On Using Conceptual Modeling for Ontologies

S. Spaccapietra[1], C. Parent[2], C.Vangenot[1] and N. Cullot[3]
[1] Database Laboratory, EPFL, 1015 Lausanne, Switzerland
stefano.spaccapietra@epfl.ch, christelle.vangenot@epfl.ch
[2] HEC-INFORGE, University of Lausanne, 1015 Lausanne, Switzerland
christine.parent@unil.ch
[3]LE2I Laboratory of Business, University of Burgundy, 21000 Dijon, France
nadine.cullot@u-bourgogne.fr

Abstract. Are database concepts and techniques suitable for ontology design and management? The question has been on the floor for some time already. It gets a new emphasis today, thanks to the focus on ontologies and ontology services due to the spread of web services as a new paradigm for information management. This paper analyzes some of the arguments that are relevant to the debate, in particular the question whether conceptual data models would adequately support the design and use of ontologies. It concludes suggesting a hybrid approach, combining databases and logic-based services.[1]

1. Introduction

Nowadays, all major economic players have decentralized organizational structures, with multiple autonomous units acting in parallel. New information systems have to handle a variety of information sources, from proprietary ones to those available in web services worldwide. Their complexity is best controlled using a network of coordinated web services capable of grasping relevant information wherever it may be and exchanging information with all potential partners. Data semantics is at the heart of such multi-agent systems. Interacting agents in an open environment do not necessarily share a common understanding of the world at hand, as used to be the case in traditional enterprise information systems. For instance, in a single enterprise environment, the concept of "employee" has a unique definition shared by every application within the enterprise. In a multi-agent system, the interpretation of the "employee" concept may vary based on whether or not specific types of personnel (e.g., students in their summer jobs, trainees, visitors) have also to be considered as employees. Another example is obviously provided by contextual information, such as whether a sentence about trees refers to the vegetal or to the mathematical structure. This is also a form of semantic disambiguation.

[1] This work is supported, in the framework of the EPFL Center for Global Computing, by the Swiss National Funding Agency OFES as part of the European projects KnowledgeWeb (FP6-507482) and DIP (FP6-507483). It is also supported by the MICS NCCR funded by FNRS in Switzerland, under grant number 5005-67322 .

Lack of common background calls for explicit guidance in understanding the exact meaning of the data. XML-like formatting does not help much in this. Ontologies increasingly appear as the solution to the problem. They are the most sophisticated form of semantics repository. From a database perspective, they may be intuitively understood as the most recent form of data dictionaries, i.e. a knowledge repository whose purpose is to explain how concepts and terms relevant to a given domain should be understood. Although ontology as a science comes from philosophy, ontologies as computerized support for semantics have mainly been developed by the artificial intelligence community. This community has focused on developing reasoning mechanisms that would alleviate the task of enriching an ontology by addition of new concepts. Typically, an ontological reasoner is expected to be able to check the consistency of new concepts with already known ones and to determine their most accurate placement within the (most often hierarchical) structure of the ontology.

With ontologies becoming a necessary component of modern, interoperable information systems, we are likely to see a proliferation of ontologies and a massive growth in size and complexity of the set of concepts described in an ontology. We foresee that their role will evolve from a repository of terms that denote concepts (whose most well-known example is Wordnet) to a repository for complex information, where the description of a concept includes a formal description of a prototypical data structure (a design pattern) showing all the components of a concept, the intra-relationships between these components, and the inter-relationships between the concept and the other concepts in the ontology. These richer ontologies will have to be easily understandable, and processable, by humans and by computerized agents in search of semantics. Briefly stated, we expect significantly increasing similarity between ontologies and current database schemas.

This raises the question whether database technology could be reused to provide services for ontology design and management. The purpose of this paper is to develop some arguments for such a discussion. The arguments we present here focus on structural aspects, as we are interested in showing the benefits of using conceptual data models for modeling ontologies. Other arguments (e.g., discussion of defined versus derived objects, axioms, schema and instance querying, constraints) are also surveyed. For a more detailed analysis of the latter the reader is referred to [3]. Our idea is that database techniques could nicely complement logic-based techniques (using formalisms such as description logics or F-Logic) and a common framework could be built that would exploit the technique that best fits the task on hand.

