Requirements Ontology and Multi-representation Strategy for Database Schema Evolution

Hassina Bounif, Stefano Spaccapietra, and Rachel Pottinger

Database Laboratory, EPFL, School of Computer and Communication Science
Lausanne, Switzerland
Data Management and Mining Laboratory, University of British Columbia
Vancouver, Canada
hassina.bounif@epfl.ch,
stefano.spaccapietra@epfl.ch, rap@cs.ubc.ca

Abstract. With the emergence of enterprise-wide information systems, ontologies have become by definition a valuable aid for efficient database schema modeling and integration, in addition to their use in other disciplines such as the semantic web and natural language processing. This paper presents another important utilization of ontologies in database schemas: schema evolution. Specifically, our research concentrates on a new three-layered approach for schema evolution. These three layers are 1) a schema repository, 2) a domain ontology called a *requirements ontology*, and 3) a multi-representation strategy to enable powerful change management. This a priori approach for schema evolution, in contrast with existing a posteriori solutions, can be employed for any data model and for both 1) design from scratch and evolution and 2) redesign and evolution of the database. The paper focuses on the two main foundations of this approach, the requirements ontology and the multi-representation strategy which is based on a stamping mechanism.

Keywords: Requirements ontology, multi-representation strategy, Schema Evolution.

1 Introduction

With the emergence of enterprise-wide information systems, the number of ontologies in semantic-driven data access and processing is increasing. For example, ontologies are crucial in semantic web and natural language processing. In addition to that, ontologies have become a valuable aid for efficient database schemas modeling and integration – they provide richer semantics than studying the schemas alone. This work investigates another area in which ontologies have a colossal potential of utilization and which is related to information systems: database schema evolution. Database schema evolution has an active research agenda due to its importance, cost to users, and the complexity of the problem. Many solutions have been proposed and much progress has been made in data structures, rules, constraints, schemata models and meta-models. We build on this work and advocate a novel approach for schema evolution: we predict potential changes, and integrate them into the schema for future use. Since predicting the exact changes that *will* occur over time is impossible, we detect the changes that *are plausible to be* carried out on a schema and are important for the database users. Our intent is to move one step towards developing multi-disciplinary and a priori approaches for database schema evolution, in contrast with existing a posteriori solutions that track changes instead of planning for them.

Our approach relies on the use of: 1) a schema repository that stores and provides to the system a set of relevant schemas and their relative versions, if any. In our case, the repository contains approximately 4000 schemas, 2) a requirements ontology that contains the changes that are plausible to be carried out on a database schema and are important for the database users and 3) a multi-representation strategy to aid powerful change management.

1.1 Problem Description

Having understood the overall motivation of schema evolution and our predictive approach, we are now ready to explain schema evolution in more detail. Intuitively, schema evolution means the ability of a schema to undergo changes over time without any loss of the extant data. However, besides managing the changes to the schema, applications and data linked to it need to be adapted as well. Changes to the schema are divided into three categories depending on their impact on the schema [1]:

1- Additive : additional semantic knowledge needs to be designed on the schema
2- Subtractive: semantic knowledge needs to be removed from the schema
3-Descriptive: the same semantic knowledge needs to be designed on the schema in a different manner.

In addition to the categories of changes that could occur on the schema over time, we need to consider the general problem of database schema evolution from two different sides, depending on the kind of solution we choose: 1) From the a posteriori solution perspective, 2) From the a priori solution perspective

