered in the database before at least one of its members is registered in the database). Constraints, as other schema definition statements, are understood in the DB world as normative specifications. They entail consistency-checking mechanisms, to verify that instances in the database satisfy all constraints. DL languages are also able to express rules similar to database constraints (e.g., min-max cardinalities). However, only some of them actually constrain the instances in the A-Box. For example, a maximum cardinality specification acts as a constraint as an attempt to create more instances than allowed would result in an inconsistency that is detected and rejected. On the other hand, a minimum cardinality specification only acts as a descriptive feature, as a DL system would accept, e.g., the creation of a paper without an author, simply assuming that the information in the instance base (the A-Box) is temporarily incomplete. We believe that the possibility to define normative constraints, as in the DB approach, is a desirable feature also for ontologies.

There are such a variety of constraints that data models almost necessarily only include part of them in their constructs. Implicit *model constraints* rule the use of modeling constructs and are built-in in the data model, i.e. they act as syntactic constraints that are automatically enforced by the data management interface. For instance, in RACER a role definition has to specify a domain and a range; in conceptual data models a relationship type is only allowed to link object types, and a relationship instance is not allowed to have pending roles; in both DL and DB models cycles of is-a links are forbidden. Explicit definition of constraints is used to describe the semantics of the domain/ontology. According to the approach, they come in two different ways. 1) *Embedded constraints* are expressed using dedicated constructs in the data model. Examples include cardinality and identifier specifications (e.g., the NOT NULL and PRIMARY KEY clauses in relational DBMS), set constraints on groups of roles or is-a links (e.g., disjunction, inclusion, cover, partition), and simple integrity constraints (e.g., using the CHECK clause in relational DBMS). 2) *Integrity constraints* are those that are not directly supported by clauses in the model itself, and thus have to be explicitly expressed using a complementary technique, such as a generic declarative language (e.g., first order logic), a generic programming language (e.g. stored procedures or methods), or triggers. As DL axioms can define a large range of embedded constraints, DL approaches do not resort to additional mechanisms for integrity constraints definition. On the contrary, DB approaches rely on such mechanisms. DL also differs from DB approaches in that DL allows associating a constraint to an instance, while DB considers constraints as meta-information and always associates them to types (thus constraining under the same rule all instances of the type). In terms of expressiveness of the language to define constraints, first order logic (FOL) is the closest to full expressive power using a declarative approach. DL languages usually support a more limited expressive power. For instance, RACER cannot express key constraint involving multiple attributes and ad hoc constraints such as "an author cannot

review his/her papers". More expressive DL such as $\mathcal{DLR}$ supports this kind of constraints.

A meta-question about constraints and ontologies is whether constraints should be included in ontologies at all. Most frequently, constraints are intertwined in the T-Box with the data description statements. Meersman and his group [2] take the opposite view that all constraints should be separated from data description and defined in a commitment layer. The supporting argument is that constraints are application specific, while the ontology should be application independent. We agree that having a commitment layer is the appropriate way to handle application-specific semantics. Nevertheless, there are in our opinion constraints that belong to the ontological world, i.e. that form an essential component in the description of the semantics of things. For instance, the fact that in a conference management ontology a review of a paper should never be assigned to one of the authors of the paper is a constraint that is unanimously agreed upon. It is a constraint that is tightly linked to the semantics of a review (defined as a critical appreciation of a work by a person not involved in the work). On the contrary, whether a conference committee has one or more chairpersons varies from one conference to the other. The ontology could state a 1:N cardinality (to make sure a committee has at least a chair), leaving to the commitment layer to refine the cardinality to 1:2 or 1:3 or whatever else fits the conference specific requirements.

Checking the consistency of the set of constraints and checking the consistency between the constraints and the schema are tasks that can be performed automatically by the reasoners available in description logics. Constraints are expressed in the same formalism as the other description clauses; hence they are naturally involved in the inferring. The same functionality is not provided by current database technology, where applicable reasoning techniques cannot grasp the semantics hidden in the external language expressions used to define integrity constraints.

