Ontology Evolution and Source Autonomy in Ontology-based Data Warehouses

Dung Nguyen Xuan* Ladjel Bellatreche* Guy Pierra *

* LISI/ENSMA - Poitiers University
1 Avenue Clément Ader 86960 Futuroscope - France
(nguyenx, bellatreche, pierra)@ensma.fr

Abstract. Ontology-based integration systems (OBIS) use ontologies in order to describe the semantic of sources and to make the content explicit. Two major architectures of OBISs are available: (i) those using an unique ontology, and (ii) those using multiple ontologies. In the first architecture, all sources are related to one shared ontology. This architecture suffers from changes of ontology and sources which can affect the conceptualization of the domain represented in the ontology. Any change in the ontology may affect the sources. Therefore, sources are not really autonomous. In hybrid ontologies, each source is described by its own ontology, called local ontology. Each one references/maps a shared ontology in order to guarantee that each source shares the same vocabulary. The articulation between local ontologies and the shared ontology can be done either a posteriori or a priori. Two major issues are raised in this architecture: (i) evolution of the shared ontology and its consequence on the integrated system, and (ii) autonomy of the ontology and the local schema of each source. In this paper, we propose an approach and a model to manage asynchronous evolution of warehouse integrated systems where the articulation is done in an a priori manner. The fundamental hypothesis of our work, called *principle of ontological continuity*, supposes that an evolution of an ontology does not make false an axiom that was previously true. This principle allows to manage each old instance using the actual ontology. Therefore, it simplifies significantly the management of the evolution process and allows a complete automation of the whole integration process. Our work is motivated by the automatic integration of catalogs of industrial components in engineering databases. It has been validated by a prototype using ECCO environment and EXPRESS language.

1 Introduction

It is widely recognized that an automatic integration of heterogeneous data sources is one of the keys to improve management and productivity of several application domains like data warehouse, bio-informatic, e-commerce, etc. Generally, a data integration system combines the data residing at different heterogeneous and autonomous sources, and provides an unified, reconciled view of these data, called global schema, which can be queried by the users. A

warehouse integration consists in materializing the data from multiple sources into a warehouse and executing all queries on the data contained in the warehouse rather than in the actual sources. Warehousing emphasizes data translation, as opposed to query translation in mediator-based integration Wiederhold (1992).

There is an important number of research projects dealing with data integration. The spectrum ranges from early multi-database systems Lander and Rosenberg (1982) over mediator systems (e.g., Garlic Roth et al. (1996); Wiederhold (1992)), warehouse systems Bellatreche et al. (2004a) to ontology-based integration approaches (e.g., Observer Mena et al. (1996), Picsel Reynaud and Giraldo (2003), and a priori integration Bellatreche et al. (2004a)).

Note that each integration system addresses the following issues: (i) the source autonomy, known as *the receiver heterogeneity problem* Goh et al. (1994), and (ii) resolution of various conflicts from multiple sources due to semantic heterogeneities. Among these conflicts we can cite Goh et al. (1999): *naming conflicts*, *scaling conflicts*, *confounding conflicts* and *representation conflicts*.

Source autonomy: Sources are not forced to adapt their data and their semantics to integrate sources Goh et al. (1994). Therefore, most of the sources operate autonomously: i.e., they are free to modify their ontology and/or schema, remove some data without any prior "public" notification, or occasionally block access to the source for maintenance or other purposes. Moreover, they may not always be aware of or concerned by other sources referencing them or integration systems accessing them. Consequently, the relation between the integrated system represented by a warehouse and its autonomous sources is slightly coupled. Consequently, it may generates maintenance anomalies Chen et al. (2004). In that context (where changes are asynchronous), evolution management concerns schema and populations of sources, and the shared and local ontologies.
The problem of source autonomy has received a little attention in the literature, especially in the context of ontology-based integration systems compared to the first issue (conflict resolution).

Conflict Resolution: The fundamental challenge of the source integration systems is the difficulty to integrate automatically, at the meaning level, multiple sources Levy et al. (1996), Doan et al. (2004), Lawrence and Barker (2001), Chawathe et al. (1994), Reynaud and Giraldo (2003) and Castano and Antonellis (1997). By exploring the existing integration systems, two generations of integration are distinguished: In the first generation of integration systems (e.g., TSIMMIS Chawathe et al. (1994)), data meaning was not explicitly represented. Thus, the meaning of concepts and the mappings between concepts were manually encoded in a view definition Chawathe et al. (1994) (SQL view).

Systems in the second generation use ontology and in particular a shared ontology to map meaning of each data source Wache et al. (2001). Ontologies provide a way to represent explicitly the formal semantics. These systems are called ontology-based integration systems (OBISs). An ontology is "an explicit specification of a conceptualization" Gruber (1995). A classical definition of an ontology is an explicit, and formal shared and referencable description of concepts and their relationships that exist in a certain universe of discourse. References to an ontology provides a shared vocabulary that labels concepts of a domain. Ontologies are then used in an integration task to describe the semantic of the sources and to make the content

explicit. Two main architectures of OBISs are available Wache et al. (2001): (i) those using an unique ontology, and (ii) those using multiple ontologies.

1. In the unique ontology architecture, all information sources are related to one shared ontology. A prominent approach of this kind of architecture is SIMS Arens and Knoblock (1993). This architecture suffers from changes of ontology and sources which can affect the conceptualization of the domain represented in the ontology. Any change in the ontology may affect the sources. Therefore, sources are not really autonomous.

2. In hybrid ontologies, each source is described by its own ontology, called local ontology. Each one references a shared ontology in order to guarantee that each source shares the same vocabulary Mitra et al. (2000); Hakimpour and Geppert (2002); Goh et al. (1999). The articulation (or linkage) between local ontologies and the shared ontology (each concept of a local ontology O_i is mapped into a concept of the shared ontology) can be done either a posteriori Mitra et al. (2000) or a priori Bellatreche et al. (2004a). In a posteriori articulation it is hard to ensure a fully automatic integration process Mitra et al. (2000).

   In a number of domains, including Web service, e-procurement, synchronization of distributed databases, the new *challenge* is to perform a fully automatic integration of autonomous sources.

   Claim 1 *In order to avoid a human-controlled mapping between concepts at integration time, this mapping shall be done* a priori *at the database design time.*

   This means that some formal shared ontologies must exist, and each local source shall embed some ontological data that references explicitly this shared ontology. Some systems are already developed based on this hypothesis: Picsel2 Reynaud and Giraldo (2003) project for integrating Web services, the COIN project for exchanging for instance financial data Goh et al. (1999). Their weakness is that once the shared ontology is defined, each source shall used the common vocabulary. The shared ontology is in fact a global ontology and each source is less autonomous.