1) From the a posteriori solution perspective
Historically, from this perspective, to resolve the schema evolution problem, one should take into consideration two major criteria, which are respectively [2]: a) the semantics of change, i.e. the understanding of the change that has taken place because of several reasons such as the new perceptions of the real world over time and technology development and performance strategies and b) the propagation of this change on the schema immediately or at a deferred time fixed by the database administrator. There is a posterior order in which the change must be received after by the schema and its components. Schema evolution is resolved either by versioning the original schema, by modifying it using restricted evolution primitives, by adopting views on the top of it or by refining it by accommodating the exceptional information in the database [3]. All these solutions react to changes that could occur on the schema. However, they are insufficient solutions, especially when the schema is facing complex changes. For instance, the modification approach simply modifies the schema to adhere to the new requirements. This changing of the schema without saving past information may lead to a loss of data. The versioning approach replicates the schema to save both the old and the new version. This replication avoids data loss;

however, it creates complex navigation through the different generated versions and slows down the DBMS (Database Management System). The combination solution – a solution that incorporates both existing approaches (e.g. the work presented in [4]) – avoids the above problems. Unfortunately, it also is characterized by the complexity and the onerous mechanisms to be executed. Hence, a new approach must be used; the a priori solution.

2) From a priori solution perspective
To resolve the problem of schema evolution from this perspective, one must clearly take into account these imperative criteria:

- Understand the current database structure and content
- Identify the dependencies among the current database schema, data and applications because the impact of one element on another needs to be known and accounted for before making changes on the database
- Detect potential changes that are plausible to occur on the schema
- Understand the potential future changes and new applications and identify their impacts on the current schema
- Consider the two possible cases, related to the database, that are respectively: 1) the case in which the initial database schema is not created yet and 2) the case in which the initial schema has already been created; however, it needs to be redesigned.

Compared to the previous perspective, in the a priori solution, the order of applicability of the changes has been modified. The changes are incorporated before they really occur. There is what is called an a priori order in which the potential change must be received before by the schema and its components.

1.2 Contribution and Outline of the Paper

The contributions of this paper are as follows:

1. Presentation of the predictive approach for database schema evolution, including the characteristics and the differences with other existing approaches for schema evolution
2. Presentation of the requirements ontology, including its role, construction and structure
3. Presentation of the multi-representation strategy with the two defined mechanisms views and stamping
4. Presentation of examples showing how the predictive approach works and outlining the role of both the requirements ontology and the multi-representation strategy.

The paper comprises five main sections. Section 2 presents the articulation of the predictive approach. Section 3 presents 1) the role of the requirements ontology in the predictive approach for evolution and 2) the structure and how it is built from the schema repository. Section 4 describes the multi-representation strategy and how it is used for schema evolution. Section 5 presents a motivating example to demonstrate the feasibility of the proposed approach. Section 6 is a conclusion and a summary of the important points dealt with in this paper and introduces perspectives on the future work.

2 Predictive Approach for Schema Evolution

The predictive approach for schema evolution is an innovative approach for schema evolution. In this section, we consider some of the specific characteristics that justify the decision of the selection of such approach. We begin by listing them, and then consider each one in more depth in turn:

- A priori solution (a)
- Predictive analytic solution (b)
- Proactive schema solution (c)
- Three different stages solution (d)
- Data modeling methodology independent solution (e)
- Collecting data solution (f)

a) A priori Approach

In contrast with existing posterior solutions for evolution such as modification or versioning approaches that support the evolution at evolution time, the *Predictive Approach* prepares the database schema for future use before the changes occur. The basic motivations that influence our choice of a methodology that plans in advance for evolution are: 1) the problem of schema evolution is better understood now because researchers have already provided an overview of its causes and consequences, therefore it is now time to turn towards complex and multi-disciplinary approaches 2) the posteriori approaches have not been sufficient solutions for schema evolution even if they are considered to be standard solutions and finally 3) the a priori approach is absolutely the best alternative: the key to evolution problem lies in thinking of the evolution from the beginning of the lifecycle of the database.