The next section introduces an example that we use in Section 3 (after briefly recalling the concepts of the underlying conceptual data model) to illustrate how a conceptualization can be formulated using a conceptual data modeling formalism. Section 4 discusses respective merits of using data modeling versus description logic formalism. Section 5 concludes suggesting to combine both formalisms into a hybrid system. We assume the reader is familiar with the features supported by ontological formalisms (e.g., description logics).

2. A Motivating Example

The example we show hereinafter has been first introduced by Boris Motik as part of deliverable D1.1 of DIP, an European project aiming at developing semantic web services. Let INT be a company that wants to provide a web service consisting in an integrated tourism portal offering hotel information. INT does not itself own hotel data. Instead, it simply integrates web services by various providers. Each provider classifies its hotel data in a proprietary structure. Let us assume that there are two providers of such information, TUI and Thomas Cook, TC for short. Figure 1 presents the conceptualization that TUI uses to describe his offers. The drawing uses a simple notation, based on an underlying binary data model, the kind of model many ontology tools adhere to. Oval nodes (e.g. Hotel) denote conceptual entities. Nodes without a surrounding oval (e.g. name) denote properties of the corresponding entities. Labeled arcs between oval nodes denote a relationship between the conceptual entities. Non-labeled arcs link a conceptual entity to its properties.

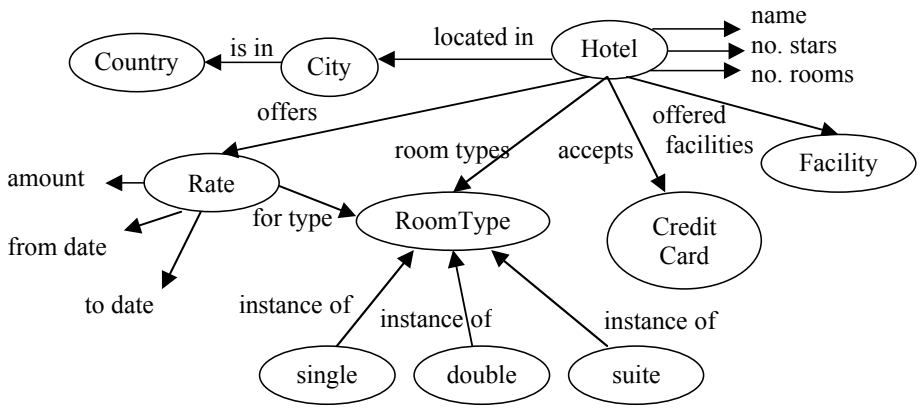

Figure 1. Conceptualization of TUI

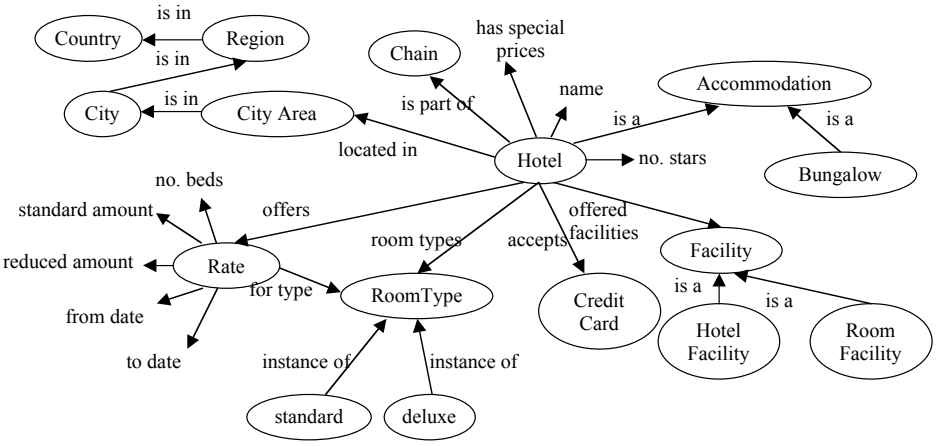

Figure 2. Conceptualization of TC

TC uses a different conceptualization of his business domain (cf. Figure 2). For example, geographical information is more fine-grained, as it represents the country, region, and the area within the city where a certain hotel is situated. Room types are split according to a different criterion. The rate structure is also different.

