A FLEXIBLE SUPPORT OF NON CANONICAL CONCEPTS IN ONTOLOGY-BASED DATABASES

Youness Bazhar[1], Yamine Aït-Ameur[2], Stéphane Jean[1] and Mickaël Baron[1]
[1]*LIAS - ISAE ENSMA and University of Poitiers*
Futuroscope, France.
[2]*IRIT - ENSEEIHT*
Toulouse, France.
{*bazhary, jean, baron*}*@ensma.fr, yamine@enseeiht.fr*

Keywords: Ontology, Database, Ontology-Based Database, OWL, Behavioral, Structural and Descriptive Semantics, Canonical and Non Canonical Concepts.

Abstract: Several ontologies are currently defined for various domains and for a wide range of applications. As a consequence, two problems shall be addressed (1) the size of data described by ontologies is huge and (2) existing ontologies are not all similar: they can be defined with different formalisms leading to different types of ontologies (canonical, non canonical and linguistic ontologies). The first problem has been solved by using databases to store data and ontologies that define the semantics of data. However, these databases, called Ontology-Based Databases (OBDB), do not handle the second problem i.e., that different types of ontologies exist. Indeed, they usually support a given ontology formalism without providing any mean to handle ontologies defined in an other formalism. In this paper, we propose an extension of the OntoDB OBDB to support the introduction of new operators that could be exploited by its exploitation language OntoQL. Such operators can be implemented by an external program located outside the database, or could invoke a web service. We show that these operators can be used to introduce new ontology constructors in order to support a given ontology model inside OntoDB. As a running example, we consider the support of the OWL non canonical constructors which are not supported in the native OntoDB OBDB.

1 INTRODUCTION

Since its introduction, the notion of ontologies has been widely used and has regained a lot of interests with the recent development of the Semantic Web. Indeed, ontologies are defined in a lot of domains such as engineering, medicine, biology or chemistry and set up for a wide range of applications like natural language processing, information retrieval, electronic commerce, software component specification or information systems integration etc. The intensive use of ontologies in a large number of domains leads to two main difficulties. (1) The amount of data described by ontologies can be huge, especially in domains like as E-commerce, engineering or Semantic Web. (2) Different ontology models exist to define different types of ontologies. For example, a lot of ontologies in engineering are defined with the PLIB ontology model (Pierra, 2007), (Pierra and Sardet, 2010) whereas most ontologies in the Semantic Web are defined with models such as RDF-Schema (Brickley and Guha, 2004) or OWL (Dean and Schreiber, 2004). The first difficulty has been overcome by the introduction of a new type of databases that store both data and the ontologies that define the meaning of these data. A lot of such databases, called Ontology-Based Databases (OBDB), have been proposed in the literature (e.g., *Oracle* (Chong et al., 2005), *RStar* (Lu et al., 2007)). If these OBDBs provide scalable storage systems for ontologies and its associated instances, they are usually defined for a given ontology model without any extension capability. Thus they are not able to address the second issue i.e., the wide diversity of existing ontologies. Our claim is that solutions can be provided to handle this problem in a flexible manner if meta-modeling capabilities are offered by the OBDB. In this paper, we consider the OntoDB OBDB (Dehainsala et al., 2007). This OBDB has originally been defined for the PLIB ontology model and thus it supports basic ontology con-

structors of classes and properties. However it does not support the definition of *non canonical concepts* (NCC) (derived or defined concepts) i.e. concepts defined by a complete axiomatic definition expressed in terms of other concepts (Gruber, 1995). This kind of concepts are particularly important in the OWL ontology model. For example, non canonical classes can be defined as a union of other classes or as a restriction on a property value. In this context, our proposition consists in extending OntoDB to support the introduction of new operators that could be exploited in order to introduce non canonical constructors. These operators are flexible since they can be implemented by an external program located outside the database, or could invoke a web service. As a proof of concepts, we show how the non canonical constructors of OWL can be defined and implemented on OntoDB. The remainder of this paper is organized as follows. Section 2 reviews some related work to our issue. Section 3 presents the OntoDB OBDB together with its exploitation language, OntoQL, on which our approach is based. Then, Section 4 exposes how we have extended the OntoDB/OntoQL system in order to support the non canonical constructors of OWL. Section 5 presents the implementation of our approach on OntoDB. Finally, Section 6 concludes the paper.