Two major issues are raised in the multiple ontologies architecture: (i) the evolution of the shared ontology and its consequence on the integrated system, and (ii) the schematic autonomy of the local ontology and the local schema of each source.

Our work deals with an integration of autonomous and asynchronous data sources, where each one has its own ontology referencing a shared one Bellatreche et al. (2004b). In this situation, it may be impossible to find a local ontology similar to the shared one. This is because of two reasons: (i) the first one concerns the covered domain (autonomy of domains of each source), and (ii) the second one concerns the synchronism (asynchronous evolution):

1. generally, the covered domain is not the same:

   - each local source references the shared ontology, but it takes just a fragment of that ontology,
   - conversely, usually, some shared concepts have to be refined locally in order to represent the local requirements. Our goal is to ensure a maximal autonomy of sources, and provides an automatic integration.

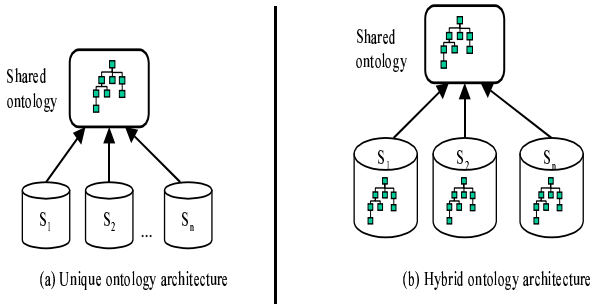

FIG. 1 – *The two possible ways for using ontologies for content explication.*

2. Ontologies evolve, and it is impossible to completely synchronize them. This is due to following factors:
   - the evolution of the shared ontology and existing data: one source may disparate, and we want to save the old data,
   - the time needed to broadcast evolution among local and shared ontologies.

We have already proposed a solution for the problem of domain autonomy, called *semantic integration approach by articulating ontologies* Bellatreche et al. (2004a). It consists in giving to each source the autonomy on defining its own ontology, but, it should (the local ontology) references systematically shared ontology(ies) (the notion of smallest subsumes class reference Bellatreche et al. (2004b)).

In this paper, we propose an approach and model dealing with the problem of management of asynchronous evolution of ontologies and data. Our approach is based on the particular characteristics of ontologies (contrary to conceptual model), which consists in defining a set of constraints to be respected simultaneously by the shared ontology and all the data sources participating in the integration process.

2 Problem Position

Let's consider a warehouse integrated system described in Figure 2. Let O be the shared ontology, and $S = \{S_1, ..., S_n\}$ a set of data sources participating in the integration process, where each source S_i ($1 \leq i \leq n$), stores explicitly its local ontology referencing the shared one O (this reference is done a priori), and the mappings between the local ontology and the data. This kind of sources, is called, *ontology-based data source* (OBDS) Pierra et al. (2004). The warehouse integrated system has also an OBDS structure. The data warehouse ontology may be the shared one.

In several situations, members of a community defining an ontology (resulting from a consensus) want to extend it. This extension may refine some already modeled concepts or to acknowledge an evolution in the conceptualized world. For instance, in a product ontology, the fact that a new kind of products have been appeared on the market. In this case, we cannot assume that each source will immediately reflect the changes in the shared ontology.

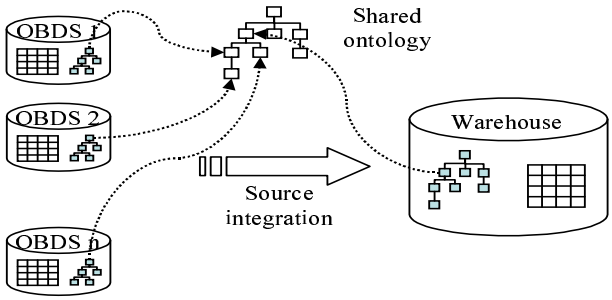

FIG. 2 – *Semantic integration by a priori ontology articulation.*

In this context, where changes occur in a asynchronous manner across the various sources and the shared ontology, *three problems* should be resolved:

1. *The traceability of two links between local ontologies and the shared ontology*: local ontologies define the meaning of local data. Moreover, when a local ontology references a shared ontology, the meaning of the local ontology entries are themselves imported from the shared one. Therefore, if the shared ontology evolves, it should contribute with a particular release to the definition of local data. This problem may be solved by assigning a version number to each concept of the shared ontology, and by increasing this number when some changes occur in the concept definition Noy and Klein (2003); Klein and Noy (2003).

2. *The data access transparency*: evolution of the shared ontology must not destroy the integration system: a single version of the shared ontology must provide an access to all integrated data, despite the fact that they might correspond to different versions of the shared ontology.

3. *The management of life cycle of instances:* in some situations it will be interesting to know at each moment the existence of each instance. This requires to save all versioned instances of all tables. This problem is known as "schema versioning" in the literature Wei and Elmasri (1999). But two difficulties exist: (i) the replication of data (maintenance and storage overhead) et (ii) the conflict between the automatic refreshment and the query processing on multi-versioned data Wei and Elmasri (1999).

Example 1 *To illustrate these problems, let consider the following example:*

Figure 3 shows a warehouse integrating two ontology-based sources, $Source1$ *and* $Source2$*, where each one references a shared ontology. Suppose that at instant* t*, the version of the local and shared ontologies is 1. At instant* $(t+1)$*, the shared ontology and the* $Source_2$ *endure the following asynchronous evolutions:*

1. *The shared ontology evolves independently from sources, where its class* C *changes and its version becomes 2,*

2. *The* $Source_2$ *has a change in the ontology level, where a new attribute is added to class* C_2*. Consequently, the class* C_2 *and its sub class* C_{21} *have the version 2,*

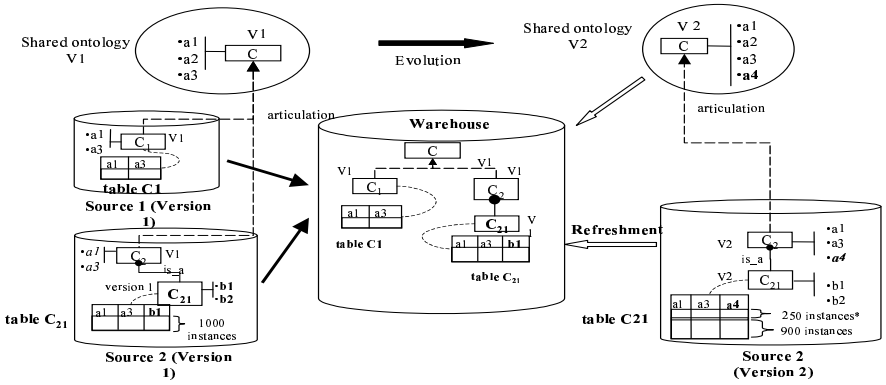

FIG. 3 – *Evolution examples.*

3. *Two different changes are done on the table corresponding to the sub class C_{21} on content and schema levels, respectively: (1) insertion and deletion operations (750 instances are deleted et 900 are inserted), and (2) the attribute b_1 is deleted and a_4 is added.*