To create its portal, INT needs to bridge the semantic differences between the information coming from TUI and from TC. First, INT must decide upon its own conceptualization of the domain. It may either accept one of the existing conceptualizations, or choose from existing domain standards, or develop its own proprietary conceptualization. After choosing the conceptualization formalism and creating its own conceptualization, INT has to describe how data in each source relates to its own conceptualization. Conceptualization formalisms offer different primitives for performing this task. Ontological work focuses mainly on logic-based conceptualization formalisms, because their inference capabilities are commonly seen as a key to solving this type of problem. For example, dependencies between the conceptualization of INT and those of the sources can be expressed as logical axioms, which, when executed, can be used to actually perform data integration. On the other hand, conceptual modeling approaches most frequently use data manipulation languages to express the mapping between a global integrated schema (here, the INT conceptualization) and its corresponding source schemas. Alternatives for the description of these mappings are well documented in the literature.

3. Conceptual Modeling for Ontologies

As ontologies grow in size and in practical relevance, it is legitimate to question whether database techniques could provide interesting support for ontology management. On the one hand, database systems are known to offer scalable and efficient management of huge amounts of structured data, which is what ontologies may become in the near future. On the other hand, conceptual modeling approaches (that have specifically been designed to support a semantically rich description of structured data sets) could, at least to some extent, handle the description of the conceptualization that is the subject of an ontology. Exploring this idea is definitely worth a discussion. Arguments in favor of "highly intuitive" ontology models, with a "frame-like look and feel" or "database schema" alike, have already been developed in e.g. [8], [5] and [7]. Specific proposals include [6] and [3]. Mappings from conceptual models to description logics have been proposed in e.g. [2] and [1].

The following brief description of conceptual modeling expressiveness is based on work on extended entity-relationship (EER) models, which are most frequently seen as offering the richest semantic expressiveness. In EER models and alike, data structures are basically graphs of object types interconnected by relationship types. Both object and relationship types may be characterized by associated properties (attributes and methods). Attributes may be atomic (as in relational tables) or composed of other attributes, thus allowing the definition of multilevel property trees for object and relationship types. It is then possible, for instance, to represent a real world entity as a single object in the database. Attribute cardinalities state whether an

attribute is optional or mandatory, and monovalued or multivalued (list, set, or bag). Relationship types connect object types via roles. A relationship type may be defined with 2 (for binary relationships) to n roles. When two or more roles connect to the same object type, the relationship type is said to be cyclic. Relationship types may be adorned with specific semantics, of which the most well known is aggregation semantics (expressing that an object is a component of another object).

Object and relationship instances bear a system-defined, unique identity. Object types and relationship types may be organized into generalization/specialization lattices using is-a links. Inheritance, refinement, redefinition and overloading mechanisms apply as proposed in traditional object-oriented data models. Some advanced conceptual models, however, depart from object-oriented rules by adopting a multi-instantiation paradigm, i.e. allowing the same real world entity to be simultaneously represented by several instances in different classes that are not in a sub-type/super-type relationship. Allowing multi-instantiation is necessary from the modeling point of view, in particular to be able to properly describe situations such as, for instance, a real world object being at the same time a hotel and a restaurant (assuming Hotel and Restaurant are two object types) without forcing the definition of a so-called intersection class, Hotel&Restaurant, sub-class of both Hotel and Restaurant. Another facility from some conceptual model is classification dynamicity, i.e. the possibility for an instance to move to another class (e.g., a guesthouse becoming a hotel, a student becoming a faculty).