Checking the consistency of schema specifications is also an issue where DL and databases take different approaches. In databases, it is not possible to validate a schema that does not obey model constraints. As there is no defined construct, there is no need for sophisticated reasoning in checking the consistency of the schema. Reasoners are necessary in DL to check the consistency of primitive, defined concepts, and other axioms. They define a valid schema as a schema such that it is possible to find one instance of the ontology that has at least one instance in each one of its concepts. While database users can never actually use an invalid schema, in RACER, for example, users can request a consistency check anytime and they can continue working on an inconsistent schema.

7 Ontology Querying

Like databases, ontologies are queried by different categories of users, with different needs:

- Ontology administrators, whose role (like DBAs, database administrators) is to design and maintain the ontology schema and monitor its evolution. While a DBA is seen as a central authority, ontology creation and evolution is often seen as a more cooperative activity, distributing the task among many people. Hence schema querying is likely to be a more intensively used functionality, with the schema continuously and incrementally growing with many defined concepts.
- End-users, who will face a large and complex schema that they may not know well. They also will query the schema to know what is in the ontology. They may also write mixed queries to access both the schema and the instances. For example, a user of a geographic ontology describing the various states of a country may ask for all information (list of properties with their description and value) about rivers and synonyms of rivers. Therefore, both the schema and the instance base should be accessible through the same query language, possibly within the same query.
- Application developers, who need to gain access to the ontology and its services via an API.

Requirements for instantiated descriptive ontologies include the usual services supported by DBMS, namely a generic assertional query language with associated tools for automatic query optimization. The expressive power, and its optimization possibilities, of the language are bound to the characteristics of the associated ontology model. For instance, queries in a language for a binary model, like those of KAON and RACER, will return types or instances of the elementary constructs of the model: concepts and binary relationships. On the other hand, queries on semantic models with structured objects, like MADS, will return structured instances, i.e. more informative and more condensed results. Therefore, for descriptive ontologies frameworks, where understanding the data structure may be a challenge, semantic query languages returning structured objects are likely to perform better than languages for binary models.

Another requirement is that the same query language should support querying both the schema and the instances in the same way. Models that host the description and the instances of the ontology within the same structure automatically fulfill this requirement as it is possible in RACER or KAON. For models that clearly separate the schema from the instances, like database models do, a solution is to store the schema as instances of a meta-schema described with the same model as the ontology. Such a solution is currently provided by relational DBMS that support a data dictionary made up of a set of predefined tables that describe the tables of the application schema.

Several functionalities should be provided for schema querying. Exploration of the schema is the first one. When the schema contains concepts defined by logical formulas, reasoning comes as the second one.

Schema exploration. The query language should allow getting all information existing in the schema. Examples of such queries could be:

Give the characteristics of a relationship (transitive, symmetric, inverse)
Give the relationships going from (or to) a concept.

Queries of this type can be formulated in the RACER language using the elementary predefined functions that are provided for navigation inside the schema: *describe-concept, describe-role, reflexive?, symmetric?, transitive?, feature?, role-inverse, role-domain, role-range*. KAON also provides similar functions, such as *Properties_From, Properties_To, Domain_Concept, Range_Concept, SubConcepts, SuperConcepts*.

In MADS, this can be done by defining the schema of the meta-model of MADS, and querying this meta-base with one of the generic languages of MADS (visual or textual algebraic). However, for humans a much simpler way to explore the schema is to visualize and browse it using the MADS schema editor.

Reasoning on the schema. Users of ontologies that contain non-primitive concepts defined by logical formulas need a schema query language with new functions for helping them in their understanding of the defined concepts. Typical functions of this type are (here by concept we mean any kind of concept, be it primitive or defined):

Are two concepts equivalent or disjoint? Does a concept (or relationship) subsume another one? Classify the whole set of concepts. What are the super- or sub-concepts (at any level) of a concept?

These functions require an inference engine for their evaluation. This justifies the choice of formal models, such as DL, that have powerful tools to classify concepts using the subsumption mechanism.