These evolution scenarios of sources and ontologies cause the following problems:

1. *How to integrate the $Source_2$ of version 2 which does not reference the last version of the shared ontology in the warehouse?*
2. *Suppose that an user wants to store the class C of version 2 in the warehouse. In this case, is it possible to access data of $Source_1$ using the class C of version 2 [1]? How the link between the class C with version 2 and $Source_1$ is established?*
3. *Suppose that we want to store all data of two versions of table $T_{C_{21}}$ in the warehouse. Therefore, how we dial with replicated data and its maintenance?*

In the following sections, we give answers to these problems.

3 Notions of the semantic integration method using an a priori articulation of ontologies

In this section, we present formally our approach for an automatic integration process based on a priori articulation between local ontologies and a shared ontology. In the next section, this model will be used to present our proposal concerning both ontology evolution and source autonomy.

3.1 Basic concepts

Formally, an ontology is defined as the 4-tuples $O < C, P, Sub, Applic >$, with:

[1] *note that the articulation between $Source_1$ and the shared ontology is represented by the relation between the class C and the class C_1 all of version 1*

- C is the set of the classes used to describe the concepts of a given domain (like travel service Reynaud and Giraldo (2003), equipment failures, electronic components Pierra et al. (2003) etc.).
- P is the set of properties used to describe the instances of the C classes. Note that it is assumed that P defines a much greater number of properties that are usually represented in a database. Only a subset of them might be selected by any particular database [2].
- Sub is the subsumption function defined as $Sub : C \rightarrow 2^C$ [3], where for a class c_i of the ontology it associates its direct subsumed classes [4]. Sub defines a partial order over C.
- Finally, $Applic$ is a function defined as $Applic : C \rightarrow 2^P$. It associates to each ontology class those properties that are applicable (i.e., rigid) for each instance of this class. Applicable properties are inherited through the is-a relationship and partially imported through the case-of relationship, i.e, $\forall c_s, c \in C, c_s \in Sub(c) \Rightarrow Applic(c) \subset Applic(c_s)$.

Example 2 *To illustrate the case-of relationship, let's consider Figure 4 showing an extension of the shared ontology Hard Disk using the case-of relationship. The local ontology* External Disk *of S_2 imports some properties of the shared one (Model Code, Capacity, Interface, Transfer rate, etc.). Note that this local ontology does not import some properties of the shared ontology like Rotational rate, Data buffer, etc. To satisfy its needs, it adds other properties describing its External Disk like Read Rate, Write Rate, Marque and Price. This particular relationship is a key mechanism allowing each source both to make its own extensions and to exchange information with other actors referencing the same shared ontology.*

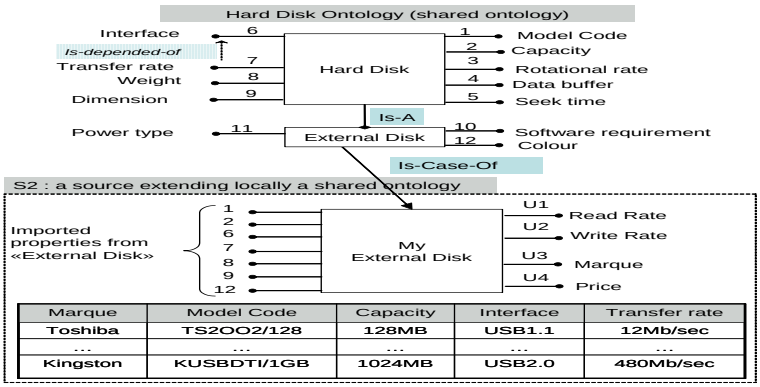

FIG. 4 – *Case_of and IS_A relationships.*

To associate instances with an ontology, we define the three following constraints referenced to as the ontology-instance's strong typing assumptions Pierra et al. (2004); Jean et al. (2005):

[2] a particular database may also extend the P set
[3] We use the symbol 2^C to denote the power set of C.
[4] C_1 subsumes C_2 iff $\forall x \in C_2, x \in C_1$.

1. **R1-** We assume that the set of classes to which an instance belongs to are ordered by the subsumption relationship has an unique minimal class (called *instance basis class*).

2. **R2-** Each ontology specifies for each of its classes, those properties that are allowed for use in a class instance (applicable properties).

3. **R3-** Each object can be described only by applicable properties of its basis class ($Applic(c)$, where c is the basic class of the considered instance).

On the basis of these assumptions, an ontology-based data source (OBDB) may be formally defined as 4-tuples $< O, I, Sch, Pop >$, where:

- O is an ontology ($O :< C, P, Sub, Applic >$);
- I is the set of instances of the source; each one represented as an instance of its basis class;
- $Pop : C \rightarrow 2^I$, associates to each class (leaf class or not) its own instances (polymorph), that consists of instances of which this class, or any of its subsumed classes, is the basic class.
- $Sch : C \rightarrow 2^P$ associates to each ontology class c_i of C the properties which are effectively used to describe the instances of the class c_i; Sch has two definitions based on the nature of each class (a leaf or a no-leaf class). For each class c_i, Sch(c_i) shall satisfy the following: $Sch(c_i) \subseteq Applic(c_i)$.

Note that Sch and Pop are two functions linking the ontology part and data part in an OBDS. For each leaf class of the subsumption hierarchy, i.e., each class c_i such that $Sub(c_i) = \phi$, its corresponding table T_i is such that: (i) the schema of T_i is $Sch(c_i)$, and (ii) the population of T_i is $Pop(c_i)$. For all non leave classes, c_k, $Pop(c_k)$ is union of population of all the tables associated with classes that are subsumed by c_k. $Sch(c_k)$ is the projection on $Applic(c_k)$, of the union of schemas of all classes subsumed by c_k [5].