Good conceptual models come with formal definitions, rules to translate conceptual specifications into logical level specifications, and implementations in several marketed CASE tools and in research prototypes.

3.1 Conceptual Design for the Example Databases

The TUI conceptualization from Figure 1 can be easily reformulated in an EER formalism by using the following very simple (but not very intelligent) rules:

- Ignore links labeled "instanceOf" and their source ellipsis (EER schemas do not describe instances),
- Each ellipsis translates into an object type,
- Each link between ellipses translates into a binary relationship named after the label associated to the link,
- Each label not in an ellipsis translates into a property (an attribute) of the object type it is linked to.

Figure 3 shows the diagram for this EER schema.

However, such a schema definition, although syntactically correct, does not fully use the power of conceptual models to organize the TUI conceptualization. Basically, it does not use the facility to define complex attributes, which allows elaborating a description of a real world entity as a single object type. Using this facility, and assuming TUI wants to keep a catalogue of cities and countries, the conceptual schema for the TUI conceptualization reduces to the one illustrated in Figure 4, with Figure 5 showing the attributes of Hotel (indenting is used to visualize attribute composition). The figures show cardinality constraints (on roles of relationship types

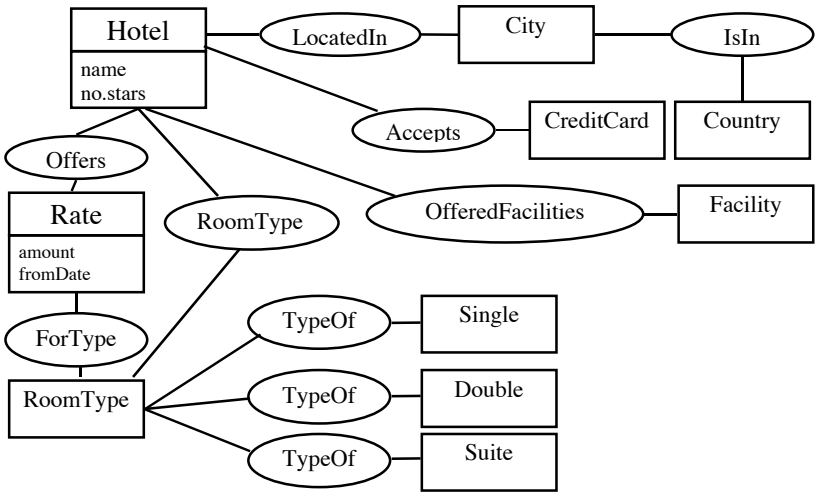

Figure 3: Straight reformulation of the TUI conceptualization (Figure 1) using EER formalism

attributes), as these are traditionally included in an EER schema definition. In Figure 4 we have assumed that a country includes many cities and a city has many hotels, while a hotel may only be located in one city (but some hotels are in no city) and a city is located in only one country. Figure 5 assumes that a hotel may have many facilities, may accept several credit cards, and offers many rates (at least one), a rate possibly holding for different types of rooms (e.g. a hotel having the same rate for single and double rooms).

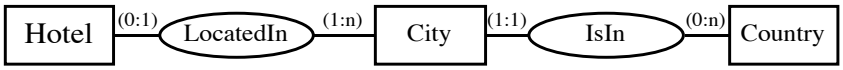

Figure 4: an EER diagram for a proper conceptual schema of the TUI conceptualization