b) Predictive analytic solution

A predictive solution generally refers to data mining techniques such as classification to predict the value of a particular attribute based on the value of other attributes. The attribute to be predicted is called the dependent variable while the attributes used for making prediction are called the independent variables [5]. Our predictive solution 1) uses data-mining techniques such as Classification Based on Association Rules, and 2) in particular, it explicitly includes a requirement analysis phase. In the requirements analysis phase, besides assessing the current user requirements via the database user's feedback and comments, additional requirements called *potential future requirements* are investigated using the current requirements of several databases. These new requirements, representing potential future needs that might emerge during the lifecycle of the database in the future are inspected inside a schema repository. In this case, the current requirements are representing the independent variables while the future requirements are the dependent variables.

c) Proactive schema solution
Our solution takes actions to handle changes before the evolution of the database schema. In other words, this approach generates a scalable database schema called the *predicted schema* at design time that contains three different layers which are respectively: 1) Operational Layer, 2) Existing but not Operational Layer, 3) Not Existing but Planned Layer. In these three layers, the schema potential structural changes are hold at the present time for its use in the evolution time.

d) Three different stages solution
The approach holds one additional stage which is novel in database modeling design; in contrast existing approaches enclose just two stages: (1) design time and (2) evolution time. The additional third stage of the predictive approach, the *before design time stage*, is an important stage. It paves the way for the design time stage by conducting the preparation of the accurate data for the schema evolution from the schema repository.

e) Data modeling methodology independent solution
Because of the diversity of the data models used to represent a database schema, we have chosen to develop an approach that can be adopted by any modeling methodology.

f) Collecting data solution
Our solution uses the direct collection of information from different external sources and a consultation of schemas of existing databases.

There are multiple advantages of this approach. Indeed, it contributes significantly in the ability of the database schema to 1) accommodate the future changes, and 2) facilitate the work of designers and help them save time and money on the evolution of their databases. Consequently, all these qualities have a positive impact on both database users and the organisms that employ such pre-emptive approaches.

3 Requirements Ontology for Schema Evolution

The requirements ontology is a domain ontology in which requirements are expressed with concepts (terms), relationships and constraints. It allows the system to relate the current schema to possible future needs. For example, if a database designer needs to create a database for meetings, the requirements ontology associated to this database contains concepts, relationships and constraints related to meeting domain such as *MEETING*, *PARTICIPANT*, *ROOM* and *AGENDA* concepts and *IS*, *HAS* relationship types and so on. This is illustrated in figure 1. The requirements ontology gains its insights into the possible future needs of the schema through various methods; its construction is described further in Section 3.2.

The requirements ontology looks like a global entity relationship model; however, it is richer than an entity relationship model because 1) it contains more semantics related to a specific domain. 2) the instances of the requirements ontology are divided into two categories: in addition to the instances representing the current requirements, the database designer needs to choose the concepts that might correspond to potential future requirements. Consequently, several design suggestions about the entities with their relationships are taken from the requirements ontology and are provided to the database designers.

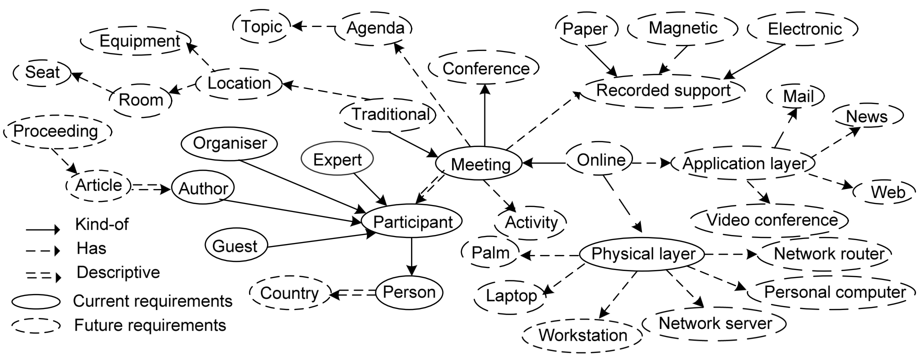

Fig. 1. Presentation of a part of requirements ontology of meeting domain

Since the system is designed to be flexible enough for many situations, and it is impossible to predict with 100% accuracy which requirements will be desired, the system includes three strategies to the database designers for the selection of those requirements ontology concepts and their corresponding relationships. These strategies are:

1) Blind selection: all the concepts of the requirements ontology belonging to future requirements are selected without exception.