3.2 An a Priori Integration Approach

Let $O :< C, P, Sub, Applic >$ the shared ontology and $S = \{S, ..., S_n\}$ the set of OBDSs participating in the integration process. In an a priori integration, we first integrate ontologies, and then data Bellatreche et al. (2004b). Integration of ontologies is done using subsumption relationships between classes of different ontologies and by "importation" of properties from the subsuming class to the sub summed class. We assume then that each source S_i has been designed following two steps Bellatreche et al. (2004b,a):

1. The source administrator has to define its own ontology: $O_i : < C_i, P_i, Sub_i, Applic_i >$. But we assume that that each local ontology class c_l references by a case-of relationship, its smallest (w.r.t sub order) subsuming class c_g in the shared ontology. We also

[5] in the presentation of $Pop(c_k)$, a property of $Sch(c_k)$ which is not represented for a sub class, will be , for a particular need, valued to null. This allows to integrate sources do not having the same choice about the properties of the ontology that are not used effectively for each class

assume that through this case-of relationship, c_l "imports" from c_k all the properties of $Applic(c_g)$ that the source administrator wants to use in the context of its own source. These mappings are stored in the source S_i and constitute an *articulation* M_i between O and O_i that defines a function: $M_i : C \rightarrow 2^{C_i}$, where $M_i(c)$ is the set of classes of C_i directly subsumed by $c \in C$,

2. The administrator chooses for each class c_i: (i) its schema $Sch_i(c_i)$, and (ii) its instances $Pop_i(c_i)$ [6].

A source S_i articulated with the shared ontology O can thus be formulated as follows: $S_i :< O_i, I_i, Sch_i, Pop_i, M_i >$. The integration system is also an OBDS structure: $DW :< O_{DW}, I_{DW}, Sch_{DW}, Pop_{DW} >$. We assume that in the integration system, each instance of each particular source is represented as an instance of its smallest subsuming class in the shared ontology. This means that each class of S_i references (directly or by inheritance) its smallest subsumed class(es) of the shared ontology. It (the class of S_i) imports from the subsumed classes of the shared ontology the relevant properties. All classes and properties of S_i that are not identified to the imported ones are specific to the source S_i.

1. O_{DW} is computed as an integration of the local ontologies into the shared one.

   (a) $C_{DW} = C \cup (\cup_{1 \leq i \leq n} C_i)$,

   (b) $P_{DW} = P \cup (\cup_{1 \leq i \leq n} P_i)$,

   (c) $Applic_{DW}(c) = \begin{cases} Applic(c), & \text{if } c \in C \\ Applic_i(c), & \text{if } c \in C_i \end{cases}$

   (d) $Sub_{DW}(c) = \begin{cases} Sub(c) \cup M_i(c), & \text{if } c \in C \\ Sub_i(c), & \text{if } c \in C_i \end{cases}$

2. $I_{DW} = \cup_{1 \leq i \leq n} I_i$,

   The instances are stored in tables as in their sources.

   (a) $\forall c_i \in C_{DW} \wedge c_i \in C_i \wedge \forall i \in [1..n]$:

   - $Sch_{DW}(c_i) = Sch_i(c_i)$, and
   - $Pop_{DW}(c_i) = Pop_i(c_i)$,

   (a) $\forall c \in C$

   - $Sch_{DW}(c) = Applic(c) \cap (Sch(c) \cup (\cup_{c_j \in Sub_{DW}(c)} Sch(c_j)))$,
   - $Pop_{DW}(c) = \cup_{c_j \in Sub(c)} Pop(c_j)$

[6] for non leaf classes, these choices should respect constraints resulting from the polymorphism: $\forall c_i \in Sub(c_k) : Pop(c_i) \subset Pop(c_k), Sch(c_i) \cap Applic(c_k) \subset Sch(c_i)$

4 Constraints for evolving warehouse ontology-based integration systems

4.1 The Principle of Ontological Continuity

The constraints that we should define in order to handle the evolution of warehouse ontology-based integration systems result from the fundamental differences existing between the evolution of conceptual models and ontologies. According to Minsky, a conceptual model is a object allowing to respond questions on another object, named the modeled domain Minsky (1979). When the questions change (when the organizational objectives are modified), its conceptual model is modified too, without broadcasting the information that the modeled domain is modified. Contrary to conceptual models, an ontology is a conceptualization aiming to represent entities of a particular domain in consensual form for a community. It is a logic theory of a part of the world, shared by the whole of the community, and allows to their members to understand each others. That can be, for example, the set theory (for mathematicians), mechanic (for mechanics) or analytical counting (for accountants). For this type of ontologies, two changes should be identified: *normal evolution*, and *revolution*. A normal evolution of a theory is its deepening. New truths, more detailed are added to the old truths. What was true yesterday remains true today. Concepts are never deleted contrary to Maedche et al. (2002). But it may be possible that axioms of a theory become false. In this case, It is not any more an evolution but a *revolution*, where two different logical systems will coexist or be opposed.

The ontologies that we consider correspond to this philosophy. They are ontologies either standardized, for example at the international level, or defined by significant consortium which formalize in a stable way knowledge of a technical domain. The changes in which we are interested in our approach are those representing evolution of the axioms of an ontology and not revolution.

Therefore, we thus impose to all manipulated ontologies (local and shared) to respect the following principle:

Principle of ontological continuity: *if we consider that each ontology of the integrated system as a set of axioms, then any true axiom for a certain versions of ontology will remain true for all the later versions.*

4.2 Constraints on the Evolution of Ontologies

In this section, we show the constraint on each concept (classes, relation between classes, properties and instances) of an ontology. Let $O^k =< C^k, P^k, Sub^k, Applic^k >$ be the ontology with version k.

4.2.1 Permanence of the classes

The existence of a class could not be disapproved in a later stage: $C^k \subset C^{k+1}$. To take into account the reality, it will be relevant to consider it obsolete. It will then be marked as ("defecated"), but will continue to form part of the later versions of the ontology. In addition, the definition of a class could be refined without inferring the membership of form instance to that class. This is means that:

- The definition of classes itself may evolve,
- Each class definition will be associated to a version number.

4.2.2 Permanence of properties

Similarly $P^k \subset P^{k+1}$. A property may become obsolete while the existing value of that property remains undisputed. Similarly, a definition or a domain values of a property may evolve. Taking into account the ontological principle of continuity, the domain of values could be only increasing, certain values may eventually marked as obsolete.

4.2.3 Permanence of the Subsumption

Subsumption is also an ontological concept which could not be infirmed. Let $Sub^* : C \rightarrow 2^C$ be the transitive closure of the direct subsumption relation Sub. We have then:
$\forall C \in C^k, Sub^{*k}(c) \subset Sub^{*k+1}(c)$.
This constraint allows obviously a refreshment of the hierarchy of subsumption of the classes, for example by intercalating intermediate classes between two classes linked by a relation of subsumption.

4.2.4 Description of instances

The fact that a property $p \in Applic(c)$ means that this property is rigid for each instance of c. This is an axiom that cannot be infirmed:
$\forall c \in C^k, Applic^{*k}(c) \subset Applic^{*k+1}(c)$.

