Context Representation in Domain Ontologies and Its Use for Semantic Integration of Data

Guy Pierra

Laboratory of Applied Computer Science (LISI)
National Engineering School for Mechanics and Aerotechnics (ENSMA), Poitiers
86960 Futuroscope Cedex - France
pierra@ensma.fr

Abstract. The goal of this paper is to identify various aspects of context-awareness needed to facilitate semantics integration of data, and to discuss how this knowledge may be represented within ontologies. We first present a taxonomy of ontologies and we show how various kinds of ontologies may cooperate. Then, we compare ontologies and conceptual models. We claim that their main difference is the consensual nature of ontologies when conceptual models are specifically designed for one particular target system. Reaching consensus, in turn, needs specific models of which context dependency has been represented and minimized. We identify five principles for making ontologies less contextual than models and suitable for data integration and we show, as an example, how these principles have been implemented in the PLIB ontology model developed for industrial data integration. Finally, we suggest a road map for switching from conventional databases to ontology-based databases without waiting until standard ontologies are available in every domains.

1 Introduction

A number of computer science problems, including heterogeneous database integration, natural language processing, intelligent document retrieval would benefit from the capability to model the absolute meaning of things of a domain, independently of any particular use of them. In the structured-data universe, information is represented as data. Indeed, many studies have been performed to integrate heterogeneous and autonomous databases [8], in particular using domain ontologies [33]. Distributed architecture models have been developed, where mediators [34] provide uniform access to heterogeneous data sources. Mediators export integrated schemas that reconcile data both at the structural (schematic heterogeneity) and at the meaning level (semantic heterogeneity). If large progresses have been made to automate schema integration at the structural level, using in particular new model management techniques [3], the major challenge remains the automation of semantic integration of several heterogeneous schemas. Such an automation would need enabling programs to identify unambiguously:

S. Spaccapietra (Ed.): Journal on Data Semantics X, LNCS 4900, pp. 174–211, 2008.

- those data having exactly the same semantic meaning,
- those data that are similar and may be converted in or compared with each others by given processes, and
- those data having no semantic commonality.

In the above list, data may be either atomic or structured data like tuple or entity instance. On the Internet, data is not the only means for representing information, another one is largely used, namely documents. Large progresses were achieved by search engines to retrieve over the Internet the most relevant documents with respect to a user query expressed as a sentence of words. Unfortunately, if semantic of both the query and target documents are not made computer-sensible, it is impossible to retrieve documents dealing with the query subject without using exactly the same words (e.g., worker instead of employee, size instead of length or convertible instead of car). Here again some kinds of computer interpretable representation of word meaning is needed:

- in a first step to improve search engine in order to retrieve which documents are semantically relevant for a topic defined by a set of words, even when the same words are not used, and
- in a second step, to retrieve which documents might provide answer to a user query.

Both kinds of information integration requiring explicit representation of meaning, these last ten years a lot of work has been done to develop domain ontology models[1] intended to capture the a priori nature of reality of some Universe of Discourse (UoD), as independently as possible from any particular use of this reality. Once defined, such representations may then be used to reconcile various information sources addressing this UoD at the meaning level.

The word ontology is now extensively used in a number of computer science domains, including e.g., knowledge management, natural language processing, database, object oriented modeling. If there seems to be some consensus on what an ontology structure should be - categories (classes), properties, logical relationships - the focus of the various approaches is so different that the same word seems to represent quite different realities, and that ontologies developed, e.g., for natural language processing seems to be nearly useless for e.g., database integration, and conversely.

The goal of this paper is twofold. The first goal is to investigate the concept of an ontology in a structured-data integration perspective. We claim that the major difference between ontologies and conceptual models is the existence of a consensus that founds ontologies as a shared meaning, and that consensus, in turn, needs representation of the modeling context. As an example, it is easy to reach consensus on the fact that the *resistance* is an essential property of a *resistor*. But the *resistance* depends upon the *temperature* where the *resistance* is measured, and it is nearly impossible to reach consensus on the *temperature*

[1] In this paper, we only consider domain ontologies, and not upper-level generic ontologies. Thus, for short, ontology means domain ontology.

where the *resistance* should be measured since this temperature depends upon the resistor target usage. To include in the ontology model a mechanism allowing to represent with each *resistance* value the *temperature* where it was evaluated (value-evaluation context awareness) enables consensus since it allows each user category to select its own *resistance* measuring process while making explicit commonality and differences. Thus, we propose five principles allowing to make ontologies much more generic through context representation, and thus more suitable for heterogeneous data integration. The second goal is to show how these principles have been implemented in the ontology model we have developed over the last 15 years. Discussed within an international standardization project (see Annex A), the PLIB ontology model (officially ISO 13584), was initially developed for giving meaning to technical data and for providing for automatic integration of heterogeneous engineering data sources. The overview of PLIB presented in this paper allows both to illustrate the context representation principles that we propose, and to show some typical uses of this ontology model.

The content of this paper is as follows. In the next section, we discuss the various kinds of ontologies needed for representing semantics. We distinguish between document-oriented linguistic ontology (LO) and information-modeling-oriented conceptual ontology (CO). In section 3, we investigate the differences between ontologies and models, and we propose mechanisms to represent both modeling and value context within an ontology. In section 4, we outline how PLIB ontologies are specified (and exchanged) thanks to an executable specification defined in the EXPRESS data specification language, and we present how context-awareness mechanisms are represented in the PLIB ontology model. In section 5, we present a formal model of PLIB ontologies, including the mapping capabilities to external ontologies, and we outline how such ontologies may be used to integrate various heterogeneous data bases. We suggest, in section 6, a road map for switching from conventional databases to ontology-based databases. A discussion of related work regarding context representation, ontology models and data integration is presented in section 7. Conclusion is presented in section 8. Some standards being quoted in the paper, Annex A outlines the standardization activities around PLIB.

2 Concept Ontologies and Linguistic Ontologies

Since the term ontology was borrowed from philosophy and introduced in the computer science vocabulary, many definitions have been offered. The most commonly cited definition is one by T. Gruber "An ontology is an explicit specification of a conceptualization", therefore "shared ontologies" provide for "knowledge sharing" [15]. In all the ontology models, such a conceptualization consists of three parts :

– *primitive items* of the ontology (where items are either classes or properties) are those items "for which we are not able to give a complete axiomatic

definition; we must rely on textual documentation and a background of knowledge shared with the reader" [15],
- *defined items* are those items for which the ontology provides a complete axiomatic definition by means of necessary and sufficient conditions, and
- *logical relationships* (or *inference rules*) provide for reasoning over ontology items, and for supporting some problem-solving activities over the ontology domain.

The agreed definition and structure description leave open what may be considered as the major criteria for classifying ontologies and ontology models: whether their area of interest consists of beings (what does exist in the world) or of words (how beings are apprehended and reflected in a particular natural language).

We call linguistic ontologies (LO) those ontologies whose goal is to represent the meaning of the words used in a particular UoD in a particular language. We call concept ontology (CO) those ontologies whose goal is to represent the categories of objects and object properties that are used to apprehend some part of the world. These two kinds of ontologies address quite different problems and should have quite different contents.

LOs [9] are document-oriented. The typical problem they address may be termed as follows:
"find all documents relevant to a query expressed as a set of words possibly connected by logical operators like AND, OR and NOT, even if these documents do not contain these words".

Since natural languages contain a number of different words for reflecting identical or similar meanings, LOs are large in nature. They include a number of *conservative definitions*, i.e., defined items that only introduce terminology and do not add any knowledge about the world [15]. They are language-specific and they use a number of linguistic relationships such that *synonym, hypernym, hyponym, overlap, covering, disjoint* to capture in a semi-formal way meaning relations [33]. Such relationships being not formally grounded, inference could only provide some help to a user supposed to be involved in some computer-aided search process. Development of LOs may be done through a semi-automatic process where significant words are automatically extracted from a document collection and then validated and structured by experts.

COs, for instance the measure ontology [15], are information-modeling-oriented. The typical problem they address may be termed as follows:

"decide whether two instances belong to the same class and whether two properties have identical meaning or may be converted into each other".

To be able to represent all the beings existing in some part of the world, COs need only to describe those primitive concepts that cannot be derived from other concepts. Like technical vocabulary where one and only one word should always be used for the same meaning, COs may be restricted to primitive concepts. Such primitive COs, that we call *canonical conceptual ontologies* (CCOs), are compact in nature. To reduce again the number of concepts that need to be represented, COs may also be property-oriented. This means that in place of

introducing a number of different concepts such that "10-HP-engine","20-HP-engine","25-HP-engine", they introduce only two concepts that may express the same meaning: one class (engine) and one integer-valued property (power in HP). Indeed, only those classes that cannot be represented by restriction of an existing class by means of property values need to belong to a property-oriented CCO. The focus being on primitive concepts, and understanding such concepts being based on reader background knowledge, an extensive information model has to be used to describe both textually and formally each primitive concept. COs are multilingual because most concepts are language-independent. Even though a collection of documents in one particular language of which significant words are automatically extracted may be used as a starting point for defining a CO of some domain, development of a CO is mainly manual. Finally, if the relationships involved in a CO are formally defined, and if two data sources reference the same CO, semantic integration of these data sources may be done automatically [2].

Table 1. Typical characteristics of LO and CO

	LO	**CO**
Tokens	Word	Concept
Token identification	Word	Id
Token definition	Sentence	Model
Ontology Size	Extensive	Minimal
Relations	Formal + Linguistic	Formal
Content	Primitive Items + Conservative Definitions	Primitive Items (CCO) + Conservative Definitions (NCCO)
Focus	Class-oriented	Property-oriented
Development	Automatic/Semi-automatic	Semi-automatic/Manual
Ontology usage	Computer-aided tasks	Task automation

When the goal of a CO is to define a common language for data exchange or data integration, CCOs are well suited. For data integration, the use of CCOs assumes that each source or agent is in charge of converting its own vocabulary onto the shared CCO. It is the approach followed by the PLIB ontology model developed to support exchange of industrial data. In PLIB ontologies, equivalences between ontology concepts are not represented within the PLIB ontology but as an external mapping between ontologies, called ontology articulations (see section 5.2). Example 1 presents an informal description of a small CCO that represents some categories of industrial components[2].

Example 1. A circular bearing is a mechanical component used to connect and to transmit load between two cylindrical shapes having the same axis but different diameters and rotational movements. Characteristic properties include *width*, *inner diameter* and *outer diameter*. But a crucial property, called *life-time*, is

[2] Such an ontology is formally defined in ISO 23768 using the PLIB ontology model.

the length of the time period where the bearing will behave correctly. The value of this property depends upon the number of revolutions done by the bearing, and of the load it must support. Mathematically, the bearing *life-time* is a function of the *velocity* (i.e. rotational speed), the *radical load* and the *axial load*. At the class level, *circular bearings* may be *circular ball bearings* (there are also other kinds of bearings, not modeled within this small ontology, having, e.g., needles or rollers). The *ball diameter* is a property that should be defined at the level of *circular ball bearing* where it is meaningful. Figure 1 presents the main properties of a *circular ball bearing*.