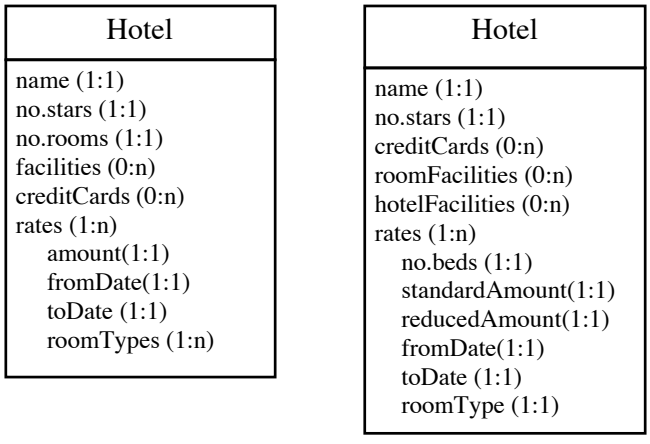

Figure 5: Hotel attributes in the TUI (left) and TC (right) conceptualizations

The same reasoning scheme may be applied to produce an EER schema for the TC conceptualization shown in Figure 2. The structural difference is that the is-a links to Accomodation in the TC structure will be mapped onto is-a links (rather than relationship types) in the EER design. A possible schema diagram for the TC conceptualization is shown in Figure 6. TC Hotel attributes are shown Figure 5.

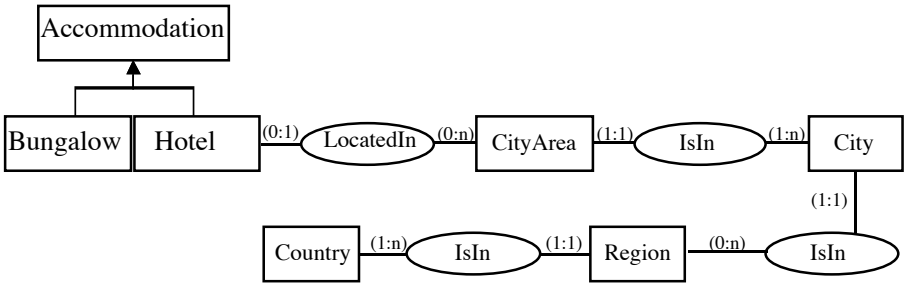

Figure 6: A possible EER design for the TC conceptualization

3.2 Integration

Bridging the semantic differences between the TUI and TC descriptions requires a precise identification of how things modeled in one description correspond to things modeled in the other description. Such mapping knowledge is expressed as interschema corresponding assertions, which we show below in the format suggested in [10] for the TUI and TC Hotel object types as described in Figure 5. For instance, assuming the hotel concept has the same semantics in TUI and TC, and assuming both providers may have offers for the same hotel, an assertion may be:

TUI.Hotel ∩ TC.Hotel
WCI TUI.Hotel.name ≡ TC.Hotel.name
WCA TUI.Hotel.creditCards ≡ TC.Hotel.creditCards
WCA TUI.Hotel.facilities ⊇ TC.Hotel.(roomFacilities ∪ hotelFacilities)
WCA TUI.Hotel.rates.amount ≠ TC.Hotel.rates.(standardAmount ∪ reducedAmount)

The first line asserts that the Hotel object types in TUI and TC describe overlapping sets of real world entities, i.e. the same hotel may be instantiated in both TUI and TC. Beyond overlapping, other possible choices are equality, inclusion and disjointedness.
The second line (WCI stands for With Corresponding Identifiers) states that two instances in TUI.Hotel and TC.Hotel represent the same real world hotel if the value for the name attribute is the same in the two instances. This provides knowledge about the mapping between instances of TUI and TC. This mapping enables gathering all information about a real world thing that is available in two interrelated sources. The following lines, introduced by the WCA (With Corresponding Attributes) acronym, define corresponding attributes for the related types, i.e. attributes that at least partially represent the same real world property, irrespectively of how it is coded in the representation. The first WCA line states that for corresponding instances the

creditCards attributes hold identical information in both TUI and TC. The second WCA line states that facilities in TUI include the facilities that in TC are split into roomFacilities and hotelFacilities. The third WCA line states that although the two rates attributes hold the same semantics (i.e., they denote the cost for a room), the set of values in TUI and TC are different (TUI and TC record different rates even for the same hotel). Hence, to retrieve all the rates offered by a given hotel, both TUI and TC have to be searched and the results have to be merged.
Similar assertions hold for relationships types. An example is:

TUI.LocatedIN ≡ TC.LocatedIn.IsIn

which states that the TUI relationship, LocatedIn, is equivalent to the composition of the two TC relationships, LocatedIn and IsIn. Both provide the same path between Hotel and City.