Note that this does not suppose that same properties are always used to describe the instances of the same class. It is not a question of an ontological characteristic but only of a schematic nature.

4.3 Identification of Different Elements

Evolution management supposes the power to indicate and thus identify all the evolved elements.

4.3.1 Identification of classes and properties

We already specified that any class and any property were associated universal identifiers (GUI). In fact these identifiers contain two parts: a code (unique) and a version (integer): *GUI: = version code*.

Any reference between elements using the GUI is itself versioned by the versions of its *extremities*. Finally, any definition of class or property contains in particular the date from which this version is valid.

Note that a class has three types of evolution: (1) an ontological evolution (for example, modification of the definition or augmentation of the applicable properties), (2) a schematic evolution (more or less properties used to describe the instances), and (3) evolution of its population (insertion and deletion of instances). For reasons of simplicity, these three types of

changes is ensured by incrementing the same indicator of the version. A version thus characterizes a definition, a schema and a population.

4.3.2 Identification of instances

In order to recognize at the maintenance time of the warehouse, any source must define for each class having a population a *semantic key*. This key is constituted by the representation (in character string form) of one or several values of applicable properties of this class. The properties are always provided for each instance of the class and will have never to be modified. The other properties could or not be provided according to choice's (with each version) of the schema of the instances of the class. Their values will not have, on the other hand, to be modified from one version to another.

The life cycle of an instance (appearance and disappearance) is then defined by the versions of the classes to which it belongs.

5 Floating Version Model

Our management model has several objectives: (i) to allow a total access to the whole of the existing instances of the warehouse via shared ontology, (ii) to know the history of the instances, and (iii) possibly, to know, for each instance, which version of ontology it corresponds. We describe below successively how we can reach the three aspects above defined.

5.1 Management of Updates

We suppose that our warehouse is refreshed in the following way. At given moments chosen by the data warehouse administrator, the current version of a source S_i is integrated in the warehouse. It includes its ontology, its references to shared ontology, and its contents *(certain instances were eventually already integrated, others are new, others are removed)*. This scenario corresponds, for example, in the engineering domain with a warehouse consolidating descriptions of components of a whole of suppliers. A maintenance is carried out each time that a new version of an electronic catalogue of a supplier is received.

5.2 A Global Access to Current Instances

We call current instances of the warehouse the instances resulting from most recent maintenance from each source. The principal difficulty, resulting from the autonomy of each source, is that during two successive maintenance done by two different sources, the same class of shared ontology c can be referred by an articulation of subsumption (see section 3.2) in different versions. For example c^k and c^{k+j} by two classes c_i^n and c_j^p. According to ontological principle of continuity, it is advisable to note that:

1. All applicable properties in c^k are also applicable in c^{k+j},
2. All subsumed classes by c^k are also subsumed by c^{k+j},

The relation of subsumption between c^k and c_i^n could be replaced by a relation of subsumption between c^{k+j} and c_i^n. The class c^k is not thus necessary to reach the instances of c_i^n. This remark leads us to propose a model, called *model of the floating versions*, which enables us to reach the data via only one version of the warehouse ontology. This version, called *"current version"* of the warehouse ontology, such as the current version of each one of its classes c^f is higher or equal to the largest version of that class referred by an articulation of subsumption at the time of any maintenance.

In practice, this condition is satisfied as follows:

- If an articulation M_i references a class c^f with a version lower than f, then M_i is updated in order to reference c^f,
- If an articulation M_i references a class c^f with a version greater than f, then the warehouse load the last version of the shared ontology and migrates all references M_i $(i = 1..n)$ to new current versions.

Example 3 *During the maintenance process of a class C_1 that references the class C with version 2, the version of C in current ontology is 1. In this case, the warehouse downloads the current version of shared ontology. This one being 3, therefore the class C_1 is modified to reference the version 3 (Figure 5).*

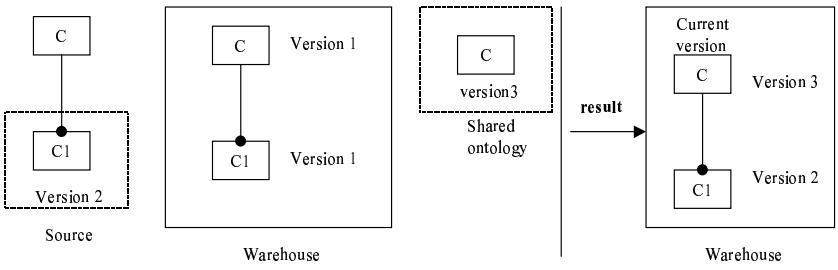

FIG. 5 – *A Model of the floating versions.*

If the only requirements of users is to know the current instances, then, at each maintenance step, the table eventually associated to each class coming from a local ontology in the warehouse is simply replaced by the corresponding current table in the local source.

5.3 Representation of history of instances

In some situations, it may be useful to know the existence of instances in the warehouse at any last moment. To do so, we do not need to archive also the versions of ontologies since the current version is compatible with all the last instances. This problem is known by "schema versioning" Wei and Elmasri (1999), where all versioned data of a table are saved. Two solutions are possible to satisfy this requirement:

- In the approach *explicit storage* Bebel et al. (2004); Wei and Elmasri (1999), all the versions of each table are explicitly stored (see Figure 6). This solution has two advantages: (i) it is easy to implement and allows an automation of the process of update of data, and (ii) query processing is straitforward in cases where we precise the versions on which the search will be done. On the other hand, the query processing cost can be very important if the query needs an exploration of all versioned data of the warehouse. Another drawback is due to the storage of the replicated data.

- In the approach *implicit storage*Wei and Elmasri (1999): only one version of schema of each table T is stored. This schema is obtained by making the *union* of all properties appearing in various versions. To each maintenance, we add the existing instance of the current table. The instances are supplemented by null values (see Figure 6). This solution avoid the exploration of several versions of a given table. The major drawbacks of this solution are: (i) the problem of replicated data is always present, (ii) the implementation is more difficult than the previous one concerning the automatic computation of the schema of stored tables; (iii) the layout of the cycle of life of data is difficult to implement (**"valid time"** Wei and Elmasri (1999)) and (iv) the semantics ambiguity of the null values.