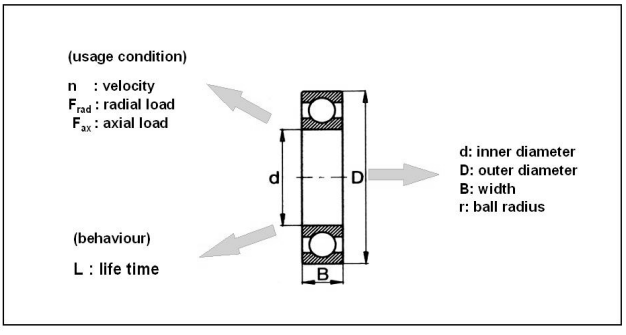

Fig. 1. Characterization of a bearing

Equivalence of concepts may also be represented within a CO. This may be done for instance using either formal class relationships like set-oriented operations (union, intersection and difference) and class restrictions (by property values), as it is done in OWL [20], or property value deduction, as it is done by F-logic rules, or property value algebraic derivation function, like in the EXPRESS language [30]. For example, the *thickness* of the bearing may be defined as (*outer diameter* − *inner diameter*)/2. Such COs, that we call *non-canonical conceptual ontologies* (NCCO), allow to integrate in the same ontology different conceptualizations and the articulations between them. NCCOs are in particular largely used in artificial intelligence. They allow to make inference over concept equivalence, but they often encounter scaling problems for processing large data sets.

Table 1 emphasizes the main differences between COs and LOs. But, in fact, LO, NCCO and CCO are complementary, and, in a number of ontology-based applications, all three kinds of constructs are needed over the same domain. It is the case, for instance, when a domain ontology is built using natural language processing (NLP) tools that extract terms from a document corpus. This set of terms is progressively structured by domain experts into a LO that contains both formal relationships, such that subsumption and class-property links, and linguistic relationships such that homonymy or synonymy. In this ontology, entries are still words of a particular language. From this LO, a NCCO may be extracted under expert supervision. In this ontology, entries become identifiers,

and relationships are based on a clear mathematical semantics. Finally, a CCO may be chosen from the NCCO and all non-canonical concepts are formally defined in terms of canonical concepts. Note that this process is quite similar to the one defined in [24]. Similarly, all three kinds of constructs are also needed when one wants indexing a set of documents by means of concepts of a CO, either CCO or NCCO. If the starting ontology is a CCO, a NCCO needs first to be defined to address all the concepts usual in the CCO domain, even when some of them might be represented by some other ones. Then, a LO must be developed on top of this NCCO. This LO must include all the language-specific terms, and term patterns, that may be considered as representing each particular concept or particular property. Then, these terms may be used for indexing a set of documents, either automatically or under expert supervision. These two possible approaches for building domain ontology suggest a layered view of domain ontologies [18] in which CCOs, NCCOs and LOs may cooperate. In this view, a kernel CCO defines all the UoD semantics. We call this layer the *characterization layer*. Thanks to this CCO, any object belonging to the UoD may be characterized by class belonging and property-value pairs, thus providing a canonical language for information exchange. At the second layer, that we call *integration layer*, a NCCO extends this conceptual vocabulary by means of conservative definitions to encompass all concepts broadly used in the domain. Using the ontology defined according to this layer, several data sources addressing the same domain but based on different CCOs may be integrated, and inference may be performed. At the third layer, that we call *discourse layer*, a LO provides a multilingual natural language interface for person-system and person-person communication. Figure 1 show the resulting onion-shaped model that we call the Characterization-Integration-Discourse CID model of domain ontology.

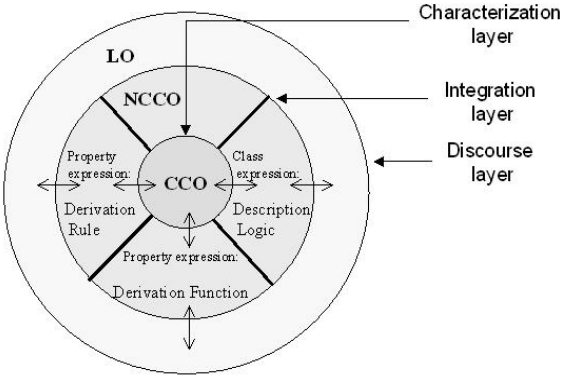

Fig. 2. The CID layered model of domain ontologies

Example 2. A CCO for characterizing (to some extent) persons might define the class *person* with two properties: *gender* and *age*. If one wants to cover more broadly the same domain, the two classes *man* and *woman* need to be introduced in a NCCO. *Man* for instance being defined as a *person* whose *gender= male*. Migrating to LO would need, beside the terms used as labels for the NCCO and that are formally grounded on this NCCO, other terms such that *children*, *boy*, *girl*, *oldster* that could not be formally defined.

Data integration being mainly concerned with CO, in the next section we discuss the differences between CO and conceptual models.

3 Concept Ontologies Versus Model

In the previous section we have discussed the differences between the various kinds of domain ontologies and we have proposed a model representing how they may fit and cooperate altogether. In this section we propose to clarify the differences between domain ontologies and conceptual models. Indeed, a conceptual model may be considered as an "explicit specification of a conceptualization". Therefore, as noticed by Guarino and Welty [16], conceptual models are sometimes denoted as ontologies. But we perfectly know that conceptual modeling leaves open the semantic heterogeneity problem. Thus, it is worthwhile investigating the difference between a shared ontology and a model if we want to use ontology as a tool for semantic integration of data.

An old definition from Minsky [22] would introduce the discussion: "To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A". This definition emphasizes the ternary character of a model relationship: it depends on the object (A) and the observer (B), but it depends also on which questions the observer is interested about A. In other terms, in which context the model was built. In data engineering, we are in line with this definition when we teach that a conceptual model shall be built within a precise context. The key point here is that when one designs an application system, the context of the modeling activity is defined by the target system goals and environment. The functions that two systems must perform are never exactly the same. Thus models are always slightly different, enough to make instance data incompatible.

This shows that usual conceptual models can hardly fit several needs. If one wants to build shared ontologies, i.e., ontologies that reflect the information requirements of several application contexts, not only the conceptualization approach must be different from usual modeling activity directed toward a specific target system, but also the conceptualization formalism must have specific capabilities to allow specification of generic models. These generic models must be either context-independent or at least context explicit to fit the needs of various application contexts.

Importance of context representation for semantic integration of heterogeneous database was already underlined by researchers in multidatabase systems. Kashyap et al. [19] proposed an explicit representation of the modeling context

at the schema definition level. For instance, what is the meaning of the "width" property when we try to use it with a "car engine" without knowing in the context of which class and with which precise meaning this property was described?

The property becomes clear when we know that it was defined in a *packaging perspective* for any *material object* as the *width of the virtual box* where it might be packaged.

But even if a property definition is clearly understood, property value may also be context dependent, such context-sensitivity was studied in particular by [31], [14]. These authors proposed to represent context at the extensional level, i.e., at the level of data values and object instances. For instance, what means the temperature of a particular city if we do not know when this temperature was measured, and in which unit? What means the life time of a bearing if we do not know which load it supports and what would be its rotational speed?

In fact most of the causes of semantic conflicts in data integration result from implicit context, either in schema definition or in value evaluation. They may be solved if both the *modeling context* and the *value context* are made explicit. Goh [14] identified three main causes for semantic heterogeneity:

1. *naming conflicts* occur when naming schemas of information differ significantly. A frequent phenomenon is the presence of homonyms and synonyms. We claim that naming conflicts may be avoided if data base schemas refer explicitly for all the shared concepts they represent to identifiers of a shared conceptual ontology, and if this shared ontology makes explicit the context of each definition. *Driving license id* is unambiguous if it is defined in the context of a *French car drivers* class, it becomes ambiguous (and may have several values) in a context of a *person*.
2. *scaling conflicts* occur when different reference systems are used to measure the value of some properties. Examples are different currencies. Scaling conflict may be avoided, either by associating explicitly at the schema level a computer-interpretable representation of the unit that shall be used for any value of a property, or by associating explicitly with each value its own unit.
3. *confounding conflicts* occur when information items seem to have the same meaning, but differ in reality, e.g., due to different temporal contexts. We claim that confounding conflicts may be avoided by investigating whether a value is an intrinsic and permanent property of some instance, or it depends on some evaluation context, and, in the latter case, by associating this value with its context. For instance the *driving license id* of a person depends on the *country* where the license was issued, its *weight* depends on the *date* where it was recorded, but its *birth date* is not context dependent (once the scaling conflicts is solved as above).

Moreover, most causes of schematic conflicts, and in particular schema isomorphism conflicts which means that semantically similar entities have a different number of attributes [19] also result from context sensitivity. It is not so difficult to identify, to describe and to reach consensus in some community on all the major properties which are rigid [16], i.e. which necessarily hold for all instances

of a class. For instance, each *customer* has a *birth date*, each *mechanical component* has a *weight*, and each *town* has a (current) *number of inhabitants*. But it is impossible to agree on those rigid properties that should be represented for each class in a database. Thus, ontological description of a class should describe all its rigid properties (at least within some rather broad context common to all data sources that might exist in some target community) in order to reduce context-dependency in the class description. Then, each schema may restrict this general description to its design context by selecting those ontology-defined properties that are relevant to the problem at hand and are thus represented in the database. For instance, the *weight* or *birth date* of a *person* are seldom used in a customer database. So, when several schemas refer to a same ontology by means explicit mappings [2], these mappings allow to identify automatically which ontology-defined properties are semantically equivalent in several data sources, which properties are represented in some data sources without being represented in some others, and, possibly, which properties if any are not defined in the common ontology.

This discussion allows to define five principles that should be followed by ontology models to provide for automatic integration of several data sources. It also suggests five mechanisms that may be proposed for satisfying each principle.

- *Definition context representation.* At the schema level the modeling context in which each class or property is defined should be explicitly represented and minimized.
  Proposed mechanism: to represent its definition context, each property should be defined in the context of a class that defines its domain of application. To minimize its context-sensitivity, each class should define all its rigid properties, at least in some very broad context common to all the target data sources.
- *Point of view representation.* The perspective adopted by the modeling team when the ontology is designed and agreed upon in some community should be explicitly represented.
  Proposed mechanism: if several perspectives are needed over the ontology target domain, an ontology of perspectives should be defined or referenced. Then, each needed perspective should correspond to a specific domain ontology. Different perspectives over the same real world object should be represented either by an instance aggregate, one instance per perspective-specific ontology, or by multi-instantiation.
- *Locality of interpretation context.* Importation of resources from one ontology into another one should be possible while controlling the impact of the former over the interpretation of the latter.
  Proposed mechanism: importation on a class per class basis, and, for a class, on a property per property basis should be feasible. Domains of both ontologies should be separated.
- *Value context representation.* At the value level, the local context in which each value is evaluated should be explicit.
  Proposed mechanism: when the property value of some ontology class instance depends upon some evaluation context, this evaluation context should

be modeled by properties defined over this evaluation context, and the former property should be modeled as a function over the latter properties.