2) Case based selection: This selection mechanism allows the inclusion of concepts that are particularly likely to be needed in the future. In particular, the concepts that satisfy one of the following cases are selected. This is a way to find out the important concepts that have already pre-established links among them

Case 2.a: a concept belonging to future requirements which is situated between two concepts of current requirements. This is illustrated in figure 2.

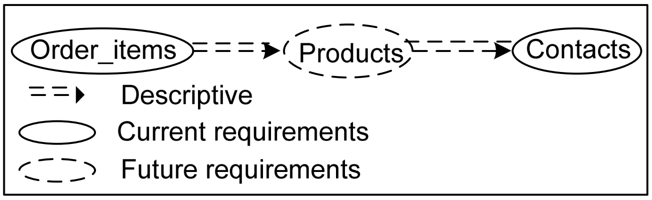

Fig. 2. Case of future concept between current concepts

Case 2.b: a concept that is a final node (part i) or belongs to the end of a branch in the requirements ontology (part ii) is selected. This is illustrated in figure 3 and 4.

- (i) A final node

A final node is a concept in the requirements ontology which has only one relationship with another concept of the requirements ontology. For example in figure 3, the concept *Ref_Order_Status*, belonging to the category future requirements, has only one relationship in the whole requirements ontology which is with the concept *Order* that is its direct ancestor. An ancestor is a node from which starts the selection of the nodes of the requirements ontology.

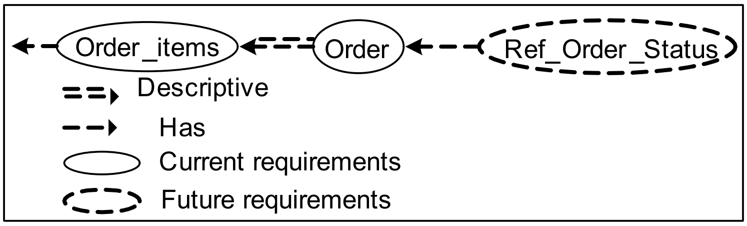

Fig. 3. Case of a concept as a final node

- (ii) An end of a branch

A branch is a sequence of concepts linked together with one or several links however one concept of this succession has only one relationship with another concept of the sequence in the whole requirements ontology. This is illustrated in figure 4 in which the concept *Product_description* is a concept of the sequence that has only one relationship with the concept *Product*.

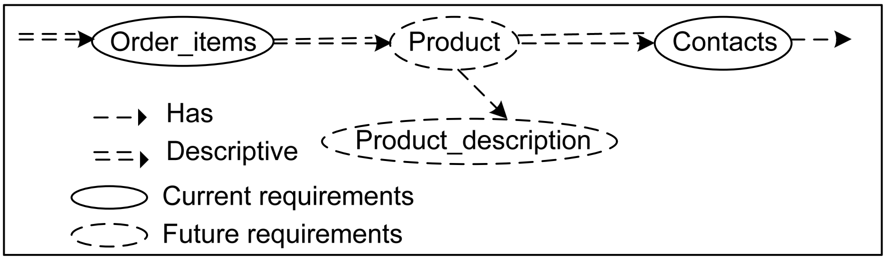

Fig. 4. Case of concepts as an end of a branch

Case 2.c: the concepts that form a clique of a graph in the requirements ontology are selected. This is illustrated in figure 5.