Instances querying. Databases and DL offer complementary functionality for instance querying. Databases systems usually provide powerful assertional query languages complemented with efficient query optimization tools. These languages, like the ones of MADS, support object preserving as well as object generating queries. They also allow users to define new structures for existing objects by pruning existing properties or computing new, derived properties. On the other hand, DL systems support a set of simple functions for accessing instances and derived facts computed by their inference engines, like the closure (resp. inverse, symmetry) of the transitive (resp. inverse, symmetric) relationships. Moreover DL systems, like RACER, that allow users to associate logical formulas to instances, provide a new reasoning function: "To which most specific concepts does this instance belong?"

8 Additional Requirements

Up to now we have only discussed traditional requirements as addressed by current ontology frameworks. This section highlights some additional features that we believe will in the short term gain importance for the full development of ontologies. The first and most evident additional feature is the possibility to associate temporal specifications to the concepts and roles of an ontology. The semantics of terms and concepts evolves in time, new terms and new concepts appear while other become obsolete. It is therefore important that each item in an ontology be qualified using a temporal specification that says when the definition of this item is valid. How to define and implement such lifecycle spec-

ifications for concepts and roles, as well as time-varying attributes, has been thoroughly investigated by the temporal database community. Results should simply be taken over to ontologies. Similar considerations may apply to spatial specifications, well-known in the world of geographical information systems (GIS). They may describe, for instance, the geographical coverage of a given term (e.g. "char" to denote a car holds in Quebec but not in Paris). Moreover, research in data visualization has shown that ontologies may be displayed as a concept space, where spatial concepts such as distance, neighborhood, inclusion, or orientation may fully apply. Spatial information supports storing the position and topology of concepts in such abstract spaces. Spatial information will also play an important role for ontologies where geographical aspects are part of the domain of discourse. To support its description, concepts and techniques may be borrowed from GIS research. Finally, as we have importance as the actual use of ontologies becomes practically relevant. The need for context information is recognized in the ontology literature, but the current status shows limited results and significant advances may still be foreseen in this domain. How to define a context, how to analyze interrelationships between contexts, how to characterize constraints on contexts, are examples of open research issues. RACER and KAON currently support none of these additional features. Some extensions of DL have been proposed for spatial and temporal modeling [21],[22]. DOGMA includes context information. MADS supports space, time, and context description and manipulation.

9 Conclusion

Ontologies are promised to a brilliant future. As a consequence, usability criteria will assess their success. In our opinion, this means that focus will be on more informative ontologies, showing, in addition to terms and concepts of a domain, how domain data are semantically structured and interrelated. We termed these ontologies descriptive, as opposed to first-generation taxonomic ontologies. The paper investigated requirements for the design and management of descriptive ontologies, and contrasted the requirements with the functionality currently provided by database conceptual models. Proper identification of the requirements has been supported by an analysis of some recent representative proposals for ontology systems (namely, RACER, KAON, and DOGMA). The rationale for this work is the close resemblance between requirements for database design and those for ontology design. We attempted to highlight similarities as well as significant differences in the approaches. Most differences appear to be linked to the current state of art in both domains. These ones may disappear thanks to further research. However, important differences (such as closed versus open world assumptions) are inherently due to the different goals of ontology and database services. Ontologies are meant to describe and explain the world, while databases are meant to describe that part of the world whose representation has to be managed for some application purpose. Overcoming differences is a meaningful way to benefit one domain with results from the other domain. Pre-

vious work has investigated how to extend description logics to provide more data semantic services, or how to map description logic specifications into conceptual model specification, and vice versa. Our aim has been to identify the enhancement that conceptual models would need to make them fit the requirements of ontologies. Briefly stated, the necessary enhancements have obviously to do with supporting reasoning. A major addition is the support of intentionally defined concepts à la DL. This somehow resembles view definition and queries in databases, but views are not part of the database schema and queries raise a number of open issues (e.g., how to place the query object type in the generalization hierarchy in order to explicit the semantic relationship between the new type and the existing types). A minor addition is the explicit definition of the specialization criterion in is-a clusters, so that the system can compute the appropriate sub-class for an object whose value changes or is first created. More additions that we feel important to match coming requirements are provision for spatio-temporal data modeling and context management. We have currently defined and implemented a conceptual data model, MADS, that supports advanced data structure, time, space, and context modeling requirements, as well as query placement to some extent. As a further step towards ontologies, we are extending the model to support imprecise information, where incompleteness is seen as a form of imprecision. This is intended to allow building ontology services above the conceptual services currently provided by prototypes developed within the MurMur IST project [23].