– *Value scaling representation* When the same property magnitude may be represented by different values depending upon some scaling process, scale should be explicitly represented, either at the ontology level, or at the instance level.
Proposed mechanism: When property value represents a physical (resp. a financial) amount, represent or reference in a computer-sensible way the physical unit (resp. the currency unit) used for scaling the value.

We present in the next section how these mechanisms have been implemented and may be represented in the PLIB ontology model.

4 PLIB: A Context-Explicit Ontology for Data Integration

Initiated in the early 90's the goal of the PLIB project was to develop an approach and standard models for exchanging and integrating automatically engineering component databases [26]. To allow such an automatic integration, an ontology-based approach has been developed. An ontology model (known as the PLIB ontology model) has been defined[3] and each PLIB-based data source is supposed to contain at least three parts: (1) an *ontology*, (2) a *database schema*, and (3) *instance data* represented according to the schema that references the ontology. Because one cannot assume that complete shared ontologies will ever exist, each database must have its own local ontology. But, to make automatic integration feasible, each particular local ontology may also contain (4) a *mapping* onto pre-existing shared ontology(ies) (e.g., standard ontologies) through semantic relationships. In particular, a specific subsumption relationship called *case-of* was defined to allow a local ontology to reference a shared ontology and to import properties without needing to duplicate class or property definitions. Development of standard ontologies is encouraged. Several such ontologies already exist or are in progress (see Annex A).

The role of a PLIB ontology is twofold. First it is intended to support user query over integrated component databases. Such queries need to be supported at various levels of abstraction (a *screw*, a *machine screw*, an *hexagon machine screw*, an *ISO 1014-compliant hexagon machine screw*). Thus, subsumption is a key feature of any PLIB ontology. Second, it provides for automatic integration.

We first present in this section a formalization of PLIB ontologies. 4.1 gives a short overview of the EXPRESS data specification language, and 4.2 presents, through two simplified schemas, the global architecture of the formal PLIB specification. Then, clauses 4.3 to 4.7 present the main mechanisms used to make context explicit in PLIB ontologies. Finally, clause 4.8 discusses the relationships between ontologies and schema in databases.

[3] ISO 13584-42:1998. Industrial Automation Systems and Integration, Parts Library, Methodology For Structuring Part Families. H. U. Wiedmer and G. Pierra, Eds. ISO, Geneva, 1998.

4.1 Specification of the PLIB Ontology Model: EXPRESS

The PLIB ontology model is defined in EXPRESS, a standard data specification language initially developed in ISO [30] to represent product models in the engineering field. The major advantage of this language is the integration of schema definition, constraint specification and instance representation capabilities in a common formalism with common semantics. This integration avoids the use of several models and languages like e.g., UML, OCL and XMI.

A specification in EXPRESS is represented by a set of schemas that may refer to each other. Each schema contains two parts. The first part is a set of entities that are structured in an object oriented approach supporting multiple inheritance. The second part is a procedural part that contains procedures and functions. These procedures and functions are used for restricting the allowed interpretation of the schemas by describing constraints on data. They are also used to specify how the value of a property of some entity may be computed from values of some other properties (derivation functions).

Each entity is described by a set of properties called attributes. Each attribute has a range (where it takes its values) defining a data type. It can be either a simple type (like integer, string ...), an entity type (meaning that the attribute establishes a relationship with another entity), a union of type (like integer OR string) or a collection over any data type (collections may be list, set, bag and array that are hard encoded in EXPRESS).

Syntactically one writes:

```
SCHEMA Foo1;
TYPE number_or_string = SELECT (REAL, STRING);END_TYTPE;
                                ENTITY b;
ENTITY a;                         att_1:number_or_string;
 att_a:OPTIONAL INTEGER;          att_2:LIST [0:?] OF STRING;
INVERSE                           att_3:a;
 att_i:                         DERIVE
  SET [0:2] OF b FOR att_3;       att_4:BOOLEAN
END_ENTITY;                          := EXISTS(SELF.att_3.att_a);
                                END_ENTITY;
END_SCHEMA;
```

Informally, the entity *b* has three attributes: a value that may be either a real or a string, a list of any number of strings and a pointer to another entity *a*. Entity *a* has only one integer attribute that may have no value (lack of value is represented by a particular symbol). Attribute *att_i* is an inverse attribute of entity *a*, corresponding to the inverse link defined by attribute *att_3* in entity *b*. At most, two instances of *b* may reference an instance of *a*. Attribute *att_4* is a derived attribute of entity *b* computed by the predefined EXPRESS function EXISTS. This function evaluates to *true* if its parameter has a value. In the case of entity *b*, this parameter is the optional *att_a* attribute of the instance of entity *a* referenced by the current instance (optional keyword SELF) of entity *b*. As usual, EXPRESS uses the dot notation to access attributes of an entity.

Semantically, an entity has a model. In the EXPRESS community, the model is named a physical file. The model consists of a set of entity instances with explicit instance identity. The attribute values are either literal values of the EXPRESS simple or structured built-in types or they are references to other entity instances. Instead of entering into deep semantic details, we give below an example of a model (physical file) which can be associated to the previous entity definitions. Note that an EXPRESS schema is an executable specification. EXPRESS compilers are able to generate both storage structures for managing EXPRESS models, checking constraints over these data or computing derivation functions and programs able to read and write physical files for which a standard syntax has been defined.

Let us consider a particular instance of the entity *b*, where *att_1* evaluates to 4.0, *att_2* is the list ('hello', 'bye') and *att_3* points the particular instance of the entity *a* whose *att_a* attribute evaluates to 3. Then, the model (physical file) associated to these particular instances of the entities A and B is described by (derived and inverse attributes are not represented as they may be computed):

```
# 1=A(3);
# 2=B(4.0, ('hello','bye'), #1);
```

It is possible to limit the allowed population (elements) of the models to those instances that satisfy some stated constraints. EXPRESS uses first order logic which is completely processable since the set of instances (physical file) is finite. Constraints are introduced thanks to the WHERE clause of EXPRESS that provides for instance invariant, and thanks to the global RULE clause that provides for model invariant.

Let us assume that the allowed values for *att_a* in *a* are [1..10] and that exactly two instances of entity *a* shall have an attribute value equals to 1. We may write (QUERY is a built-in iterator on class, and SIZEOF a built-in function that returns the size of a collection):

```
ENTITY a;
  att_A:OPTIONAL INTEGER;
WHERE
  WR1: correct_range (SELF.att_A);  -- WR1 is the constraint label
END_ENTITY;
RULE Card FOR a;
  WHERE SIZEOF(QUERY(inst <* a|inst.att_a=1)) = 2; END_RULE;
FUNCTION correct_range (val: integer): Boolean;
  BEGIN RETURN ((val>=1) AND (val<=10));END_FUNCTION;
```

All value domains and operators are extended with the INDETERMINATE ('?') value to process optional attributes, and EXPRESS uses a tree-valued logic (TRUE, UNKNOWN, FALSE) to return values of predicates that cannot be assigned a Boolean result. Assignment, sequence and control structures (if statements, loops and recursion) can be used in the function bodies. These features give powerful expression possibilities to the language. Indeed, one gets the same

expression possibility as other recursive specification languages. Derivations and constraints are the only places where functions may occur. They provide the two high level abstraction mechanisms identified as necessary in data driven active databases. Therefore, it is possible to specify formally a large class of problems. Moreover, derivations and constraints are inherited. These features define a set inclusion semantics to the EXPRESS inheritance mechanism.

4.2 PLIB Syntax and Semantics

To provide for easy integration of several ontologies, PLIB uses a meta-modeling approach for representing both ontologies, and ontology-based representation of domain objects. These two schemas are connected by formal constraints to ensure that instances of ontology classes fit with class descriptions. The partial schema below outlines the global structure of the PLIB ontology (meta) model (the data_type entity, not detailed, allows to specify the data type of a property).

```
SCHEMA PLIB_ontology;
TYPE class_id=STRING;END_TYPE; TYPE prop_id=STRING;END_TYPE;
TYPE class_ref=SELECT (class, class_id);END_TYPE;
TYPE prop_ref=SELECT (property, prop_id);END_TYPE;
ENTITY property;
 code:STRING; version:STRING; name:STRING; domain:class_ref;
 range:data_type; value_context:SET [0:?] OF prop_ref;
DERIVE
 prop_id:STRING:= compute_class_id(domain)+'.'+code +'-'+version;
END_ENTITY;
ENTITY class;
 id:class_id;  name:STRING;  superclass:OPTIONAL class_ref;
 case_of:SET [0:?] OF class_ref;
 imported_properties:SET [0:?] OF prop_ref;
 new_applicable_properties:SET [0:?] OF prop_ref;
DERIVE
 known_applicable_properties
    :SET [0:?] OF prop_ref := compute_applic (SELF);
WHERE
 WR1: is_acyclic([SELF], SELF.superclass);
 WR2: correct_importation(SELF.imported_properties, SELF.case_of);
 WR3: correct_applicability(SELF.new_applicable_properties, SELF);
END_ENTITY;
```

Properties and classes are identified by universal identifiers (UId) (*prop_id* and *class_id*), but references between them are done either by these identifiers, or by instance references to allow exchanging partial ontologies or referencing external ontologies. A property is defined in the context of the higher class (*domain*) where it is meaningful, even if it is not applicable (semi-rigid [16]) to some of its subclasses. Its UId concatenates the identifier of this class (user-defined function *compute_class_id*), its *code* and its *version*. If the value of a property depends

upon some evaluation context (see 4.6), parameters that characterize this context are specified (*value_context*). A class has at most one superclass. It selects among all the inherited (semi-rigid) properties, those that become applicable, (rigid [16]), i.e., essential for all its instances (*new_applicable_properties*). Moreover, it may also establish subsumption links with other preexisting classes (*case-of*), for instance from standard ontologies, of which it imports any number of properties (*imported_properties*).