2 Related Work

In the last years, many OBDB architectures have been proposed to store ontologies and instance data. We present these architectures according to the number of schemes used to store all information and describe the eventual extension capabilities provided. **Type 1 OBDBs** are used to store RDF data that may contain ontology descriptions together with their instances. Main RDF-OBDBs are Oracle (Chong et al., 2005) or Sward (Petrini and Risch, 2007). These OBDB follows the simple model of RDF to store data. Indeed, a single schema composed of a unique triple table `(subject, predicate, object)` is used to store both ontology descriptions and instance data. As RDF data may include RDFS or OWL ontology descriptions, most of these OBDBs provide a support for the semantics of RDFS or OWL. This semantics is usually hard coded using deductive rules (Chong et al., 2005) or with external reasoners (Harris and Gibbins, 2003).

Type 2 OBDBs store separately ontology descriptions and instance data in two different schemes. Main examples are RStar (Lu et al., 2007) or OntoMS (Park et al., 2007). The schema for ontology descriptions depends upon the ontology model used to represent ontologies (e.g., RDFS, OWL, PLIB). It is composed of tables used to store each ontology modeling primitive such as classes, properties and subsumption relationships. For instance data, different schemes have been proposed. They have different scalability characteristics. These OBDBs mainly support the usual subsumption semantics as specified in the RDFS semantics (Hayes, 2004) (i.e, `subClassOf` and `instanceOf` relationships). They use different database mechanisms like views (Pan and Heflin, 2003), labeling schemes (Park et al., 2007) or the subtable relationships issued from object-relational databases (Alexaki et al., 2001; Broekstra et al., 2002). Some OBDBs address more complex reasoning using logic-based engines (e.g. Datalog engine) of deductive databases or OWL reasoners (Mei et al., 2006; Volz et al., 2005; Borgida and Brachman, 1993; Pan and Heflin, 2003).

Type 3 OBDBs. OntoDB (Dehainsala et al., 2007) proposes to add another schema to type 2 OBDBs. This schema called *meta-schema* records the ontology model. For the ontology schema, the meta-schema plays a role similar to the one played by the system catalog in traditional databases. If this part may allow the support of evolution of the used ontology, we show in next section that the behavioral semantics of the added constructors can not be defined.

Synthesis. As we have seen in this section, studies on OBDBs have been mainly focused on the scalability of these new types of databases. Considering support of ontology, each OBDB supports the semantics of a given ontology model using hard coded techniques either by using database mechanisms or by relying on an external logical engine. The aim of our work is to provide a more flexible approach that can be followed to support the semantics of several ontology models in different ways. As stated previously, OntoDB provides an interesting part, the meta-schema, to support the evolution of the used ontology model. Thus, our approach suggests to exploit this schema for encoding our proposed extensions. We present this architecture in detail in the next section and show its current limitations.

3 The OntoDB/OntoQL system

In next subsections we first present the OntoDB (Dehainsala et al., 2007) architecture, that supports the storage of different ontology models, ontologies and their instances in the same database, together with its associated language OntoQL (Jean et al., 2006). Then, we show through a motivating example the lim-

its of this OBDB architecture and of the OntoQL language to support the definition of the behavioral semantics of OWL ontologies NCCs.

3.1 The OntoDB architecture

OntoDB is an operational OBDB whose architecture is divided into 4 parts according to figure 1. The *Meta-base* and *Instances* parts are the traditional parts available in all database management systems (DBMS). The first one, namely the system catalog, contains the system tables used to manage all the data contained in the database. The second one contains the data that represents ontologies instances. The *Meta-schema* part holds ontology models (e.g. OWL meta-schema), and, finally, the *Ontologies* part contains ontologies conforming to the ontology models supported by OntoDB. In OntoDB, the whole data issued from different levels of information (meta-schema, ontologies and instances) is stored in relational tables thanks to the services offered by the *Meta-schema* part. As an example, let's consider Figure 1. In this example, three modeling levels are represented. The meta-schema level defines two entities: `Class` and `Property`. `Class` is the abstract description of one or many similar objects. Classes are organized in a hierarchy linked by an inheritance relationship (`superClass`). (`Property`) describes properties of a class. Ontologies level defines the `Person`, `Student` and `Professor` classes with various properties as instances of the meta-schema previously defined. Finally, the instances level defines instances of the `Student` and `Professor` classes defined in the ontologies level. Figure 2 shows how the data