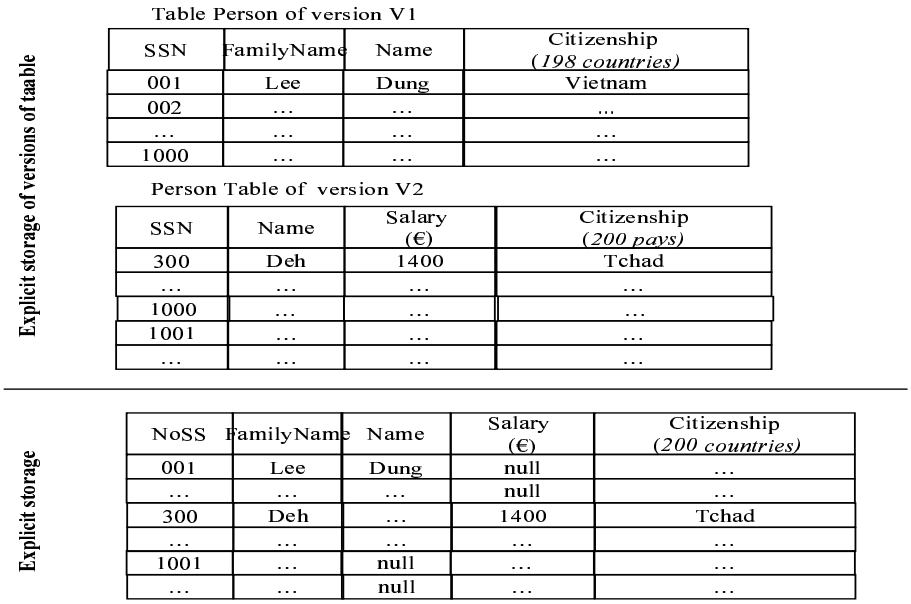

FIG. 6 – *Implicit and explicit storage.*

Our solution follows the second approach and solves the problems in the following way:

1. The problem of **replicated data** is solved thanks to the single semantic identification (value of the semantic key) of each instance of data,

2. The problem of automation of the update process of table schema is solved through the use of universal identifiers (GUI) for all the properties which can result from a null value in the table of a source, or the need for supplementing each instance.

3. The problem of the representation of the cycle of life of the instances is solved by a pair of properties: $(Version_{min}, Version_{max})$. It enables us to know the validation of a given instance.

4. The problem of semantic ambiguity of the null values: is handled by archiving the functions Sch of various versions of each basic class of a table. This archive enables us to determine the true schema of version of a table, and thus the initial representation of each instance.

5.4 The Data Warehouse Structure

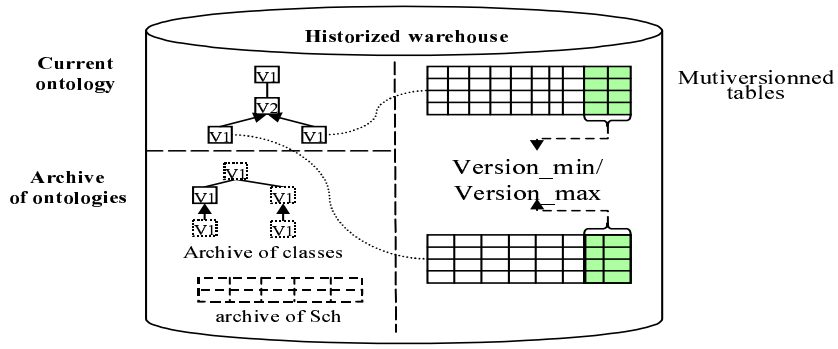

FIG. 7 – *Structure of warehouse integrated system.*

The articulation between a local ontology and shared ontology that is stored in the current version of the warehouse ontology may not be its original definition (see the Figure 5). In this case, it is necessary to reach an instance through the ontological definitions existed when this instance was itself activated. It is necessary to archive also all the versions of the warehouse ontology. It can be useful, for example, to know what was, at the time of the instance, the exact domain of its enumerated properties.

We also implemented this possibility to offer the archive in a warehouse, all versions of classes having existed in the life of the warehouse, and all the relations in their original form. Note that the principle of ontological continuity seems to make seldom necessary this complex archive. To summarize, we present the complete structure of a warehouse in a multiversioned environment (see Figure 7). This structure composed by three parts:

1. The current ontology: it contains the current version of the warehouse ontology. It represent also a generic interface to access data.

2. Ontology archive: contains all versions of each class and property of the warehouse ontology. This part gives to users the true definitions of versions of each concept if it is

necessary. Versions of schema of table T_i are also historized by archiving the function $Sch^k(c_i)$ of each version k of c_i where c_i corresponds to the table T_i .

3. multiversioned tables: contain all instances and their first and last activated versions.

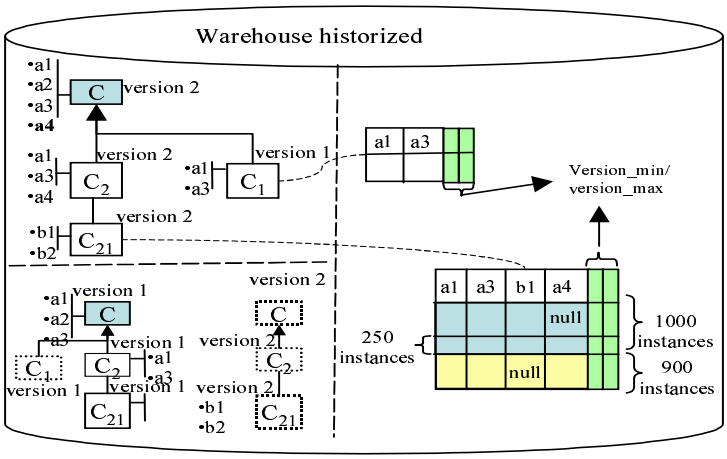

FIG. 8 – *Figure 3 after the maintenance phase.*

The result of problem of asynchronous evolution management presented in example of section 2 is illustrated in Figure 8.

6 Implementation

In order to validate our work, we have developed a prototype integrating several OBDSs (Figure 9), where ontologies and sources are described using PLIB ontology models Pierra et al. (2003)which are specified by Express language Schenk and Wilson (1994) . Such ontologies are exchangeable as instances of EXPRESS files ("physical file"). To compile EXPRESS files, the ECCO Toolkit of PDTec which offers the following main functions Staub and Maier (1997):

1. Edition and syntax and semantic checker of EXPRESS models;

2. Generation of functions (Java and C++) for reading, writing and checking integrity constraints of a physical file representing population of instances of an EXPRESS schema;

3. Manipulation of the population (physical file) of EXPRESS models using a graphical user interface;

4. Access to the description of a schema in the form of objects of a meta-model of EXPRESS;

5. Availability of a programming language called EXPRESS-C. It is an extension of the EXPRESS language. It allows managing an EXPRESS schema and its instance objects. It has two main extensions: capability to make input-output with external environments and to support event-based programming i.e., triggering a program by event occurrence.