The rules that govern the semantics of such a specification are formally defined by means of user-defined functions. As examples, *is_acyclic* asserts that subclass/superclass relationships do not include loops, *correct_importation* asserts that only properties defined for the classes referenced by means of *case_of* are imported from these classes and *correct_applicability* asserts that the current class is (possibly by inheritance) in the domain of all the properties it selects as applicable by *new_applicable_properties*). The final set of applicable properties of a class (that gathers inherited applicable properties, new applicable properties and imported properties) is also formally specified by means of a function (*known_applicable_properties*). When the specification is run over some model (physical file), if some classes are only referenced by their *class_id*s but are not available in the model, the assertions do not fail. They return an UNKNOWN result as allowed by the EXPRESS language.

Let us consider the class of *ball bearing* presented in Fig.2. Let us assume that it is a subclass of a predefined *bearing* class whose *class_id* attributes equals 'XX.bearing-1' and where all Fig.2 properties, but *ball radius*, are already defined as applicable. Then, the model (physical file) allowing to extend this predefined ontology by a new *ball bearing* class associated with a new *ball radius* applicable property would be as follows (*measure_type*, not detailed here, allows to represent a subtype of *data_type* that is a real number associated with a measure unit, and '()' represents the empty set):

```
# 1=PROPERTY('b_radius','1','ball radius','XX.ball_bear-1,#10);
#10= MEASURE_TYPE(...);
# 2=CLASS('XX.ball_bear-1','ball bearing','XX.bearing-1',(),(),(#1));
```

Ontologies being represented as instances of the PLIB ontology schema, another schema, called the PLIB instance schema, has been developed for representing domain objects (e.g., a particular ball bearing) as ontology individuals. Such individuals, called ontology-based data, may be exchanged together or without the domain ontologies to which they correspond. The partial schema below outlines the structure of the PLIB instance schema (the REFERENCE clause imports all the definitions from a referenced schema):

```
SCHEMA PLIB_instance;
REFERENCE FROM PLIB_ontology;
TYPE primitive_value=SELECT (integer, string, instance); END_TYPE;
ENTITY property_value;
 property:prop_ref; value:primitive_value;
WHERE WR1: correct_type(SELF.property, SELF.value);
```

```
END_ENTITY;
ENTITY instance;
 class:class_ref; properties:LIST [0:?] OF property_value;
WHERE WR1: correct_properties(SELF.class, SELF.properties);
END_ENTITY;
```

When ontology and ontology-based data are gathered in the same model (i.e., physical file or database), thanks to the constraints specification capabilities of the EXPRESS language, constraints over the instance schema allow to assert that each ontology individuals complies with its ontological definition in the following sense. The user-defined *correct_type* function ensures that the value of each property belongs to the range defined for this property at the ontology level. The user-defined *correct_properties* function ensures that an instance of an ontology class may only be described by properties that are applicable to this class, and that if the evaluation context of a property depends upon some other properties, any value of the former is associated with values of the latter. Note that the database schema of a class consists of the union of all its applicable properties that are associated with values for at least one of its instances.

Let us consider an instance of the *ball bearing* class presented in Fig.2 that is only described by its *ball radius* property that evaluates to 3.0 (in the unit specified for this property in the ontology, e.g., millimeter). Then, the model (physical file) allowing to represent this particular bearing would be as follows :

```
# 1=PROPERTY_VALUE('XX.ball_bear-1.b_radius-1',3.0);
# 2=INSTANCE('XX.ball_bear-1',(#1));
```

Thus, both PLIB ontologies and PLIB ontology-based data may be modeled, exchanged and checked for consistency by automatic tools generated from the two EXPRESS schemas. Note that only simplified versions of these schemas were presented above. We describe informally in the next sections, the various mechanisms used to make context explicit in PLIB ontologies.

4.3 Global Structuring of the Definition Context and Point of View Representation

The role of ontologies being to capture the essence of beings, PLIB supports a distinction between:

- those properties that are rigid [16] for a class, i.e., that are *essential* for any instance of a class (i.e., that must hold or have a value): all these properties are associated with a particular class
- those properties that may or not hold or exist according to the role in which an entity is involved.

Each category of real world objects is represented by one or more ontology classes. One particular class, point of view-independent, contains all the rigid

properties. If needed, point of view-specific classes gather those additional properties that correspond to a particular point of view over objects of the real world class.

For instance having a *birth date* is an essential property for any *person*: such a birth date may be unknown in some context, but, if it does not exist, the person does not exist. Contrariwise, having a *salary* is not an essential property. It exists only if the *person* is an employee of some organization and it corresponds to a *working status* point of view. For a mechanical part, having a *mass* is a rigid property, having a *price* is not. The *price* only exists if the part is sold on the market, and the price depends on the market (e.g., wholesaler or retail sale, quantity of order, discounted customer). It corresponds to the *marketing* point of view.

Of course, in a database schema, a *person* may have a *salary*, and a *part* may have a *price* and a *supplier*, but this is based on some implicit context assumptions that shall be explicit at the ontological level.

In fact, a PLIB ontology consists of three categories of classes of which only the first one was presented in clause 4.2.

- *definition classes* (modeled by the *class* entity in clause 4.2) capture the *beings* of the area of interest, together with all their rigid properties.
- *functional model classes* represent the additional properties that result from a particular role or point of view [27]. A functional model class exists only when associated with a definition class. Each instance of a functional model class is a view of an instance of a definition class. This relationship is termed *is-view-of*.
- *Point of view classes* capture the modeling context of (i.e., the point of view corresponding to) each particular functional model class: each functional model class shall reference a point of view class as its modeling context.

For instance, the definition class of a person should contain properties such that *birth date, gender, current name, first name*. An *employee* functional model class should contain properties like: *date of first employment, status, salary*. A *working status* point of view class allows to define the context of the functional model class. It may also contains for instance the *date of recording*, and the *employer id* attribute.

The definition class of a particular subclass of *mechanical* part, e.g., *screw* should contain properties like *threaded length, total length, threaded diameter, material*. The *screw procurement* functional model class should contain properties such that *price, quantity of order*. The *marketing* point of view class specifies the context of the screw procurement. It contains properties such that *date, kind of market* (e.g., wholesale, retail sale, negotiated), *supplier*.

4.4 Representation of the Local Definition Context

As noted in [19] a property cannot be understood if we don't know in which context it was defined: the same property names and informal definitions may

be used with quite different meanings in different context. Thus, to define unambiguously classes and properties of an ontology, a basic modeling principle is that:

- a property cannot be defined without defining, in the mean time, its field of application by means of the class where it is meaningful; this class constitutes its definitions context;
- a class cannot be defined without defining, in the mean time, the properties that are essential for its instances.

Following this principle, a PLIB ontology includes two aspects:

- a classification tree where classes and properties are identified and connected;
- a set of meta-attributes that describe successively each class and each property.

Defined through a set of formal relationships, the first aspect allows to formally retrict the allowed interpretations of an ontology. Described through a number of human-readable pieces of information, the second part allows to make understandable the real world semantics of the conceptualization represented by the ontology.

Property definitions are formulated in the context of the higher class where they are meaningful (attribute *domain* in 4.2), even if they don't apply to all its subclasses (in PLIB jargon they are said to be *visible* for this class, and all its subclasses). Then, class definitions specify which properties are *applicable*, i.e., essential for every instance of this class (attribute *new_applicable_properties* and *imported_properties* in 4.2). Finally, when instances are represented within some model (physical file) or some database schema, only a subset of all the applicable properties may be used to describe them (such properties are said to be *provided*). For any class C the following holds:

$$provided(C) \subset applicable(C) \subset visible(C)$$

This formula shows, at the property level, the difference between an ontology and a schema: various schemas, designed by various database administrators, may represent for the same ontology class C various subsets of *applicable(C)*. During an integration process, and thanks to the UId of each ontology concept, it will be obvious which properties are the same and which are not.

Concerning the classes to be defined, PLIB is property-oriented: all what can be described meaningfully by properties is defined by properties. A class shall only be introduced in an ontology when it constitutes the domain of a new property, i.e., the property would be meaningless for the superclass of this new class, but it is meaningful for the new class and all its subclass. Thus reference ontologies are in general rather flat. For example, the *internal diameter* property is meaningless for a *mechanical component* whatever its definition. It becomes meaningful if one introduces a new subclass of mechanical components that models *circular bearings*.

But these formal relationships between classes ands properties are not sufficient for unambiguous definitions. Indeed, a PLIB ontology mainly consisting of

primitive items, i.e., items whose definition "must rely on textual documentation and a background of knowledge shared with the reader to convey meanings" [15], the PLIB ontology model includes an extensive number of (meta) attributes used for representing the real world conceptualization and for connecting the ontology constructs to the background knowledge of the ontology user. In clause 4.2 only the *name* (meta) attribute was presented, in fact these (meta) attributes include: names and synonymous names, symbols, definitions with notes and remarks, pictures and drawings, references to document.

4.5 Locality of Ontology Interpretation Context

When a particular domain ontology is developed, it is often the case that (1) its domain overlaps the target domains of some other ontology, and (2) the perspective adopted in these various ontologies is, at least, partially different. For example, a travel ontology needs the capability to capture the concept of a *plane*. Let's assume that a plane ontology exists. If such an ontology has been developed to provide a suitable vocabulary for exchanging information between airplane manufacturers and airline companies, the ontological definition of a plane might contain such properties as *frequency of maintenance operation*, *guaranty duration*, and a number of technical properties which are useless in the context of the travel ontology. If the capability to use the basic *plane* properties in another ontology requires to integrate all the *plane* properties, more all the *plane* subclasses, probably the travel ontology designers would prefer to define their own *plane* concept. Indeed, the plane ontology might contain a number of technical details not understandable by the travel ontology designers, thus, they would not be able to understand the global conceptualization resulting from the global merging.

For addressing this issue, the PLIB ontology model introduces the *case-of* subsumption relationship. This relationship affects only one class of the referenced ontology from which it imports some selected properties, and the interpretation domains of both ontologies remains different.

Note that this importation is compatible with the local definition context representation discussed in 4.4: the referencing class being subsumed by the referenced one, the former is included in the domain of the latter. This mechanism allows to provide a view of a local ontology in terms of a global one, and, if the local classes is also defined as a restriction of the referenced classes, to migrate instances from the global context to the local one.

4.6 Representation of the Local Value Context

In a number of cases, the value of an instance property changes when its evaluation context changes. This means that the range of such properties is not a value set, it is a function set. Let C be the set of all instances of a class, P be a property whose domain includes C, D be the range of values of P, $EVAL_{C,P}$ be the set of all the states of the context where values of property P may be

evaluated for any instance of C, P_1, ..., P_n be the set of properties allowing to characterize the states of $EVAL_{C,P}$, and D_1, ..., D_n their ranges of values.

- A *characteristic property* (*characteristic* for short) is a property that defines a function over C:
$$P : C \rightarrow D.$$
- A *context parameter* is a property that defines a function over $EVAL_{C,P}$:
$$P_i : EVAL_{C,P} \rightarrow D_i.$$
- A *context dependent property* is a property whose value is a function of the context:
$$P : C \rightarrow (EVAL_{C,P} \rightarrow D).$$

Table 2 shows various examples of characteristics and context dependent properties.