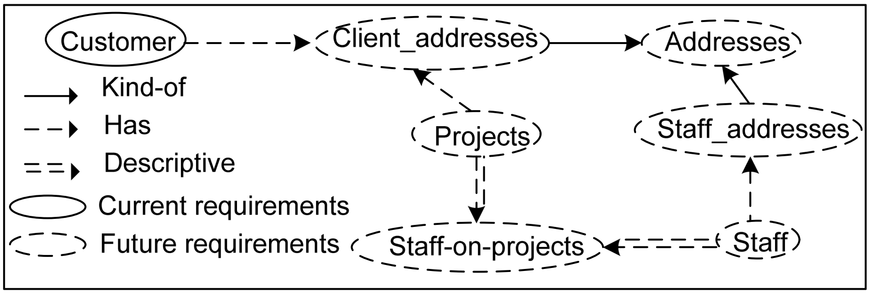

Fig. 5. Case of a clique graph

3) Change perspective selection: the concepts are selected according to their *change perspectives*. A change perspective is a special graph that shows the potential changes of each concept belonging to the ontology. It is built using several data mining techniques such as classical association rules and classification based on association rules[5]. The requirements ontology contains three different types of change perspectives with specific roles which are respectively: informative, descriptive and predictive.

To illustrate this point, we consider the following example that represents the modelling of a simple case of scientific meeting in which the requirement is to determine the people participating to a conference. At the ontological level, for the concept "Meeting", the change perspectives, illustrated in figure 6, show the potential changes related to it and the other concepts that might be implied at the evolution time.

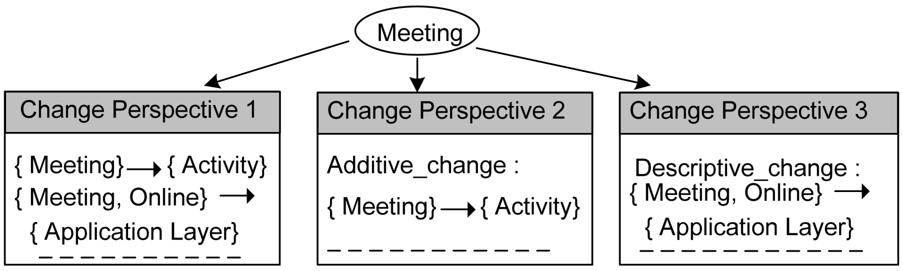

Fig. 6. Change perspectives for the concept meeting

3.1 Requirements Ontology Role

There are two ways in which the requirements ontology is used in the predictive approach for evolution. These two primary functions are 1) design and evolution and 2) redesign and evolution.

1) Design and evolution
In case the initial database schema is not created yet, the ontology fulfils several tasks, as presented in [6] and [7]: it generates a design "from" scratch using the defined terms and relationships as a representative model of the domain. It suggests possible missing entities and relationships in the case just a part or selected parts of it is/are considered by the database designer. The requirements ontology offers additional features; it includes terms, relationships and constraints that might represent potential future requirements and identifies in advance their dependencies with terms and relationships representing current requirements. Consequently, it facilitates the work of database designer when changes should be made on the schema.

2) Redesign and evolution
In case the initial schema has already been created and now needs to be redesigned, then the ontology fulfils other important tasks as presented in [6] and [7]. For

example, it is used to check for missing entities or relationships or inconsistencies in an existing or partial design because the data model produced in the redesign process, called a reverse engineered (RE) data model [8], cannot be considered as a conceptual schema. The RE model converts all the logical schema tables to entities without making distinction between data tables and the other tables of the schema whose function is to join tables. This model is not a logical schema because some important schema information is lost during the conversion process, such as in the case of foreign keys.

In the following, we portray the three categories of changes presented previously, which are 1) the additive evolution, 2) the subtractive evolution and 3) the descriptive evolution and how the requirements ontology proceeds in each case.

1) Additive Evolution

There are two types of additive changes: simple and complex. For the simple additive change, the database administrator can use the functionalities of the DBMS (Database Management Systems) to add for example a table or an attribute. In contrast, for the complex additive change adding an element perturbs the dependency between existing elements and causes damaging effects on existing applications. Consequently, the logical schema is in inconsistent state and the associated applications do not work anymore.