References

1. Meersman, R.: Ontologies and Databases: More than a Fleeting Resemblance. In: OES/SEO Workshop Rome. (2001)
2. Jarrar, M., Meersman, R.: Formal Ontology Engineering in the DOGMA Approach. In Meersman, R., Tari, Z., et al., eds.: CooPIS/DOA/ODBASE, Springer-Verlag, LNCS 2519 (2002) 1238–1254
3. KAON: KAON - The Karlsruhe Ontology and Semantic Web Tool Suite (2003) http://kaon.semanticweb.org/.
4. Horrocks, I.: DAML+OIL: A reason-able Web Ontology Language. In Jensen, C., et al., eds.: EDBT 2002, Springer-Verlag, LNCS 2287 (2002) 2–13
5. McGuinness, D.: Description Logics Emerge from Ivory Towers. In: Proceedings of the International Workshop on Description Logics, Stanford, CA (2001)
6. Borgida, A., Brachman, R.J.: Conceptual Modelling with Description Logics. In Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: Description Logic Handbook, Cambridge University Press (2002) 349–372
7. Borgida, A., Lenzerini, M., Rosati, R.: Description Logics for Databases. In Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: Description Logic Handbook, Cambridge University Press (2002) 462–484
8. Calvanese, D., Giacomo, G.: Expressive Description Logics. In Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: Description Logic Handbook, Cambridge University Press (2002) 178–218
9. Haarslev, V., Möller, R.: RACER System Description. In Goré, R., Leitsch, A., Nipkow, T., eds.: Proceedings of International Joint Conference on Automated Reasoning (IJCAR'2001), Springer-Verlag (2001) 701–705