Table 2. Representing value context

Entity	**Person**	**Ball bearing**	**Plane**
characteristic	birth date	inner diameter	plane type
context-dependent property	hair color	life time	cheapest fare
context parameter	date	load, speed	customer age

In the PLIB ontology model, the signature of the function corresponding to a context dependent property is defined by the *value_context* attribute (see 4.2), and context parameters must be explicitly defined within the ontology. Two means are provided for specifying the function itself:

- either, as suggested in [31], it is discretized at the instance level as one or several sets of property-value pairs, each set defining the particular value of the context dependent property for a particular evaluation context state, defined by context parameter values;
- or, when the dependency may be expressed by an algebraic function that is the same for all instances and all interpretations, the function itself may be represented at the ontology level as an instance of an expression meta-model[4].

Moreover, on the database site, a database administrator may implement the function as a database-defined function, allowing a user to query the database by means of context parameters and context-dependant property values.

Of course, the ontology designer may decide to freeze all the context parameter values within a property definition, like: *hair color when birth; life time for 100 Pascal radial load and 6000 RPM; cheapest fare when 65 years old*. But, if the whole evaluation context is not specified within a property definition, this property shall be represented as a context-dependent property. In this case, the

[4] This expression meta-model is defined in ISO13584-20.

context parameters of which its value depends shall be explicitly modeled at the ontology level, together with the dependency relationships.

Note that representing instances is a question of schema and not of ontology. As discussed in Sciore et al. [31], all the context-parameter/value pairs that characterize a context dependent property value shall be represented by some means: at the property value level, at the instance level if the same context has been used for all the instance properties, or even at the level of the whole database if properties of all instances were evaluated in the same context. Anyway, the PLIB ontology model includes axioms that ensure that context-dependant values cannot exist in a model without their evaluation context.

4.7 Representation of Value Scaling

To provide for automatic value conversion and integration, units and currencies must be formally modeled. But they may be represented either at the ontology definition level, or at the value level. In a PLIB ontology, default units have to be represented at the ontology property definition level, together with alternative units. The default unit may be overridden by an alternative unit at the value level by associating each value with its own unit. The unit model allows to represent both dimensional exponents for a physical quantity, and all kinds of measure unit: either SI unit (e.g., millimeter), derived (e.g., m/s), or conversion-based unit (e.g., inch).

4.8 From Ontology to Schema

Provided that property inheritance and referential integrity is ensured, any subset of classes of a CCO, each one associated with any subset of its applicable properties, defines a database schema. We call *ontology-based database* (OBDB)[28] a database (1) that explicitly represents an ontology, (2) whose schema refers to the ontology for each of its represented class and property. In such a database, each data may be interpreted in a consistent way using the meaning defined for the corresponding ontology entry. Note that an OBDB is not required to populate either all the classes of its ontology or all the properties defined for a given class. Moreover, provided that the link from data to ontology is preserved, the schema structure is not required to preserve the ontology structure. Inheritance composition and view-of relationship may be "flattened". This means that values representing:

- properties of a definition class instance,
- properties of a part of this instance, and
- properties of a functional model class instance that is view-of the definition class instance

may all appear as values in the same database relation. This shows the diversity of the various schemas that may be built just from the same ontology while preserving semantic integration capabilities.

5 Formal Definition of PLIB Ontologies

In section 4 we have outlined, through partial schemas, how PLIB ontologies and ontology-based data were specified and exchanged using an executable data specification language. We also detailed informally the various mechanisms used for making context explicit in PLIB ontologies. In this section, we present a formal model of PLIB ontology semantics independent of any syntax (note that an XML syntax, called OntoML, is currently under ballot within ISO as an alternative for exchanging PLIB ontologies). This model covers globally the PLIB specification. Its only restriction is to focus on ontologies that consists of definition classes (no functional model class or point of view class as the use of these construct is not widespread). A PLIB ontology may be defined separately as a single ontology, but it may also be mapped onto one (or several) standard ontologies. These two models are presented respectively in 5.1 and in 5.2. Clause 5.3 outlines how a shared ontology and various mapped ontologies may be used to integrate heterogeneous data bases. To illustrate the various aspects of the formal definitions, example 3 represents formally the ontology described in example 1

5.1 Single PLIB Ontology

Formally, a single PLIB ontology may be defined as a 8-tuple :

$$O =< C, P, U, IsA, PropCont, ClassCont, ValCont, ValScale >,$$

where:

- C is the set of classes used to describe the concepts of a given domain;
- P is the set of properties used to describe the instances of C; P is partitioned into P_{val} (characteristics properties), P_{fonc} (context dependent properties) and P_{cont} (context parameters);
- U is the set of units of measure, including currencies, used to describe the values of the properties of a given domain that define a measure;
- $IsA : C \rightarrow C$ is a partial function[5] that associates to a class its smallest subsumer[6]; IsA implies inheritance of both visible properties (as visible) and of applicable properties (as applicable);
- $PropCont : P \rightarrow C$ associates to each property the higher class where it is meaningful;
- $ClassCont : C \rightarrow 2^P$ associates to each class all the properties that are applicable to every instances of this class (rigid properties);
- $ValCont : P_{fonc} \rightarrow 2^{P_{cont}}$ associates to each context dependent property the context parameters of which its value depends;
- $ValScale : P \rightarrow U \times 2^U$ is a partial function that associates to each property that defines a measure the default unit used to represent its values, and, possibly, the other units that may be used to override the default unit.

[5] IsA is assumed to define a single subsumption hierarchy.

[6] C_1 subsumes C_2 iff $\forall x \in C_2 \Rightarrow x \in C_1$.

Example 3. Definition of the circular bearing ontology presented in example 1 is as follows.

- $C = \{$*circular bearing, circular ball bearing*$\}$;
- P_{val}=$\{$*inner diameter, outer diameter, width, ball radius*$\}$, P_{cont}=$\{$*velocity, radial load, axial load*$\}$, $P_{fonc} = \{$*life time*$\}$;
- $U = \{$*millimeter, meter, revolutions per minute, newton, hour* $\}$;
- $IsA($*circular ball bearing*$) =$ *circular bearing*, $IsA($*circular bearing*$) = \emptyset$;
- $PropCont($*inner diameter* $|$ *outer diameter* $|$ *width* $|$ *velocity* $|$ *radial load* $|$ *axial* $|$ *life time*$) =$ *circular bearing*, $PropCont($ *ball radius* $) =$ *circular ball bearing*;
- $ClassCont($*circular bearing*$) = \{$*inner diameter, outerdiameter, width, velocity, radial load, axial load, life time*$\}$, $ClassCont($*circular ball bearing*$) = \{$*ball diameter*$\}$;
- $ValCont($*life time*$) = \{$*velocity, radial load, axial load*$\}$;
- $ValScale($*inner diameter* $|$ *outer diameter* $|$ *width* $|$ *ball radius* $) = ($*millimeter,* $\{$*meter*$\})$, $ValScale($*radial load* $|$ *axial load* $) = ($*newton* , $\{\})$, $ValScale$ $($*velocity*$) = ($*revolutions per minute,* $\{\})$, $ValScale($*life time*$) = ($*hour,* $\{\})$;

Example 4. Let's assume that, using some bearing ontology, a user queries the life-time of a circular bearing whose part identification property equals XYZ when its velocity is 1500 rpm and it supports a radial load of 6000 N. Using the OntoML PLIB syntax, a system answer would be as follows (all class and property UIds come from a bearing ontology standardized as ISO 23768, the codes of class and properties have been changed to reflect their meaning, all concepts are in version 1, and *condition* is the OntoML tag for context parameters):

```
<item class-ref="ISO23768#CIRCULAR_BALL_BEARING#1">
        <property-value property-ref="ISO23768#PART_IDENTIFICATION#1">
                <val:string-value >XYZ</val:string-value>
        </property-value>
        <property-value property-ref="ISO23768#LIFE_TIME#1">
                <val:real-value>20000</val:real-value>
                <val:condition>
                        <val:element property-ref="ISO23768#VELOCITY#1">
                                <val:real-value>1500</val:real-value>
                        </val:element>
                        <val:element property-ref="ISO23768#RADIAL_LOAD#1">
                                <val:real-value>6000</val:real-value>
                        </val:element>
                </val:condition>
       </property-value>
</item>
```

Four axioms are defined on this formal PLIB model. If we define recursively the visible properties as[7]:

$$visible(c) = visible(IsA(c)) \bigcup PropCont^{-1}(c),$$

then the following axioms shall hold :

[7] To simplify notation, we extend all functions f by $f(\emptyset) = \emptyset$.

1. IsA defines a single sumsumption hierarchy: the graph G whose vertex are classes and whose edges are the IsA relationships is a forest, i.e., a disjoint union of trees. G is defined by:
   $G = \{C, \{(c_1, c_2)\} | c_1 \in C \wedge c_1 \in Dom(IsA) \wedge c_2 \in IsA(c_1)\}$
2. IsA implies inheritance of applicable properties:
   $ClassCont(c) \supseteq ClassCont(IsA(c))$
3. A context-dependent properties must not be defined in a class that does not belong to the domain of the context parameters of which it depends:
   $\forall c \in C, p \in P_{fonc} \;\; p \in ClassCont(c) \Rightarrow ValCont(p) \subset ClassCont(c)$

Moreover, for stand-alone ontologies, one more axiom applies: only meaningful properties (i.e., visible properties) may become applicable :

(4a) $ClassCont(c) - ClassCont(IsA(c)) \subset visible(c)$

5.2 Mapped PLIB Ontology

A major focus of PLIB ontologies being heterogeneous data source integration, PLIB does not assume that all data sources use the same ontology. Each data source may build its own local ontology without any external reference. It may also build it based upon one or several existing ontologies (e.g., standard ones). A class of a local ontology may be described as subsumed by one or several other class(es) defined in other ontologies by the *case-of* relationship. Through this relationship the subsumed class may import properties (their UIds and definitions are preserved, as presented in 4.2). But it may also map properties that are defined in the referenced class(es) (the properties are different but they are semantically equivalent) . A class of a local ontology may also define properties that are neither imported nor mapped.

A PLIB ontology O_m that includes mapping onto one (or several) other ontologies may be formally defined as a pair: $O_m =< O, M >$, where : $O =< C, P, U, IsA, PropCont, ClassCont, ValCont, ValScale >$ is an ontology, and $M = \{m_i\}$, is a mapping defined as a set of mapping objects.