In Section 5.1, we illustrate through a detailed example why complex additive evolution must be handled specially and show how the predictive approach offers a better solution than existing posteriori solutions.

2) Subtractive Evolution

Subtractive evolution occurs when elements in the schema are no longer required. However, deleting an element on existing schema is not always obvious, leading to two types of subtractive changes: simple and complex. For the simple subtractive change, the database administrator can use the functionalities of the DBMS (Database Management Systems) to delete the no longer required elements. Whereas for the complex subtractive evolution, the DBMS does not offer any functions for it and the changes have direct and critical consequences on the schema and applications.

In Section 5.2, we illustrate through a detailed example why complex subtractive evolution must be handled specially and show how the predictive approach offers a better solution than existing posteriori solutions.

3) Descriptive Evolution

Descriptive evolution is made for convenience or efficiency. It is the hardest to handle in traditional database systems because it implies more than one risky modification operation on the schema. The consequences of the changes on the schema are also critical, such as data loss.

In Section 5.3, we illustrate through an example why complex descriptive evolution must be handled specially and show how the predictive approach offers a better solution than existing posteriori solutions. The lack of space in this paper does not allow us to explain in detail the example.

3.2 Requirements Ontology Construction

The requirements ontology is developed using both a schema repository in which the main concepts are extracted and WordNet ontology [9] for extracting their corresponding synonyms and antonyms. The requirements ontology consists of two kinds of partitions, the ones representing current requirements called Current sub-domains and the ones corresponding to potential future requirements called Future sub-domains.

The process of the requirements ontology creation is iterative and complex some how compared to existing approaches. It consists on four main phases: *knowledge acquisition*, *Data mining and informal conceptualisation*, *Evaluation for Refinement or Revision* and *Formal Conceptualization*

1 -Knowledge acquisition and pre-processing: consists of schemas collection and preparation

2 -Data mining algorithms and informal conceptualization: in which concepts and relationships are extracted from schema data sets repository in an unsupervised way and used as output for the informal conceptualization of the ontology from scratch.

3 -Evaluation for Refinement or Revision: means to test the validity of the concepts belonging to the taxonomy and to decide to keep or reject them using qualitative and quantitative methods.

4 -Formal Conceptualization: consists in building formally the requirements ontology using OWL and description logic.

These phases are not very developed in this paper because they necessitate a considerable space.

The schema repository contains many different schemas that model a specific domain. These schemas and their related versions may be of different types, such as ER, relational, object and object-relational schemas. The XML databases can be included as well as the ontological schemas expressed in OWL technology [10]. The schema repository has a dual role in building the requirements ontology [11]: (1) the repository serves in the data-mining process to identify and analyze trends on different kinds of schemas collected. (2) The repository contains selected concepts and relationships to be included in the requirements ontology.

3.3 Requirements Ontology Structure

The structure of this ontology includes: a) Concepts, b) Relationships, c) constraints d) Current versus Future Labels, and described in more detail below:

a) Concepts (Terms) Description:
Each term has one or several attributes with one or several values and one or several synonyms and antonyms.

b) Relationships Description
Relations are between two concepts. There are six kinds of relations: (1) hierarchic – identified by the label "kind-of", which expresses the specialization of one concept

regarding another and inherits attributes from this super concept; (2) composition – identified by the label “has”, which expresses that a concept is a part of another concept; (3) descriptive – when it is possible to define several types of relations and is identified by a verb form; (4) reflexive – allows self-loops in which an arc whose endpoints are the same concept.