From these assertions, the INT conceptualization may be built almost automatically by an integration tool. "almost" refers here to the fact that many alternative INT views may be elaborated from the same set of assertions, depending on designers' preferences (e.g., one designer may prefer a more concise schema while another designer may prefer a more exhaustive schema). The integration tool generates also the mappings that relate the INT view to both sources, TUI and TC. These mappings are used by the query execution tool that translates the user queries expressed on the INT view into queries for the sources.

In a logic-based approach, the definition of the INT view is directly done by the users. They have to define each concept and role of the INT view, through a logic formula on the TUI and TC concepts and roles, as in the following examples.

INT:Hotel ≡ TUI:Hotel ⊓ TC:Hotel
INT:CreditCard ≡ TUI:CreditCard ⊓ TC:CreditCard
INT:accepts(Hotel, CreditCard) ⊇ TUI:accepts(Hotel, CreditCard)

The inference engine will automatically check if the INT description is consistent with the TUI and TC descriptions. It will also automatically infer the answers to queries on the INT view from the definitions of the INT view. Contrarily to the database approach, no new integration tool has to be provided.

4 Discussion

This section briefly surveys advantages and disadvantages of using a conceptual modeling and database approach, versus using a (description) logic approach, for the description and management of ontologies.

4.1 Data modeling

EER conceptual models support direct modeling of rich data structures, leading to representations that are close to how humans perceive things in the real world. EER synthetic schemas are easily apprehended. Instead, most description logics rely on

simple binary data structures. This leads to an explosion in the number of concepts that are needed, similar but worse than what happens in relational databases. A one-page EER schema is likely to require several pages of description logics (DL) axioms to describe the same representation. In addition, having only binary structures blurs the distinction between what describes composite things (e.g., entities and links) and what merely describes a property. The reader of a DL description has to perform a reverse engineering process to reconstruct something that resembles her/his perception of the real world that is described. EER conciseness is definitely an advantage for humans. It is also an advantage for computer agents. Agents would also have an easier task in exploring a conceptual schema showing a clear distinction between objects and complex properties than in exploring a long list of DL axioms. Finally, to visualize an ontology structure, EER diagrams are likely to be easier to capture at a glance than the DL diagrams supported by recent DL editors.

In terms of supporting description of defined or derived concepts, the advantage currently goes to DLs. DLs allow users to define new concepts by a logical formula as complex as needed. The inference mechanisms automatically check the consistency of the definitions, deduce where the new concepts are placed in the generalization hierarchy, and infer their instances. Some conceptual models support a few derived concepts (e.g., derived object types, derived classes, derived attributes), whose instances and values can be automatically inferred. But they do not support concepts that designers would define by a logical formula without knowing where they will fit in the generalization hierarchy or even knowing the generalization hierarchy. Moreover, a derived construct in a schema has different properties than a non-derived construct (e.g., it cannot be instantiated), while defined concepts in DLs are treated as base concepts. DLs also allow users to state axioms of type inclusion, equivalence, and disjointedness involving complex terms. As DLs work with the open world assumption (OWA), all the assertions (definitions of concepts and axioms) are used by the logic reasoners for inferring new knowledge.

On the other hand, DBMS, which work with the closed world assumption (CWA), enforce integrity constraints that avoid inappropriate data to enter the database, where inappropriate means data that is not consistent with the current state of the database, assuming this state is the whole universe of discourse. Conceptual models have a number of predefined integrity constraints (e.g., cardinality constraints, key constraints) that are easily described. However, to support a declarative formulation of general integrity constraints, they have to resort to an associated logic language (usually the FOL). On the other hand, in DLs it is very difficult to assert a constraint on the known part of the world [4].