Each mapping object has four attributes: $m =< domain, range, import, map >$

- $domain \in C$ defines the class that is mapped onto an external class by a *case-of* relationship;
- $range \in UId \subset \{string\}$ is the universal identifier of the external class onto which the $m.domain$ class is mapped;
- $import \in 2^p$ is a set of properties visible or applicable in the $m.range$ class that are imported in $ClassCont(m.domain)$;
- $map \subset \{(p, id) \;\; | \;\; p \in P \wedge id \in UId \subset \{string\}\}$ defines the mapping of properties defined in the $m.domain$ class with equivalent properties visible or applicable in the $m.range$ class. The latter are identified by their UIds.

Note that each mapping object defines a subsumption relationship between the $m.range$ and $m.domain$ classes. Nevertheless, the $m.range$ class does not belong to C. The interpretation domain of the referencing ontology remains different from the one of the referenced ontology. Note also that when properties are imported, they belong to P.

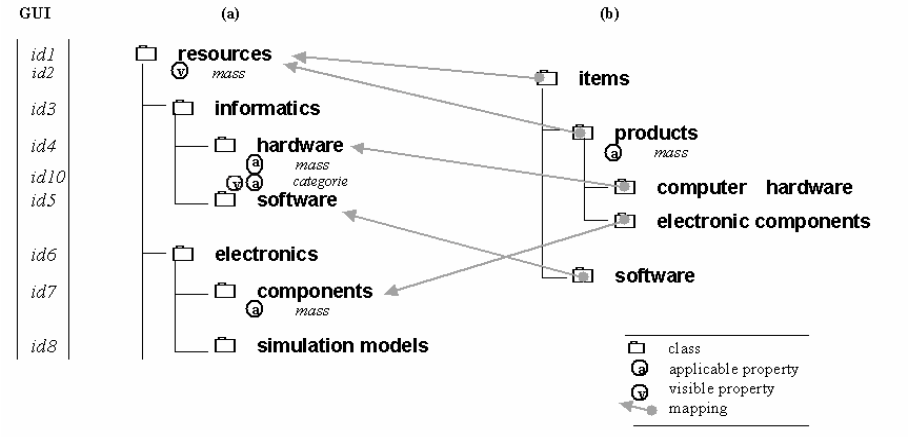

Fig. 3. An example of a reference ontology (a) and of an user defined ontology (b)

Example 5. Figure 2 (a) presents a single ontology. Class hierarchy is represented by indentation. $P = \{mass\}$. The *mass* properties applies to *hardware* and *components*, but not to *software* and *simulation models*. *mass* is visible at the level of *resources* : $PropCont(mass) = resources$, with a definition s. t. "the physical mass of a resource that is a material object". It becomes applicable in *hardware* and *components*: $ClassCont\ (hardware) = \{mass\}$; $ClassCont\ (component) = \{mass\}$

Example 6. Figure 2 (b) presents a (user-defined) ontology mapped on a reference ontology (a). $C = \{items,\ products,\ computer\ hardware,\ electronic\ components,\ software\}$ and $P = \{mass\}$. $M = m1, m2, m3, m4$ with $m_1 = (item, id1, (), ())$; $m_2 = (products, id1, (id2), ())$; $m_3 = (computer\ hardware,\ id4,\ (),\ ())$; $m_4 = (electronic\ components,\ id7,\ (),\ ())$. We note that no properties are mapped, they are all imported.

All the axioms for single ontologies hold. The specific axiom (4a) becomes (4b and 4c) that state that imported properties belongs to the set of applicable properties of the importing class (and of its subclasses), and that the other new applicable properties of the importing class shall belongs to its visible properties.
(4b) $\forall m \in M, ClassCont(m.domain) \supset m.import$,
(4c) $\forall m \in M, LetM(m) = \{m_i \in M \mid m_i.domain = m.domain\}$

$$(ClassCont(m.domain) - ClassCont(IsA^{-1}(m.domain)) - \bigcup_{m_i \in M(m)} m_i.import) \subset visible(m.domain)$$

As shown by Figure 2, the structure of a (user) ontology may be quite different from the one of a standard ontology it references. Nevertheless, a system storing the user ontology $< O, M >$ may automatically answer queries against

a standard ontology onto which O is mapped. It may also migrate instance data from its local user ontology to the standard ontology.

Note that the above mapping only allows to query local ontologies through one, or a set of standard ontologies of which the former are case-of, and to return the answer as standard ontology individuals. A typical application is the case where component provider data sources are based on ontologies defined as specialization, through case-of, of a standard ontology. The customer formulates its query in terms of the standard ontology. The answers is also returned in terms of the standard ontology, whatever local ontology is used by the provider. Nevertheless, this approach does not allow:

- to know the precise definition of the provider product; for example if additional properties were defined by the provider, value of these properties cannot be returned in the answer;
- to store automatically the returned data in the customer database when the customer has also created its own local ontology by specialization from the same standard ontology, as I recommend it in 6.

Concerning the first problem, if the customer needs this precise information, the provider may return the answer not as a projection onto the standard ontology, but in the native terms of the provider ontology together with the relevant specialization of the standard ontology defined locally. Then, these two pieces of information may be integrated automatically within a customer ontology-based database providing fine grain access to provider-defined specialization. This approach, called ExtendOnto, was proposed in [2]. Note that such an approach may be followed both with PLIB ontologies, using one of the PLIB exchange format, and with C-OWL [4] ontology, using OWL syntax.

Example 7. Let's assume that in Figure 2, 2a is the standard ontology and 2b is the provider ontology, and that a customer wants to retrieve those *hardware* products whose mass is less that 10^4 .

The provider answer may consist of two parts:

1. the *computer hardware* class definition together with all its applicable properties and its subsumption relationship with the standard *hardware* class;
2. the set of instances of the *computer hardware* class that correspond to the customer request.

Concerning the capability for a customer to map a set of standard-ontology-defined instances onto its own locally defined specialization of the standard ontology, this can be done by adding to each mapping object m a fifth attribute called $filter$ that is a predicate over the subsuming class instances:

$filter : (class\text{-}of(m.range))^I \longrightarrow Boolean.$[8]

The meaning of such a filter is that all instances of the subsuming class for which the predicate holds are members of the subsumed class:

$\forall x \in (class\text{-}of(m.range))^I, m.filter(x) \Longrightarrow x \in m.domain.$

[8] We note class-of the function that associates to a class identifier the corresponding class. As usual, we note $(.)^I$ the interpretation function.

Example 8. Let's now assume that, in Figure 2, 2a is the standard ontology and 2b is the customer ontology. The customer has retrieved those *hardware* products whose *mass* is less that 10^4, and the value of the *category* property of these *hardware* products. If, in the mapping m of his/her *computer hardware* class onto the standard *hardware* class, the following filter was added :
$m.filter = (category = "computer")$,
then all the returned *hardware* instances whose *category* values are "*computer*" are automatically recorded as a *computer hardware* instances. Other *hardware* instances are not recorded in the customer database.

5.3 Automatic Integration of Data Sources through a Priori Ontology Mapping

In the domains where it has been feasible (possibly using the context representation mechanisms defined in this paper) to define a consensual domain ontology, this ontology may be used to allow automatic integration of ontology-based data sources in the following sense:

- Let's assume that there exist some consensual ontology O over the domain that is common to all the sources;
- Let's also assume that each local source S_i is associated with a local ontology O_i and that each class C_{ij} of S_i that is in the domain modeled by O is mapped by a subsumption relationship (e.g. *case-of*), either directly or indirectly (through inheritance within O_i), onto its smallest subsuming class C_j in O (*smallest subsuming class reference requirement: SSCRR*) [2]
- Then, each local source, whatever its local ontology, may answer queries stated in terms of O.

Note that this automated integration technique leaves a lot of schematic autonomy to each data source. It only assumes that each database administrator wants to make its data available in terms of a standard domain ontology. Thus, each administrator is required to describe *a priori* a mapping between its own local ontology and the consensual ontology by means of a subsumption relationship ensuring the SSCRR assumption, and to import or to map properties having a common meaning. This *a priori* approach, different from most existing approaches where ontology mapping is done at integration time [23], seems to suit quite well the needs of a number of Web applications, including in particular B2B e-commerce. This approach is discussed in more details in [2].

6 A Road Map for Implementing Ontology-Based Databases in Manufacturing Enterprises

Currently, most of manufacturing enterprises still record their component information in conventional component databases where the various components are all described by the same set of relational attributes, one of them encoding in a long string (often called "designation") all the engineering properties (Fig. 3).

Such a representation has two major drawbacks. At the cost level, conventional component databases promote the increase in the number of similar components. Indeed, when a designer is searching for a component, there are very few chances that the best existing candidate be retrieved using string matching. As a result, new components are created again and again, increasing dramatically the cost of company products. At the quality level, few engineering properties may be encoded in a single string. Therefore, components are often selected only from force of habit without checking for each particular design whether all the engineering requirements of the problem at hand are really fulfilled by the selected component.

'SCREW-ISO1014-L10-D5-GRADa'

Fig. 4. Engineering information encoding in usual component database

Improving this situation requires migrating from conventional component database to engineering database where each class of components is defined by its own engineering properties. Taking into account that standard ontologies are emerging in more and more industrial domains, a major issue is to decide whether the corporate engineering database should use a private ontology or one (or a set of) standard ontology.

The direct use of standard ontologies may seems attractive, but it would have several drawbacks.

- A number of domains being not yet addressed, it would need to wait, but the market is not waiting.
- Even if the enterprise industrial sector is addressed, the relevant standard ontology probably does not contain all the classes and properties needed. And it surely contains a number of classes which are useless.
- Standards are rather stable. Nevertheless each standard is to be updated from time to time. Remaining in line with a standard ontology might request to change the corporate database schema when it is no longer in line with updated standards.

Contrariwise, developing its own corporate ontology would have a number of advantages.

- It is possible immediately, whatever the particular industrial sector is.
- Provided that there exist some mechanisms allowing to control impact of standard ontology evolution onto corporate ontology evolution, it would allow to gather standard definitions and local definitions. The corporate ontology may borrow class definitions and import standardized properties from standard ontologies, while adding company-specific classes and properties.
- It would ensure that each company remains free to upgrade its own ontology when and how it is needed, either by importing new standard properties or by creating new proprietary elements.

Note that both PLIB, through case-of, and OWL, through C-OWL [4] offer suitable mechanisms for controlling impact of standard ontology evolutions over local ontologies.

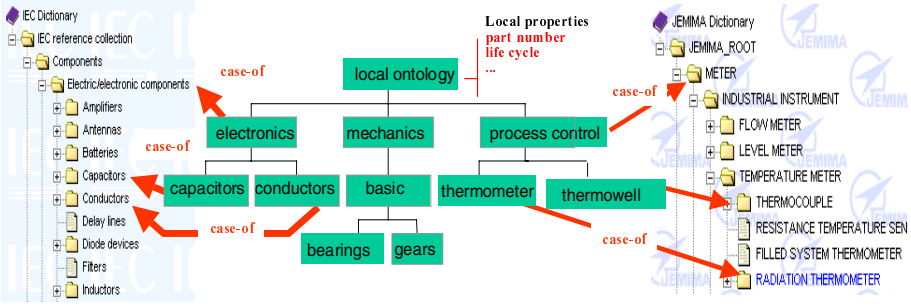

Fig. 5. Defining corporate ontology from standard ontologies (IEC 61360-4:1998 and ISO 13584-501:2006)

This suggests the following road map for switching from conventional component databases to ontology-based engineering databases.