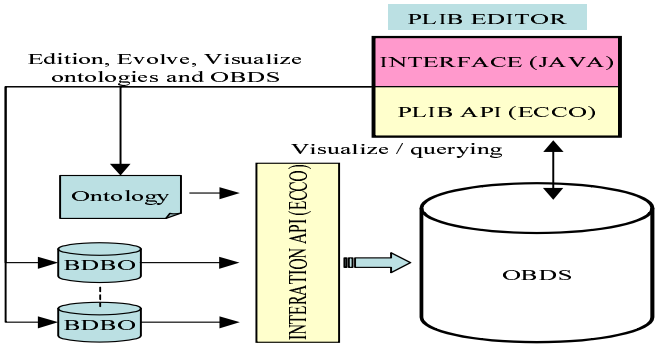

FIG. 9 – *Architecture of our Prototype.*

An ontology (or OBDS), described in file of instances of Express, is created via editor called, PLIBEditor. It is used also to visualize, edit and evolve ontologies and sources. It uses a set of PLIB API developed under ECCO. PLIBEditor proposes a QBE-like graphical interface to query the data from the ontologies. This interface relies on the OntoQL query language Jean et al. (2005) to retrieve the result of the interactively constructed queries. Figure 10 shows an ontology defining concepts of the LMD (License, Master, Doctorate) university cursus in the PLIB ontology model Pierra et al. (2003). Description of shared ontology and local ontologies is done using PLIBEditor.

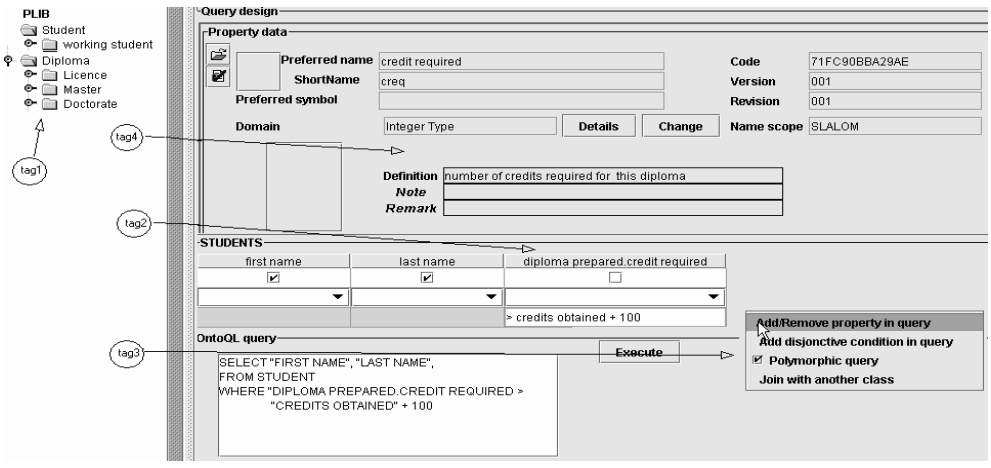

FIG. 10 – *PLIBEditor.*

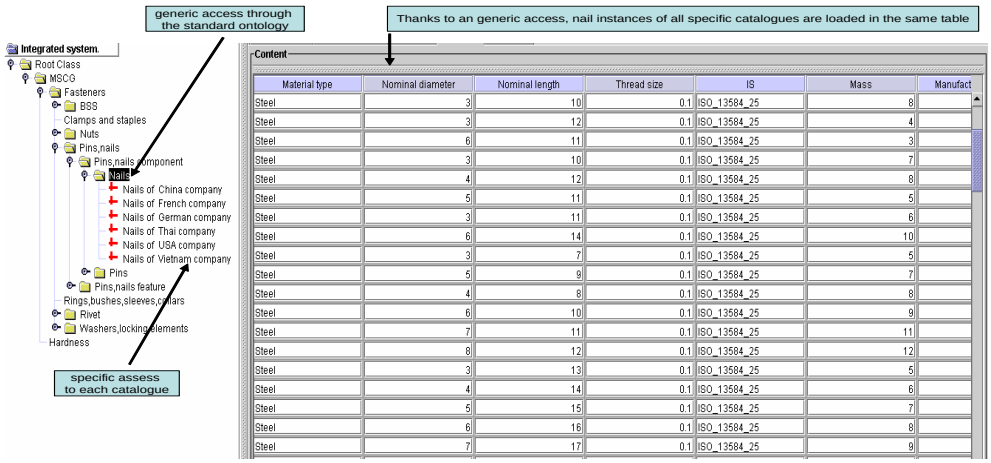

FIG. 11 – *An example of nail integrated system.*

We have developed a set of integration API allowing the management of the warehouse integrated system through a domain ontology and integrate OBDSs in that warehouse. Figure 11 shows a scenario of integrating sources modeling nails of 6 companies from China, France, Germany, Thailand, USA and Vietnam. We have also implement an ontological archiving option, allowing to store just relevant versions (chosen by the warehouse administrator). These integration APIs are incorporated in PLIBEditor.

7 Conclusion

In this article, we presented the problem of management of asynchronous evolution of the data and ontologies in a warehouse ontology-based integration systems. The sources that we considered are those containing local ontologies referencing in a priori manner a shared one. These sources are autonomous and heterogeneous. Our integration process integrates first ontologies and then the data. The presence of ontologies allows an automation of the integration process of integration by facilitating the conflict resolution. But it makes the management of autonomy of sources more difficult. This difficulty is due to the presence of a new dimension which is the ontology. To solve this problem, we presented a multi-version approach. Initially, some constraints on ontologies and the data of the sources were defined. The evolution of ontologies is carried out according to the principle of ontological continuity (an evolution of an ontology cannot cancel an axiom previously true). A structure of a ontology-based warehouse (which references also the shared ontology) is presented. It consists of three parts, namely, (1) the current ontology which contains the current version of the warehouse ontology, (2) the archive of ontologies which contains all the versions of each class and property of the warehouse ontology, and (3) the multiversioned tables containing all instances and their first and last version of activities. This structure allows the tracing the cycle of life of the instances and the data access is done in a transparent manner. Our model was validated under ECCO by considering several domain ontologies, where for each ontology, a set of sources was defined.

It would be interested to consider a mediator architecture of our proposed model and architecture, the problem of view maintenance in an ontology-based warehouse and finally, evaluation of our approach in order to measure its scalability.

References

Arens, Y. and C. A. Knoblock (1993). Sims: Retrieving and integrating information from multiple sources. *Proceedings of the International Conference on Management of Data (SIGMOD'1993)*, 562–563.

Bebel, B., J. Eder, C. Koncilia, T. Morzy, and R. Wrembel (2004). Creation and management of versions in multiversion data warehouse. *Proceedings of the 2004 ACM symposium on Applied computing*, 717–723.