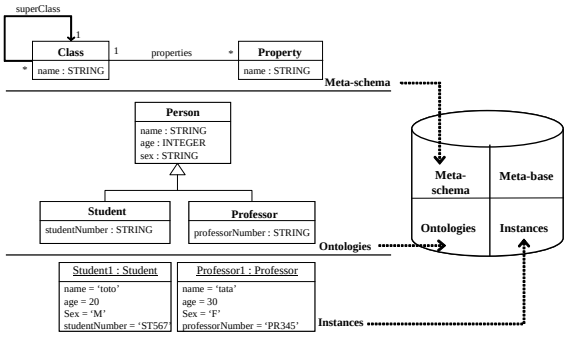

Figure 1: The storage of ontology models, ontologies and instances in OntoDB.

of the previous example are represented in OntoDB. In this simplified example, data related to the meta-schema part are stored in `Entity` and `Attribute` tables. Thus, the first table contains two rows for the two entities `Class` and `Property`, and the second table contains attributes of the defined entities. For each entity, a corresponding table exists at the ontology level. These tables contain instances of the different entities of the meta-schema part. In our example, it contains classes and properties. Finally, for each class a corresponding table is defined at the instance level to store its individuals.

3.2 The OntoQL language

Since the whole data of OntoDB is stored in a database, one can think that SQL could be used to manipulate them. In this case, users need to have a deep knowledge of the internal tables representation used by OntoDB. To overcome this limitation, OntoDB has been equipped with an exploitation language named OntoQL that hides the internal representations and directly manipulates ontology models and ontologies concepts. This language can be used to create new ontology models and instanciate them in order to create ontologies of different levels of information presented in Figure 1. The following OntoQL statements illustrate how the language proceeds:

```
CREATE ENTITY #Class (#name STRING,#supCls REF(#Class));
CREATE ENTITY #Property (#name STRING,#cls REF(#Class));
CREATE #Class Person (name STRING, age INT, sex STRING);
CREATE #Class Student UNDER Person (stdtNumb STRING);
CREATE #Class Professor UNDER Person (ProfNumb STRING);
INSERT INTO Professor VALUES ('tata', 30, 'F', 'PR345');
INSERT INTO Student VALUES ('toto', 20, 'M', 'ST567');
```

The first two statements define the meta-schema. The entity `Class` is defined with a `name` and the class it extends, the `supCls`. The entity `Property` has also a `name` and is linked to the class it describes (`cls`). In these statements, the names of entities and descriptions of entities are prefixed by the character `#`. Indeed, the meta-schema part of OntoDB can be extended and modified and thus this level of information cannot be encoded as keywords of the OntoQL language. The next three statements define the different classes of our example with their properties using the `CREATE #Class` clause. A class is defined as a subclass of an other one using the keyword `UNDER`. Finally, the last two statements define instances of our classes using an `INSERT INTO` syntax close to the one of SQL.

3.3 Encoding OWL ontologies canonical concepts with OntoQL

As we have previously outlined, the aim of this paper is to show how OWL can be supported by the OntoDB/OntoQL system. In particular, we would

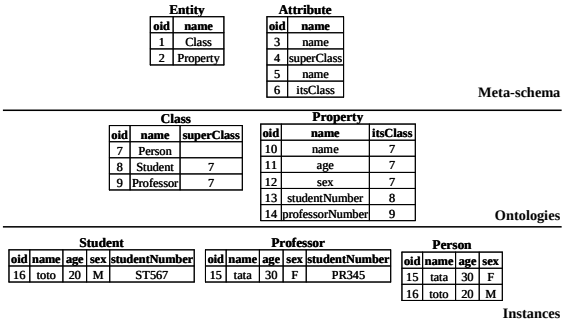

Figure 2: Meta-schema, Ontologies and instances data representation in OntoDB.