1. Define an ontology that consists of a single class to host those generic properties that need to be available for describing any existing component. Definition of these generic properties must take into account both existing standard ontologies and the current content of the company conventional component database if any. Then, this ontology will be referenced by the main corporate ontology and all the classes where generic properties need to be used will be subsumed by this class (in PLIB, using case-of).
2. Define a proprietary overall classification of the various component domains until class nodes where generic search would make sense (e.g., metric screw, circular bearing).
3. At the level of each of these classes, define the technical properties needed for characterizing their components, importing as many properties as possible from existing standard ontologies using subsumption (in PLIB, using case-of).
4. If some needed properties require more precise class for defining their application domain, refine the existing classes using subsumption relationship from corporate classes, and possibly from standard classes when properties of which they define the domain need to be imported.
5. Use this ontology structure for defining the logical schema of the new corporate engineering database, implemented either on top of an OBDB (ontology within the database) or on a relational or object relational DBMS.
6. Extend progressively the existing schema when new needs, and possibly new standard ontologies, emerge.
7. If automatic exchange of component information appears both feasible and useful, define mapping from proprietary class and property onto standard class and property when the latter become available.

7 Related Work

We discuss below three research threads deeply connected with the material presented in this paper: (1) the role of context representation in data integration, (2) context representation in some ontology models, and (3) some proposed approaches for ontology-based data integration.

7.1 Context Representation for Semantic Integration

Importance of context representation for data integration was identified by several researchers in the field of multidatabase system in the 90's. Kashyap and al. [19] proposed to represent the intentional definition context, at the schema level, as a set of meta-attributes expressing intentional properties, and in particular the constraints each object must fulfill. They proposed to use description logic (DL) to reason over such a context. But, in this work, the evaluation context of property values was only informally defined. Sciore and al. [31] proposed to represent value context at the value level by means of another set of meta-attributes. For these authors, a semantic value is a piece of data together with its associated context. This context may be represented, e.g., as a LISP-like list of meta-attribute-value pairs, or a set of environment variables. Following this work, the COntext INterchange project (COIN) was developed in MIT [13,14]. This project noted that a number of property values depends both of the evaluation context in which they are evaluated, and on the way in which they are represented. They proposed to associate with properties both *attributes* and *modifiers*. Attributes characterize the evaluation context of property values (e.g., the *date* where some financial property was evaluated). Modifiers characterize how property values should be interpreted (e.g., the *currency* in which it is represented, and possibly the *scaling factor* used to encode the value). In an integration process, both the information source and the information receiver specify their respective context for all the properties of some shared ontology. Then, a *context mediator* ensures the conversion of data from the export context to the import context to achieve interoperability at the semantic layer.

The idea to associate to each source the context of all its information element as a set of meta-data was also followed by Ziegler et al. [35] in the SIRUP project. The SIRUP system assumes the existence of one or several shared ontologies, but these ontologies are not supposed to explicitly define in which context properties are evaluated and represented. Therefore, source owner must build an intermediate model, called IConcepts (intermediate concepts), where each ontology concept is associated with "extensive meta-data ...(attribute data types, measurement unit, precision, constraints, etc)" [35].

In the context of data warehousing, a powerful data integration and reconciliation approach based on value context representation was proposed by. Calvanese et al. [5]. In this approach, domain conceptual model and source conceptual models, similar to ontologies, are formalized using a specific description logic, called $\mathcal{DLR}$, which supports n-ary relations. Articulations between global model and

source models are specified by means of inter-model assertions, and the links between conceptual and logical levels is formally defined by associating with each relational table a query over the conceptual model that describes its content. This query is adorned by annotations that represent the local value context of each table column (e.g., the currency used for a price). Data conflicts are avoided by declaratively specifying suitable matching, conversion and data reconciliation operations by means of non-materialized views adorned by the name of a program able to compute the view. Then, a re-writing algorithm is able to compute automatically (or semi-automatically) the query allowing to load the various data warehouse relations.

If all these contributions developed efficient integration algorithms once value contexts are made explicit, up to now, few ontology models provided the necessary meta-attributes. Thus, context representation could not be provided by source owners. It needed to be done at source integration time, thus preventing automatic integration. Extending ontology models to support extensive context representation, as proposed by this paper, would constitute a major step toward automatic integration of heterogeneous data sources.

7.2 Context Representation in Ontology Models

Currently, most ontology models, and in particular OIL [11], DAML [6] and OWL [20] are based on DL. The main focus of these ontology languages is semantic annotations of Web resources using terms, and inferences over these terms. As a rule, DL-based ontologies consist of two parts. The TBox specifies class-level and property-level axioms. Class, and possibly properties, are structured as a subsumption lattice. The ABox (that may be empty) consists of a number of individual assertions. A class lattice is a powerful means for representing class definition context as it supports two important reasoning tasks [1]. Subsumption checking amounts to check whether a class is a subclass of another class. Class membership inference allows to checking whether an individual is a member of a specific class.

Concerning property definition context, most formalisms allow (but do not require) that a property is associated with a domain. Provided that this capability is systematically used in each source ontology and that ontology-level information may be accessed at integration time, an integration system may be able to distinguish, e.g., *department.name* and *employee.name* and to know that these two attributes are not semantically equivalent. Moreover, the C-OWL extension of OWL [4] allows to contextualize the interpretation of OWL constructs when they are imported from an ontology into another one. In such a case, the classical OWL semantics [25] assumes the existence of a unique interpretation domain used both for the referencing ontology and the referenced ontologies. This may lead to inconsistency, in particular when imported ontologies evolve. To the contrary, C-OWL associates with each ontology its own local domain [4]. The various domains may overlap but they are different. For the importing ontology, the local interpretation of an imported construct, i.e., concept or role, is different from its interpretation in its source ontology. This interpretation is

restricted to the set of objets that belong to the local interpretation domain of the importing ontology. Bridge rules may be defined between imported and importing ontology constructs, thus controlling how object may be mapped, or migrated, from an imported ontology domain to a local domain. Such a domain contextualisation provides the required autonomy for corporate ontology to implement the road map proposed in section 6 also in OWL.

So, DL-based ontologies allow to represent important aspects of schema definition context and to contextualize interpretation domain. But they are much less efficient for representing context at the extensional level. Indeed, most DL languages support only unary (classes) and binary (properties) predicates. And binary predicates may only have class as a domain. Therefore, it is impossible, in DL-based ontologies, to connect formally two properties. It is neither possible e.g., to define, like in COIN, that the value of a *financial property* depends upon a *date*, or that a *length* depends upon a *temperature*, nor to express that the *financial value* is represented in *billions* of *Euros*, the *length* in *millimeters*, and the *temperature* in *degree Celsius*. These drawbacks of DL-based ontology languages, that exist also for OWL, require evolution of these languages, as suggested in this paper, for making them really usable in domains like engineering.

7.3 Ontology-Based Integration of Information

Various approaches have been developed for ontology-based integration of information [33]. In the single ontology approach each source is related to the same global domain ontology (e.g., PICSEL [12], COIN [14]). As a result, a new source cannot bring any new or specific concept without requiring change in the global ontology. In the multiple ontologies approach (e.g., Observer [21]), each source has its own ontology developed without respect of other sources. In this case the inter-ontology mapping is very difficult to define. This is because the different ontologies may use different aggregation and granularity of the ontology concept [33]. To overcome the drawback of single or multiple ontology approaches, several researches have proposed an hybrid approach where each source has its own ontology, but where all ontologies are connected by some means to a common shared vocabulary. For instance, BUSTER system [32] assumes that local ontologies are only restrictions of the global ontology. PLIB-based integration follows the hybrid approach and proposes a formal model for ontologies and ontology mappings. But, unlike BUSTER it does not restrict source autonomy and sovereignty: each source may define its own classes and completely re-structure the class subsumption hierarchy. It may also add whatever properties. To give modeling autonomy to the local sources, we use the same kind of ontology articulation as ONION [23]. But, unlike ONION, we suppose that articulation between local and shared ontology is done *a priori* by the local source administrator (as done in another context in e.g., [35]). As a result, our integration approach is completely automatic and it scales to any number of data sources [2]. Note that the PLIB ontology model is the first model we know that explicitly

represents ontology mapping within a local ontology as a first class citizen (see 4.2: *case_of* and *imported_properties* attributes) as suggested by model management vision [3].

8 Conclusion

The concept of a domain ontology was mainly studied in computer science since early 90's. Its intent is to capture and to represent the essential nature of things of a domain through class structures and properties. In a number of computer disciplines, such an explicit representation of semantics appeared like some kind of philosopher's stone and a lot of languages, understandings, models and approaches were developed. Not surprisingly, differences in approaches reflect differences in the addressed problems, and it is often unclear how the various approaches and languages fit with each other and how they may be used for addressing a particular problem.

In this paper, we have investigated the use of ontology in a structured data integration perspective. First we have proposed a taxonomy of ontologies. Linguistic ontologies (LO) represent words and words relationships. They are natural language-oriented. They provide, in particular, for intelligent structuring, modeling and querying set of documents, such as those available on the Web. But they may also be used for defining a canonic human vocabulary for a particular domain, or for searching for equivalence between concepts through relationships between their linguistic descriptions. Conceptual ontologies (CO) represent concepts, as they are manipulated in the structured data universe like database or data engineering, and concept properties. Like for LO, two slightly different but complementary problems may be addressed using CO. The first one is to define a set of concepts allowing software systems, databases or agents existing within some community to exchange unambiguously information about a domain. For this purpose, concept equivalence should be avoided and canonical conceptual ontology (CCO) are needed. The second one is to also map different conceptualizations over the same domain. In this case, several CCOs need to be gathered within a unique non-canonical conceptual ontology (NCCO) that includes operators for reasoning over concept equivalence. In both cases however concept definitions and value interpretations must be unambiguous across their target community, and we have shown that this requests, in turn, ontology models of which context sensitivity has been explicitly represented and minimized. We have defined five principles to ensure that the definitions and value representations within an ontology are not context-sensitive and may thus be used to support semantic integration of data while leaving enough autonomy to the various sources. We have also shown how these principles have been implemented within the PLIB model, a CCO model developed to support integration of industrial data. The goal was not to promote PLIB as an alternative ontology language, but to identify and to illustrate those mechanisms that any ontology formalism should support to be usable for large-size integration of data. These principles are as follows:

- *Definition context representation.* Each property should be defined in the context of a class. Each class should define all its rigid properties, at least in some very broad context common to all the target data sources.
- *Point of view representation.* If several perspectives are needed over the domain, an ontology of perspectives should be defined and each needed perspective should correspond to a specific domain ontology.
- *Locality of interpretation context.* Resource importation between ontology should be feasible on a class per class basis, and then on a property per property basis. Interpretation domains of both referenced and referencing ontologies should be separated.
- *Value context representation.* Value dependency between property values should be explicit.
- *Value scaling representation* Unit and scaling of values should be explicit and computer interpretable.