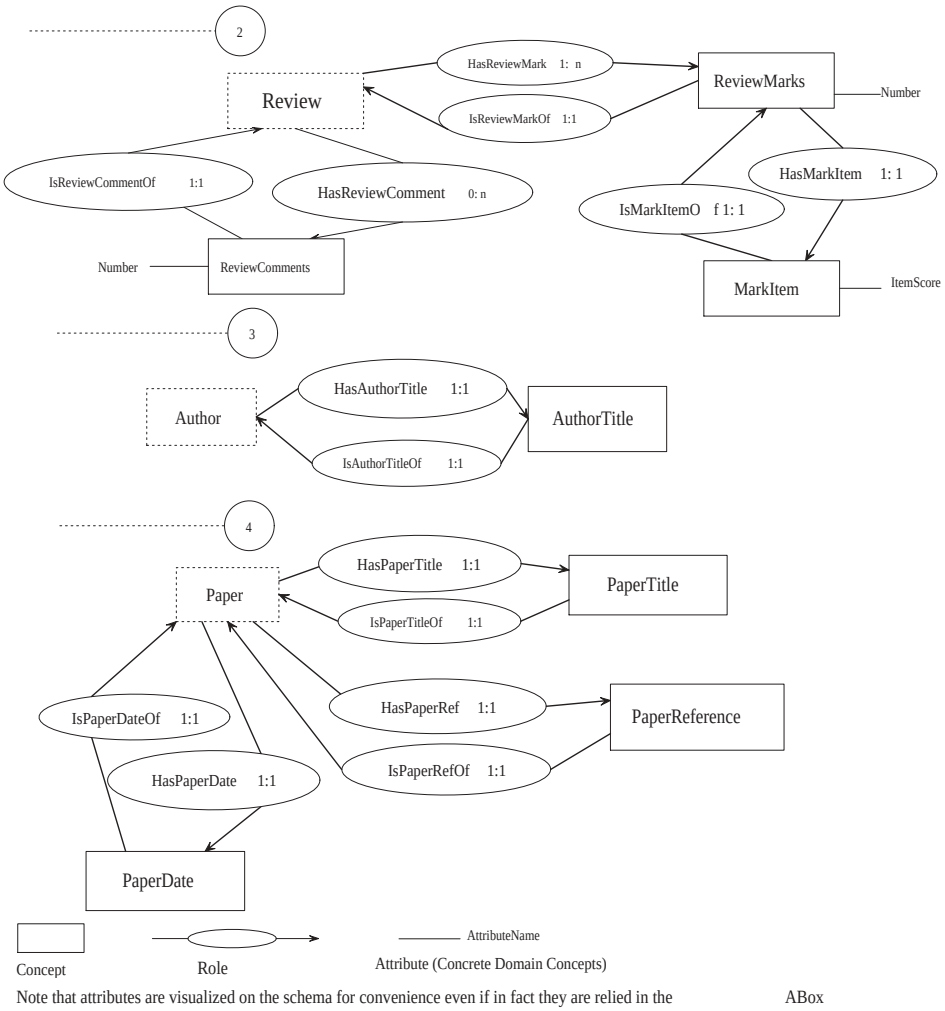

Fig. 3. Scientific Conference Ontology Snapshot with DL formalism (Part2)

10. Horrocks, I., Sattler, U., Tobies, S.: Reasoning with Individuals for the Description Logic *SHIQ*. In MacAllester, D., ed.: Proc. of the 17th Int. Conf. on Automated Deduction (CADE-17). LNAI 1831, Springer-Verlag (2000) 482–496
11. Horrocks, I., Sattler, U.: Optimised Reasoning for *SHIQ*. In: Proc. of the 15th European Conference on Artificial Intelligence (ECAI'2002). (2002) 277–281
12. Halpin, T.: Information Modelling and Relational Database Design. (2001)
13. Spaccapietra, S., Parent, C., Zimanyi, E.: Spatio-Temporal Conceptual Models: Data Structures + Space + Time. In: 7th ACM Symposium on Advances in Geographic Information Systems (ACM GIS'99). (1999) 26–33
14. Spaccapietra, S., Parent, C., Vangenot, C.: From Multiscale to Multirepresentation. In Choueiry, B., Walsh, T., eds.: Proceedings 4th International Symposium, SARA-2000, Horseshoes Bay, Texas, USA, Springer-Verlag, LNAI 1864 (2000)
15. OKBC: OKBC - Generic Knowledge Base Editor (1998) http://www.ai.sri.com/ gkb/.
16. Fensel, D., Hendler, J., Liebermann, H., Wahlster, W.: Spinning the semantic web, The MIT Press, Cambridge, Massachusetts (2003)
17. Klein, M., Broekstra, J., Fensel, D., van Harmelen, F., Horrocks, I.: Ontologies and schema languages on the Web. In D. Fensel, J. Hendler, H.L., Wahlster, W., eds.: Spinning the Semantic Web, The MIT Press, Cambridge, Massachusetts (2003)
18. Motik, B., Maedche, A., Volz, R.: A Conceptual Modeling Approach for Semantic-Driven Enterprise Applications. In Meersman, R., Tari, Z., et al., eds.: CooPIS/DOA/ODBASE, Springer-Verlag, LNCS 2519 (2002) 1082–1099
19. Borgida, A.: On the relative expressive power of Description Logics and Predicate Calculus. In: Artificial Intelligence 82. (1996) 353–367
20. Baader, F., Nutt., W.: Basic Description Logics. In Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: Description Logic Handbook, Cambridge University Press (2002) 43–95
21. Haarslev, V., Lutz, C., Möller, R.: Foundations of Spatioterminological Reasoning with Description Logics. In A.G. Cohn, L.K. Schubert, S., ed.: Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98). (1998) 112–123
22. Artale, A., Franconi, E.: A Survey of Temporal Extensions of Description Logics. In: Annals of Mathematics and Artificial Intelligence (AMAI), Vol. 30 No. 1-4, Kluwer Academic (2001)
23. MurMur: MurMur Consortium - MurMur Project: Multi-representations and Multiple resolution in geographic databases (2002) Final Report. http://lbdwww.epfl.ch/e/MurMur.