4.2 Data Manipulation

Instance creation is unconstrained in DL. Instances may be created without being attached to a concept. The creation of a new instance may not conform to the rules described by the axioms. In fact, the creation of an instance leads to one of three cases: 1) the instance fully conforms to all existing assertions; 2) the instance contradicts existing assertions, in which case the user is warned about the

contradiction; and 3) the instance neither fully conforms nor contradicts existing assertions, in which case the DL reasoner infers that there is some missing knowledge that, if known, would make the new instance conforming to the axioms. Indeed, description logic systems naturally adhere to the OWA, which assumes that present data is just the explicitly known subset of the valid data, and more valid data may be inferred by reasoning. For instance, if axioms state that every hotel has a name, the creation of a new hotel is accepted even if no name is attached to the new hotel.

On the contrary, databases follow the CWA, stating that only information that is present in the database (or derivable by explicitly defined derivation rules) is valid. If a fact is not in the database, the fact is considered false. As a consequence, the creation of new instances has to obey all integrity constraints that apply to the instance. For instance, if the schema prescribes that every hotel instance must hold a value for the hotel name, the creation of a new hotel without specifying its name is not accepted.

It is uneasy to evaluate which approach is better. In fact, each one is best suited for the purpose it has been designed for. DL and its OWA fit well within an environment where the ontology is incrementally defined, which corresponds to a situation such that at each stage the current ontology only holds part of the world of interest, hence there are many more specifications that are relevant but not yet entered into the ontology. They also fit well with the idea that ontologies evolve as a result of collaborative design, where many independent partners can contribute new specifications to the ontology. The OWA also allows DLs to naturally support incomplete information at the instance level. Inference mechanisms handle case reasoning.

The database approach only offers a partial solution for managing incomplete information, the NULL value, which has no clear semantics and is uneasy to handle. On the other hand, the database approach and its CWA fit well in normative environments, where the ontology has to interact with an information system which assumes that the data it uses comes in a given format and is consistent with the application rules that have been stated in the ontology. In database management, satisfiability issues can be discarded and decidability issues do not arise. Consequently, database systems simply do not need sophisticated reasoners to infer additional information.

Another difference between DLs and databases is that databases rely on the unique name assumption, which assumes that each instance has its own identity, different from all others. In most DLs, unless explicitly stated by the user, nothing prohibits two instances to be the same one. The logic reasoner may infer that two instances, for example h1 and h2, describing two hotels are, in fact, the same one.

In terms of querying the ontology and its instances, databases and description logics offer complementary functionality for instance querying. Database systems usually provide powerful assertional query languages, complemented with efficient query optimization tools. Description logic systems support a set of simple functions for accessing instances that were directly inserted into the Abox (instance set) or are

inferred by the reasoning engine. Simply stated, the difference is that databases have been purposely designed to store, manage, and query huge volumes of data instances, while DL approaches have typically been targeted at sophisticated reasoning over a relatively small volume of instances. Similarly, database systems can easily handle value domains (the embedded ones as well as user-defined domains) while description logics experience quite a difficulty in fully handling concrete domains (each concrete domain calls for a careful extension of the reasoning capabilities).

4.4 Beyond Data Structures

Part of the semantics of the real world comes from where things are located in space and time. Traditional modeling approaches (in DL as in conceptual modeling approaches) ignore these components, assuming that the real world of interest is now and here. On the contrary, there are a huge number of applications where spatio-temporal information is essential. Considering our hotel example, spatial information could be used to convey the actual geographical location of hotels, cities and countries. This would enable queries such as "find hotels within 10 miles of a given city". Similarly, room rates are a typical example of information that is valid only within a given time period. In the current description, this is captured using the attributes fromDate and toDate. However, this is a poor solution in the sense that only the user is aware of the temporal semantics of these attributes. From the system viewpoint, these are two "normal" attributes, with a Date domain. No temporal reasoning and no temporal operators (in the sense developed by research in temporal databases) will be deployed by the system on such data.

There has been quite an investment in the DL community to develop temporal extensions of DL languages. There have been only few efforts to similarly develop spatial extensions. Spatio-temporal DLs still are a research item for the future.