A first version of the PLIB ontology model is now standardized, and a number of standard ontologies and of implementations are now emerging in various domains, and in particular in e-procurement and e-engineering that were the initial domains targeted by PLIB. Currently, most major manufacturing enterprises are switching from conventional component databases to ontology-based engineering databases that should allow to reduce useless component diversity, to improve component selection support and to facilitate integration of supplier catalogs, whatever the ontology model used. In this domain, our recommendation is not to use directly standard ontologies if they exist. It is to define a proprietary ontology and (1) to map classes onto standard classes if and when they exist, and (2) to import as much properties as needed properties from standard ontologies. Not only this approach may be followed immediately. But it also seems much more promising for the future.

Our current implementation of ontology-based databases are mainly based on the PLIB model with mapping onto this model of other ontology-based data [7]. We are now developing layered implementations [10] based on the CID model we have proposed and where various ontology models may cooperate. At the data level, all the context representation mechanisms are implemented together with a canonical data model. In the above layer, some concept equivalence operators from OWL and FLIGHT are implemented, providing for some ontology-level reasoning. In the upper layer, linguistic access is provided, in particular using ontology model-independent query language [17]. We are also further developing the PLIB model, adding integrity constraints and UML/XML [29] view over this model.

Acknowledgements

The author would like to thank the anonymous referees who provided helpful and valuable comments on an earlier version of this paper. The research reported here was supported in part by EU Project Esprit 8984 and IST-1999-12238 and by ANR grant 05RNTL02706.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The description logic handbook. Cambridge University Press, Cambridge (2003)
2. Bellatreche, L., Dung, N.X., Pierra, G., Dehainsala, H.: Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases. Computers in Industry 57(8-9), 711–724 (2006)
3. Bernstein, P.A., Havely, A.Y., Pottinger, R.A.: A vision of managament of complex models. SIGMOD Record 29(4), 55–63 (2000)
4. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: Contextualizing ontologies. Journal of Web Semantics 1(4), 325–343 (2004)
5. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: A principled approach to data integration and reconciliation in data warehousing. In: DMDW 1999. Proceedings of the Intl. Workshop on Design and Management of Data Warehouses, Heidelberg, Germany (June 14-15, 1999)
6. Connolly, D., Stein, L., McGuinness, D.: Daml-ont initial release (2000), `www.daml.org/2000/10/daml-ont.html`
7. Dehainsala, H., Pierra, G., Bellatreche, L.: OntoDB: An ontology-based database for data intensive applications. In: Kotagiri, et al. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 497–508. Springer, Heidelberg (2007)
8. Elmagarmid, A., Rusinkiewicz, M.: Heterogeneous Autonomous Database Systems. Morgan Kaufmann, San Francisco (1999)
9. Everett, J.O., Bobrow, D.G., Stolle, R., Crouch, R.S., de Paiva, V., Condoravdi, C., van den Berg, M., Polanyi, L.: Making ontologies work for resolving redundancies across documents. Communication of ACM 45(2), 55–60 (2002)
10. Fankam, C., Aït-Ameur, Y., Pierra, G.: Exploitation of ontology languages for both persistence and reasoning purposes: Mapping PLIB, OWL and flight ontology models. In: WEBIST 2007. Proc. of Third International Conference on Web Information Systems and Technologies, pp. 254–262 (2007)
11. Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F: Oil: an ontology infrastructure for the semantic web. IEEE Intelligent Systems 16(2), 38–45 (2001)
12. Goasdoué, F., Lattès, V., Rousset, M.C.: The use of carin language and algorithms for information integration: The picsel system. International Journal of Cooperative Information Systems (IJCIS) 9(4), 383–401 (2000)
13. Goh, C.H., Madnick, S.E., Siegel, M.: Context interchange: Overcoming the challenges of large-scale interoperable database systems in a dynamic environment. In: CIKM 1994. Proceedings of the Third International Conference on Information and Knowledge Management, pp. 337–346 (December 1994)
14. Goh, C.H., Bressan, S., Madnick, E., Siegel, M.D.: Context interchange: New features and formalisms for the intelligent integration of information. ACM Transactions on Information Systems 17(3), 270–293 (1999)
15. Gruber, T.: Toward principles for the design of ontologies used for knowledge sharing in formal ontology. In: Guarino, N., Poli, R. (eds.) Conceptual Analysis and Knowledge Representation, Kluwer Academic, Dordrecht (1993)
16. Guarino, N., Welty, C.A.: Evaluating ontological decisions with ontoclean. Communications of the ACM 45(2), 61–65 (2002)

17. Jean, S., Aït Ameur, Y., Pierra, G.: Querying ontology based databases using ontoql (an ontology query language). In: ODBASE, pp. 704–721 (2006)
18. Jean, S., Pierra, G., Ameur, Y.A.: Domain ontologies: A database-oriented analysis. In: Filipe, J., Cordeiro, J., Pedrosa, V. (eds.) WEBIST (Selected Papers). Lecture Notes in Business Information Processing, vol. 1, pp. 238–254. Springer, Heidelberg (2006)
19. Kashyap, V., Sheth, A.P.: Semantic and schematic similarities between database objects: A context-based approach. VLDB Journal 5(4), 276–304 (1996)
20. McGuinness, D.L., Harmelen, F.: OWL web ontology language overview. W3C Recommendation (February 10, 2004)
21. Mena, E., Kashyap, V., Illarramendi, A., Sheth, A.P.: Managing multiple information sources through ontologies: Relationship between vocabulary heterogeneity and loss of information. In: Proceedings of Third Workshop on Knowledge Representation Meets Databases (August 1996)
22. Minsky, M.: Matter, mind and models. International Federation of Information Processing Congress 1, 45–49 (1965)
23. Mitra, P., Wiederhold, G., Kersten, M.: A graph-oriented model for articulation of ontology interdependencies. In: Zaniolo, C., Grust, T., Scholl, M.H., Lockemann, P.C. (eds.) EDBT 2000. LNCS, vol. 1777, Springer, Heidelberg (2000)
24. Noy, N.F., McGuinness, D.L.: Ontology development: A guide to creating your first ontology. Technical report ksl-01-05 and stanford medical informatics technical report smi-2001-0880, stanford Knowledge Systems Laboratory (April 2001)
25. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL web ontology language semantics and abstract syntax. W3C Recommendation (February 2004)
26. Pierra, G.: An object oriented approach to ensure portability of cad standard parts libraries. In: Eurographics 1990. Proceedings of the European Computer Graphics Conference and Exhibition, pp. 205–214 (1990)
27. Pierra, G.: A multiple perspective object oriented model for engineering design. In: New Advances in Computer Aided Design & Computer Graphics, pp. 368–373. International Academic Publishers, Beijing (1993)
28. Pierra, G., Dehainsala, H., Aït-Ameur, Y., Bellatreche, L.: Base de données à base ontologique: principe et mise en oeuvre. Ingénierie des systèmes d'information 10(2), 91–115 (2005)
29. Pierra, G., Sardet, E.: Proposal for a XML representation of the PLIB ontology model: Ontoml. Research Report RR 07-01, p. 188 (2007), `http://www.lisi.ensma.fr/ftp/pub/documents/reports/2007/2007-LISI-2007-01.pdf`
30. Schenck, D., Wilson, P.: Information modelling: The express way. Oxford University Press, Oxford (1994)
31. Sciore, E., Siegel, M., Rosenthal, A.: Using semantic values to facilitate interoperability among heterogeneous information systems. ACM Transactions on Database Systems 19(2), 254–290 (1994)
32. Stuckenschmidt, H., Vögele, T., Visser, U., Meyer, R.: Intelligent brokering of environmental information with the buster system. In: Proceedings of the 5th International Conference "Wirtschaftsinformatik", Physica-Verlag, pp. 15–20 (2001)
33. Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-based integration of information - a survey of existing approaches. In: Proceedings of the International Workshop on Ontologies and Information Sharing, pp. 108–117 (August 2001)

34. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer 25(3), 38–49 (1992)
35. Ziegler, P., Dittrich, K.R.: User-specific semantic integration of heterogeneous data: The sirup approach. In: Bouzeghoub, M., Goble, C.A., Kashyap, V., Spaccapietra, S. (eds.) ICSNW 2004. LNCS, vol. 3226, pp. 44–64. Springer, Heidelberg (2004)

A Annex: PLIB-Related Standards

Some standard numbers are quoted throughout the paper. Formal designations and some descriptions of these standards may be found at :

- *http://www.iso.org/iso/en/CatalogueListPage.CatalogueList*;
- *http://www.iec.ch*;
- *http://www.plib.ensma.fr* .

This annex gives a short overview of international standardization activities around PLIB.

One may distinguish four categories of standards.

A.1 Ontology Model

The standard ontology model was developed as a joint effort of ISO (International Organization for Standardization) and IEC (International electro technical commission) and published as the ISO 13584 and IEC 61360 standard series. The ontology model was first published in ISO 13584-42 and IEC 61360-2, as an EXPRESS specification, further extended in ISO 13584-25. A new edition is currently in process. An UML / XML self-contained view of the model, called OntoML (ISO 13584-32), is currently under ballot. Both should be published in 2008.

A.2 Ontology-Based Data

Capability to model and to exchange real world objects as ontology individuals (e.g., electronic catalogues, ontology-based database content) was specified in some other parts of ISO 13584, mainly part 20, 24 and 25 that provide both for static description (i.e., property value pairs) and dynamic behavioral description by meta-modelling of expressions and functions.

A.3 Methodological Aspect

Over the last two years, a guide for using PLIB ontology model for specification of product properties and classes was developed. It will be published as ISO/IEC Guide 77 in 2007, and recommended for use by all ISO and IEC product standardization committees.

A.4 Standard Ontologies

Several standard domain ontologies have been developed or are currently under development. Some of them are associated with maintenance agencies allowing to update continuously these ontologies. Examples of already standardized domain ontologies include : Electronic Components (IEC 61360-4), Laboratory Measuring Instruments (ISO 13584-501), Machining Tools (ISO 13399), Mechanical Fasteners (ISO 13584-511). Examples of domain ontologies under development include: Optics and Optronic (ISO 23584), Bearing (ISO 23768).