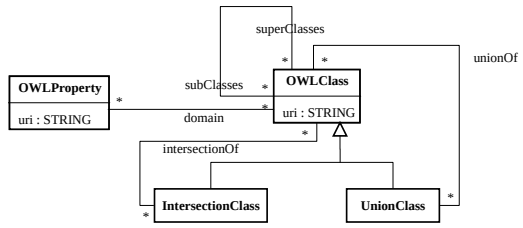

Figure 3: A Part of a simplified OWL meta-model

like to show how this system can manage the behavior of NCCs that are derived from other concepts (e.g. `UnionClass`, `IntersectionClass`, etc.). This subsection shows how the OWL canonical concepts of OWL are represented with OntoDB/OntoQL. We take a part of a simplified OWL meta-model which is presented in the figure 3. This meta-model contains two canonical concepts: `OWLClass` and `OWLProperty`. The other concepts of this meta-model are non-canonical. Indeed, they are derived concepts and have to be computed from other concepts. For instance, a `UnionClass` is an OWL class made from the union of a set of `OWLClass`. Thus, the `UnionClass` is a NCC. At this level, we are capable to encode the statements that support such concepts description. The statements to create and to store `OWLClass` and `OWLProperty` concepts are :

```
CREATE ENTITY #OWLClass (#uri STRING, #superClasses REF
(#OWLClass) ARRAY, #subClasses REF (#OWLClass) ARRAY);
CREATE ENTITY #OWLProperty (#uri STRING, #domain REF
(#OWLClass) ARRAY);
```

The `OWLClass` defines the concept of OWL class. It has an `uri`, a set of super classes (`superClasses`) and a set of subclasses (`subClasses`). The `OWLProperty` defines the concept of OWL property that has an `uri` and a `domain` (the classes which the property belongs to). These statements support the creation of structures (tables) that will permit the storage and the manipulation of OWL classes and properties in OntoDB using the OntoQL exploitation language.

3.4 Limitation of OntoDB/OntoQL

As shown in subsection 3.2, OntoQL can be used to add new entities in the meta-schema part of OntoDB. For example, let's consider the `UnionClass` concept. It supports the definition of a class as an union of other classes. Using this constructor, a class called `SchoolMember` could be defined as the union of the `Professor` and `Student` classes of our example. The OntoQL statement to extend the meta-schema with this new constructor is:

```
CREATE ENTITY #UnionClass UNDER #OWLClass (#unionOf
REF(#Class) ARRAY);
```

This OntoQL statement states that the structure of this constructor (i.e, an `UnionClass` has the same attributes of `OWLClass` and is defined by a set of classes). However, we are not capable to state within this statement nor with any other OntoQL statement, that the instances of an `UnionClass` can be computed as the union of the instances of the classes used in its definition *(Instances(C) = Instances(C1) U Instances(C2) U ... U Instances(Cn))*, and the `UnionClass` becomes a super class of the classes taking part of the union (*SubClasses(C)* = $\{C1, C2, ..., Cn\}$). As a consequence, we can assert that OntoQL does not support the behavioral semantics of the OWL NCCs (`UnionClass`, `IntersectionClass`, etc.). but, this semantics could be defined in operations such as functions, procedures defined or implemented *inside* (stored procedure) or *outside* (external program) the OntoDB database, or as web services. The former is already available in some database systems but the later is still not available. For example, we should be allowed to define the semantics of the *unionOf* operator through a statement such as :

```
CREATE #UnionClass C1 AS unionOf(C2, C3);
```

This statement should be able to modify the structure of C2 and C3 classes. Indeed, C1 becomes a super class of C2 and C3. Moreover, the OWL *unionOf* operator defines C1 individuals as the union of C2 and C3 individuals. Therefore, there is a need of an operator `unionOfInstances` which computes the union of C2 and C3 classes individuals in order to produce individuals belonging to C1. This operator should be used such as :

```
CREATE EXTENT OF C1 AS unionOfInstances(C2, C3);
```

The operators *unionOf* and *unionOf-Instances* could be implemented by internal or by external procedures or by web service invocation. Therefore, extending OntoQL to support web services, procedures, functions triggering and constraints checking will make it possible to offer the definition of the behavioral semantics of the

OWL ontologies NCCs. *Offering such a capability is detailed below*.