The picture is comparatively better in conceptual modeling, where several proposals for spatio-temporal conceptual models exist today, and there is a pretty good understanding of what are the required functionalities. Proposals exist to cover spatio-temporal phenomena, such as e.g. mobile objects and trajectories [9][11], and multi-representation (to support context-dependent information) [12]. Moreover, Geographic Information Systems routinely and efficiently implement all the logical level constructs needed for the description and management of geographic data, including the two views of space: the discrete (or object-based) view and the continuous (or field-based) view.

5 Conclusion

Conceptual modeling and the database approach provide better readability/ understandability of the content of an ontology, and more efficient management for large ontologies and associated knowledge bases. DL approaches provide better reasoning capabilities and new knowledge inference from explicitly defined

knowledge. We therefore suggest that, rather than extending either formalism to try to cover all desirable functionality, a hybrid system, where the database component and the logic component would cooperate, each one performing the tasks for which it is best suited, might be the most promising solution for semantically rich information management, in particular semantic web information services. It seems obvious to us that, for instance, ontology description services should rely on conceptual data models, while ontological consistency services and incomplete information handling should be performed using description logics reasoners.

References

1. A. Borgida, R.J. Brachman: Conceptual Modelling with Description Logics. In Description Logic Handbook, F. Baader, D. Calvanese, D. McGuinness, D. Nardi, D., P. Patel-Schneider (Eds.), Cambridge University Press, 349–372, 2002
2. D. Calvanese, M. Lenzerini, D. Nardi: Description logics for conceptual data modeling, in Logics for Databases and Information Systems, J. Chomicki and G. Saake (Eds.), Kluwer Academic Publisher, 1998, 229-264
3. N. Cullot, C. Parent, S. Spaccapietra, C. Vangenot: Ontologies: A contribution to the DL/DB debate, in Proceedings of the VLDB Workshop on Semantic Web and Databases, Berlin, September 2003
4. F.M. Donini, M. Lenzerini, D. Nardi, W. Nutt, A. Schaerf: An epistemic operator for description logics, Artificial Intelligence 100, 1998, 225-274
5. D. Fensel, J. Hendler, H. Liebermann, W. Wahlster: Spinning the semantic web, The MIT Press, Cambridge, Massachusetts (2003)
6. M. Jarrar. J. Demey, R. Meersman: On Using Conceptual Data Modeling for Ontology Engineering, in Journal of Data Semantics 1, K. Aberer, S. March, S. Spaccapietra (Eds.), LNCS 2800, Springer, 2003
7. M. Klein, J. Broekstra, D. Fensel, F. van Harmelen, I. Horrocks: Ontologies and schema languages on the Web. In Spinning the Semantic Web, D. Fensel & al. (Eds.), The MIT Press, Cambridge, Massachusetts (2003)
8. R. Meersman: Ontologies and Databases: More than a Fleeting Resemblance. In: OES/SEO Workshop Rome, 2001
9. C. Parent, S. Spaccapietra, E. Zimanyi: Spatio-Temporal Conceptual Models: Data Structures + Space + Time, 7th ACM Symposium on Advances in Geographic Information Systems, ACM GIS'99, November 5th-6th, 1999
10. C. Parent, S. Spaccapietra: Database Integration: the key to data interoperability, in Object-Oriented Data Modeling, M. Papazoglou & al. (Eds.), MIT Press, 2000
11. S. Ram, R.T. Snodgrass, V. Khatri, Y. Hwang, DISTIL: A Design Support Environment for Conceptual Modeling of Spatio-temporal Requirements. In Proceedings of the 20th International Conference on Conceptual Modeling, ER 2001, Yokohama, Japan, LNCS 2224, Springer-Verlag, 2001, 70-83
12. C. Vangenot, C. Parent, S. Spaccapietra: Modeling and Manipulating Multiple Representations of Spatial Data, International Conference on Spatial Data Handling, Ottawa, Canada, July 9-11, 2002