Bellatreche, L., G. Pierra, D. Nguyen Xuan, H. Dehainsala, and Y. Ait Ameur (2004a). An a priori approach for automatic integration of heterogeneous and autonomous databases. *International Conference on Database and Expert Systems Applications (DEXA'04)* (475-485).

Bellatreche, L., G. Pierra, D. Xuan, and D. Hondjack (2004b). Intégration de sources de données autonomes par articulation a priori d'ontologies. *Proc. du 23ème congrès Inforsid*, 283–298.

Castano, S. and V. Antonellis (1997). Semantic dictionary design for database interoperability. *Proceedings of the International Conference on Data Engineering (ICDE)*, 43–54.

Chawathe, S. S., H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom (1994). The tsimmis project: Integration of heterogeneous information sources. *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, 7–18.

Chen, S., B. Liu, and E. A. Rundensteiner (2004). Multiversion-based view maintenance over distributed data sources. *ACM Transactions on Database Systems 4*(29), 675–709.

Doan, A., N. F. Noy, and A. Y. Halevy (2004). Introduction to the issue on semantic integration. *SIGMOD Record 33*(4).

Goh, C., S. Bressan, E. Madnick, and M. D. Siegel (1999). Context interchange: New features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems 17*(3), 270–293.

Goh, C. H., S. E. Madnick, and M. Siegel (1994). Context interchange: Overcoming the challenges of large-scale interoperable database systems in a dynamic environment. *in Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*, 337–346.

Gruber, T. (1995). A translation approach to portable ontology specification. *Knowledge Acquisition 5*(2), 199–220.

Hakimpour, F. and A. Geppert (2002). Global schema generation using formal ontologies. *in Proceedings of 21th International Conference on Conceptual Modeling (ER'02)*, 307–321.

Jean, S., G. Pierra, and Y. Ait-Ameur (2005). Ontoql: an exploitation language for obdbs. *VLDB Ph.D. Workshop*, 41–45.

Klein, M. and N. F. Noy (2003). A component-based framework for ontology evolution. *Proceedings of eighteenth International Joint Conference on Artificial Intelligence*.

Lander, T. and R. L. Rosenberg (1982). An overview of multibase. *in Proceedings of the Second Symposium of Distributed Databases*.

Lawrence, R. and K. Barker (2001). Integrating relational database schemas using a standardized dictionary. *in Proceedings of the ACM Symposium on Applied Computing (SAC)*, 225–230.

Levy, A. Y., A. Rajaraman, and J. J. Ordille (1996). The world wide web as a collection of views: Query processing in the information manifold. *Proceedings of the International Workshop on Materialized Views: Techniques and Applications (VIEW'1996)*, 43–55.

Maedche, A., B. Motik, L. Stojanovic, R. Studer, and R. Volz (2002). Managing multiple ontologies and ontology evolution in ontologging. *Intelligent Information Processing*, 51–63.

Mena, E., V. Vipul Kashyap, A. Illarramendi, and A. P. Sheth (1996). Managing multiple information sources through ontologies: Relationship between vocabulary heterogeneity and loss of information. *in Proceedings of Third Workshop on Knowledge Representation Meets Databases*.

Minsky, M. (1979). Computer science and the representation of knowledge. *in The Computer Age: A Twenty-Year View, Michael Dertouzos and Joel Moses, MIT Press*, 392–421.

Mitra, P., G. Wiederhold, and M. Kersten (2000). A graph-oriented model for articulation of ontology interdependencies. *in Proceedings of the 7th International Conference on Extending Database Technology (EDBT'00)*, 86–100.

Noy, N. F. and M. Klein (2003). Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems 5*.

Pierra, G., H. Dehainsala, Y. A. Ameur, L.Bellatreche, J. Chochon, and M. Mimoune (2004). Base de Données à Base Ontologique : le modèle OntoDB. *Proceeding of Base de Données Avancées 20èmes Journées (*BDA*'04)*, 263–286.

Pierra, G., J. C. Potier, and E. Sardet (2003). From digital libraries to electronic catalogues for engineering and manufacturing. *International Journal of Computer Applications in Technology (IJCAT) 18*, 27–42.

Reynaud, C. and G. Giraldo (2003). An application of the mediator approach to services over the web. *Special track "Data Integration in Engineering, Concurrent Engineering (CE'2003) - the vision for the Future Generation in Research and Applications*, 209–216.

Roth, M. T., M. Arya, L. Haas, M. Carey, W. Cody, R. agin, P. Schwarz, J. Thomas, and E. Wimmers (1996). The garlic project. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 557–557.

Schenk, D. and P. Wilson (1994). *Information Modelling The EXPRESS Way*. Oxford University Press.

Staub, G. and M. Maier (1997). Ecco tool kit - an environnement for the evaluation of express models and the development of step based it applications. *User Manual*.

Wache, H., T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner (2001). Ontology-based integration of information - a survey of existing approaches. *Pro-*

ceedings of the International Workshop on Ontologies and Information Sharing, 108–117.

Wei, H.-C. and R. Elmasri (1999). Study and comparison of schema versioning and database conversion techniques for bi-temporal databases. *Proceedings of the Sixth International Workshop on Temporal Representation and Reasoning (IEEE Computer)*.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer 25*(3), 38–49.

Résumé

Une nouvelle génération de systèmes d'intégration utilise des ontologies pour résoudre les conflits sémantiques et structurels entre les différentes sources de données participant au processus d'intégration. Ces systèmes supposent l'existence d'une ontologie partagée de domaine et que chaque source possède une ontologie locale qui référence, et éventuellement étend l'ontologie partagée. Le processus d'intégration est alors basé sur ces références qui constituent une articulation des ontologies locales et l'ontologie partagée. Notons que les sources de données sont indépendantes, chacune peut évoluer indépendamment des autres, ce qui engendre un problème d'évolution asynchrone. Dans le contexte d'une telle intégration à base ontologique, les évolutions concernent donc à la fois les ontologies, les schémas et les données. Dans cet article, nous proposons un modèle pour la gestion de l'évolution des systèmes d'intégration à base ontologique, dont le résultat est matérialisé sous forme d'un entrepôt de données. L'hypothèse fondamentale de notre travail, appelée, *le principe de continuité ontologique*, stipule qu'une évolution d'une ontologie ne peut infirmer un axiome antérieurement vrai. Ce principe permet d'interpréter toute instance représentée. En conséquence, il simplifie considérablement la gestion de l'évolution des ontologies. Ceci permet d'assurer une intégration automatique. Ce travail a été motivé par l'intégration automatique des catalogues de composants industriels dans les bases de données d'ingénierie. Il a été validé par un prototype sous un environnement ECCO et le langage EXPRESS.