4 EXTENDING ONTODB/ONTOQL WITH BEHAVIORAL SEMANTICS

Our work focuses on extending the OntoDB platform with operations that can be dynamically defined by the user. These operations may be defined outside database and implemented with a programming language (e.g. Java). This extension is performed in two steps.

4.1 Extending the OntoDB meta-schema part

We Firstly have set up an application programming interface (API) which allows to invoke, from the OntoDB platform, external programs. Those programs are implemented with a programming language. Then, we extended the OntoDB meta-schema part with a new entity `EXTERNAL_PROGRAM` that stores all the information about the external program. Indeed, these information are the function name, the class name, the package name, and the location of the program that contains the operation to invoke. The OntoQL statement for extending the meta-schema part with the `EXTERNAL_PROGRAM` concept is :

```
CREATE ENTITY #EXTERNAL_PROGRAM (#FCT_NAME STRING,
#PACKAGE_NAME STRING, #CLASS_NAME STRING, #LOC STRING);
```

Thus, we can create external program and store data related to their accessibility in OntoDB. For instance, we can define an external program that implements the OWL `unionOf` operator and store information about this program in OntoDB as it is shown in the following statement :

```
CREATE #EXTERNAL_PROGRAM ('unionOf', 'fr.ensma.lisi.owl-
ncconcepts', 'NCOperators', 'D://owlncoperators.jar');
```

This statement stores the necessary information about the `unionOf` function implemented with a programming language and stored outside the OntoDB platform. It supports the execution of the union of OWL classes. Let's consider another operator `unionOfInstances` that computes the individuals belonging to the resulting class of the `unionOf` operation. Its definition is writting by :

```
CREATE #EXTERNAL_PROGRAM ('unionOfInstances', 'fr...owl-
ncconcepts', 'NCOperators', 'D://owlncoperators.jar');
```

This statement stores information about the program used to compute the individuals belonging to the resulting class of the OWL union classes.

4.2 Extending the OntoQL language

After extending OntoDB schemas, we allow OntoQL to take into account external functions invocation in the expression part. Thus, we are able to make external functions calls in OntoQL statements as it is shown by the following statement [1] :

```
CREATE #UnionClass SchoolMember ('schoolmember') AS
unionOf (Professor, Student);
CREATE EXTENT OF SchoolMember AS unionOfInstances
(Professor, Student);
```

5 SUPPORTING OWL NON CANONICAL CONCEPTS IN ONTODB

This section present the implementation of the `UnionClass` concept. The implementations of the other concepts are more or less similar to the one we process. We describe the modification that affects the ontologies elements and individuals. Then, we present examples of statements to express structural, descriptive and behavioral semantics of `UnionClass` concept.

`UnionClass` represents a class obtained from the union of a set of OWL classes. For instance, if we consider that the class C1 is the union of two classes C2 and C3 *(C1 = C2 U C3)*, individuals of C1 are the union of the individuals of C2 and C3 *(Instances(C1) = Instances(C2) U Instances (C3))*, and C1 become a super class of C2 and C3 : *SubClasses(C1) =* $\{C2, C3\}$. In order to show the complete process of defining a union class, we create two OWL classes (Professor and Student) and make their union.

```
CREATE #OWLClass Professor('professor');
CREATE #OWLClass Student('student'));
```

The statement for the union class is :

```
CREATE #UnionClass SchoolMember('schoolmember') AS
unionOf(Professor, Student);
```

This statement modifies the structure of `Professor` and `Student` classes. It adds the `SchoolMember` class to their super classes and sets `SchoolMember` subclasses to `Professor` and `Student`. The statement supporting the computation of the individuals of `SchoolMember` is :

```
CREATE #EXTENT OF SchoolMember AS
unionOfInstances(Professor, Student);
```

[1] In order to abbreviate, we use the classes names instead of classes Uris in all the following statements