We consider the previous example of meeting domain to show some relationships among concepts: kind-of (meeting, conference) is a hierarchic relationship, has (meeting, utterances) is a composition relationship, lives (Person, Country), originates (Person, Country) and represents (Person, Country) are descriptive relationships and finally invites (Person, Person) is a reflexive relationship.

c) Constraints
Similar to the work presented in [6] and [7], we use four types of constraints which are respectively: 1) pre-requisite constraint, 2) mutually inclusive constraint, 3) mutual exclusive constraint and 4) temporal constraint.

d) Current versus future labels
The requirements ontology is a labeled graph: special labels are added and exploited in order to indicate whether a concept, respectively a relationship belongs to current or future requirements. A concept respectively, a relationship belongs to either current requirements or future requirements but not to both at the same time. This is main structure characteristic that distinguishes the requirements ontology from the remaining domain ontologies.

e) Change perspectives

4 Multi-representation Strategy for the Predictive Approach

In the predictive approach, the predicted schema is semi-automatically generated from the requirements ontology. At the conceptual level, the predicted schema is represented either 1) with the multi-representation strategy or 2) without the representation strategy. In this paper, we stress the use of the multi-representation strategy as follows:

4.1 Definition of the Multi-representation Strategy

The multi-representation strategy is well-known in the object-modeling field, as well as in the spatial databases. In [12], the multi-representation strategy based on stamping in geographic databases is presented. On the other hand, in the object modeling, the multi-representation is called semantic object views. It allows to make the object visible for certain applications and to hide it to others using the views mechanism. In this work, we focus on the multi-presentation based on stamping.

4.2 The Predicted Schema at the Conceptual Level

This strategy consists in using stamps at the conceptual level in order to have different representations for the modeling of the same universe of discourse i.e. the modeling of the same real-world. A stamp S is defined as a vector S=<s1, s2, Sn> where each s¡ represents the ¡ representation of the real-world. For example, in the following simple example, we have defined a stamp S =<S1, S2> in which, according to the element S1 of the stamp S, the conceptual schema contains the entities E1, E1' and the relationships A1'. However, according to the element S2 of the stamp S, the conceptual schema contains the entities E1 and E2 and one relationship A1. This is illustrated in figure 7.

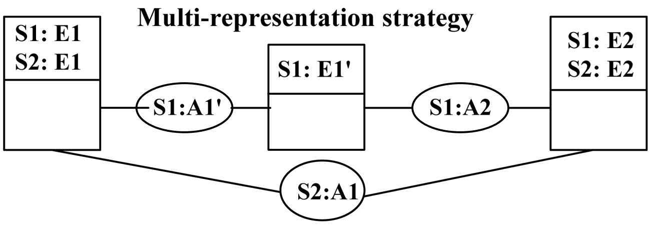

Fig. 7. A Simple Example using Multi-Representation Based on Stamping

The stamping mechanism is not a simple mechanism as it may appear. For example, in the case of successive evolutions on the database schema, the stamp components and the constraints on the stamps should be studied carefully in order to avoid any potential contradiction among them.

5 Motivating Examples

In this section, we present examples of schema changes to illustrate how the predictive approach for evolution discussed in this paper works. The examples portray the three categories of changes presented previously, which are 1) the additive evolution, 2) the subtractive evolution and 3) the descriptive evolution.

5.1 Additive Evolution

A case of complex additive change is illustrated in the side 1 of the figure 8 in which the addition of the entity E'1 creates problems for existing applications. At the time **T= t0**, we have a schema with two entities **E1** and **E2** and an association between them **A1** as presented in figure 8**.** At the time **T=t1**, the evolution time, a schema has been modified and complex additive changes occur: an Entity **E'1** and two associations **A'1** and **A2** are added. The association **A1** between the entities **E1** and **E2** is consequently deleted to avoid a redundancy which is itself a problem and information on schema is lost. The way to resolve such a problem with the predictive consists in:

1 - At the ontological level: the database designer examines whether the requirements ontology reveals the existence of concepts/relationships that belong to the category of future requirements and represent potential simple and complex additive changes.

2- At the conceptual level, the database designer incorporates these concepts/ relationships using the multi- representation strategy based on stamping mechanism. The resulted conceptual schema represents consequently two universes of discourse (real-world). This is illustrated in the side 2 of the figure 8.