6 CONCLUSION

In this paper, we have introduced our work on extending OBDBs offering meta-modeling capabilities together with their exploitation languages in order to support the representation of ontologies NCCs. Our objective was to express the behavioral semantics of these NCCs through operators dynamically defined inside or outside the OBDB. In this direction, we firstly raised lack of OBDB systems to express the behavioral semantics of NCCs, then we extended the OntoDB/OntoQL system in order to take into account the representation of ontologies NCCs by defining external programs (defined outside the database). These programs serve as operators, which can be invoked in OntoQL statements. Indeed, we have shown how we extended the meta-schema part of OntoDB in order to take into account operations invocations, and we extended the OntoQL exploitation language in order to make operations calls. Finally, as an implementation of our approach, we gave the representation of OWL ontology model in OntoDB and we proved the support of the OWL ontologies NCCs. As perspectives of our work, we expect to exploit meta-modeling capabilities of OntoDB/OntoQLsystem in order to perform model transformations and process web services orchestrations in database.

REFERENCES

Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., and Tolle, K. (2001). The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *Proceedings of the 2nd International Workshop on the Semantic Web*, pages 1–13.

Borgida, A. and Brachman, R. J. (1993). Loading data into description reasoners. *SIGMOD Record*, 22(2):217–226.

Brickley, D. and Guha, R. V. (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. World Wide Web Consortium. `http://www.w3.org/TR/rdf-schema`.

Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Horrocks, I. and Hendler, J., editors, *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, number 2342 in Lecture Notes in Computer Science, pages 54–68. Springer Verlag.

Chong, E. I., Das, S., Eadon, G., and Srinivasan, J. (2005). An Efficient SQL-based RDF Querying Scheme. In *Proceedings of the 31st international conference on Very Large Data Bases (VLDB'05)*, pages 1216–1227.

Dean, M. and Schreiber, G. (2004). *OWL Web Ontology Language Reference*. World Wide Web Consortium. `http://www.w3.org/TR/owl-ref`.

Dehainsala, H., Pierra, G., and Bellatreche, L. (2007). Ontodb: An ontology-based database for data intensive applications. In *Proc. of the 12th Int. Conf. on Database Systems for Advanced Applications (DASFAA'07). LNCS*. Springer.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies (IJHCS)*, 43(5-6):907–928.

Harris, S. and Gibbins, N. (2003). 3store: Efficient bulk RDF Storage. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PPP'03)*, pages 1–15.

Hayes, P. (2004). *RDF Semantics*. World Wide Web Consortium. `http://www.w3.org/TR/rdf-mt/`.

Jean, S., Aït-Ameur, Y., and Pierra, G. (2006). Querying Ontology Based Database Using OntoQL (an Ontology Query Language). In *Proceedings of On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences (ODBASE'06)*, volume 4275 of *Lecture Notes in Computer Science*, pages 704–721. Springer.

Lu, J., Ma, L., Zhang, L., Brunner, J.-S., Wang, C., Pan, Y., and Yu, Y. (2007). Sor: a practical system for ontology storage, reasoning and search. pages 1402–1405.

Mei, J., Ma, L., and Pan, Y. (2006). Ontology query answering on databases. In *Proceedings of the 5th International Semantic Web Conference (ISWC'06)*, pages 445–458.

Pan, Z. and Heflin, J. (2003). DLDB: Extending Relational Databases to Support Semantic Web Queries. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, pages 109–113.

Park, M. J., Lee, J. H., Lee, C. H., Lin, J., Serres, O., and Chung, C. W. (2007). An Efficient and Scalable Management of Ontology. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, volume 4443 of *Lecture Notes in Computer Science*, pages 975–980. Springer.

Petrini, J. and Risch, T. (2007). SWARD: Semantic Web Abridged Relational Databases. In *Proceedings of the 18th International Conference on Database and Expert Systems Applications (DEXA'07)*, pages 455–459.

Pierra, G. (2007). Context Representation in Domain Ontologies and its Use for Semantic Integration of Data. *Journal Of Data Semantics (JODS)*, X:34–43.

Pierra, G. and Sardet, E. (2010). *ISO 13584-32 Industrial automation systems and integration Parts library Part 32: Implementation resources: OntoML: Product ontology markup language*. ISO.

Volz, R., Staab, S., and Motik, B. (2005). Incrementally Maintaining Materializations of Ontologies Stored in Logic Databases. *Journal of Data Semantics II*, 3360:1–34.