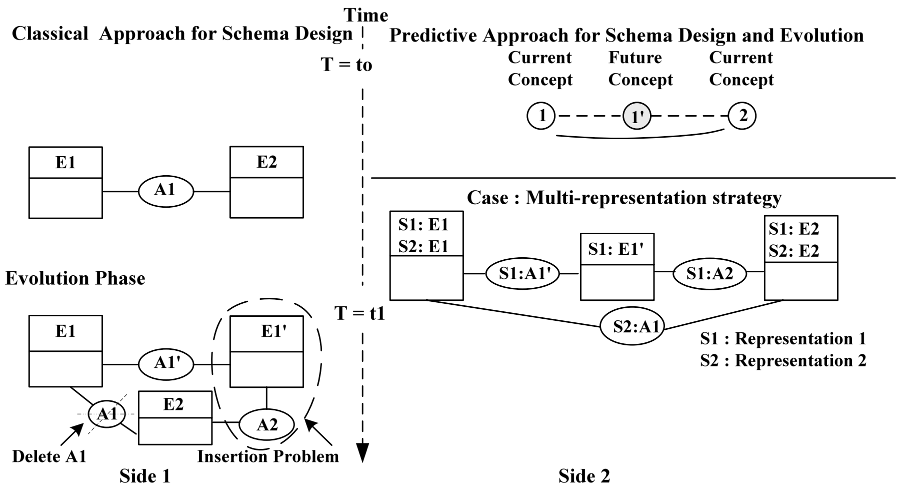

Fig. 8. Additive Evolution on an Example with both Classical and Predictive approaches

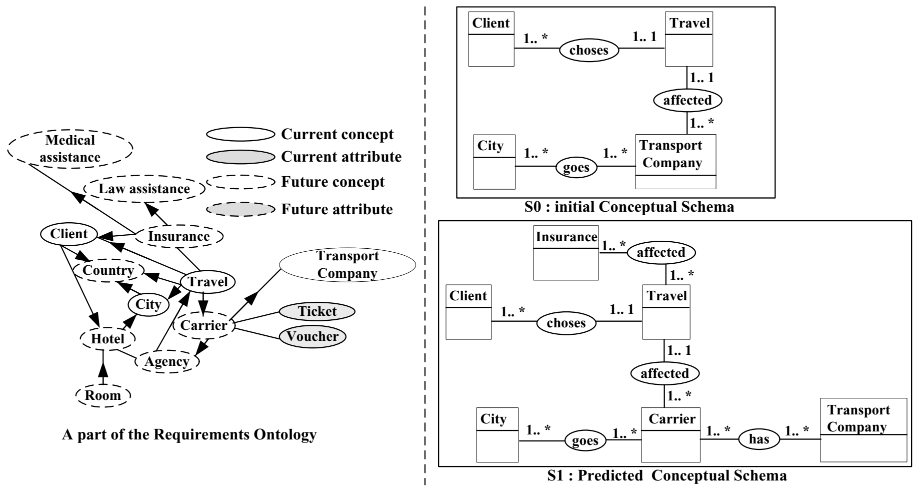

Fig. 9. An example of travel with both classical and predictive approaches

A case of complex and simple additive changes on the schema is illustrated in figure 9 where we consider the example that represents the modeling of a simple case of travel, in which the requirement is to determine the clients traveling around the world. In a classical design approach, the initial conceptual schema S0 contains four entities which are *Client*, *Travel*, *City* and *Transport Company*. However, in the predictive approach, the schema S1 that has been proposed by the requirements ontology contains six entities. The two additional entities *Insurance* and *Carrier* represent two potential future changes on the schema that belong to simple and complex additive changes respectively.

5.2 Subtractive Evolution

A case of complex subtractive change is illustrated in the side 1 of the figure 10 in which the deletion of the entity E2 creates problems for the existing applications that need such entity. The whole process to resolve such problem is illustrated in the side 2 of the figure 10.

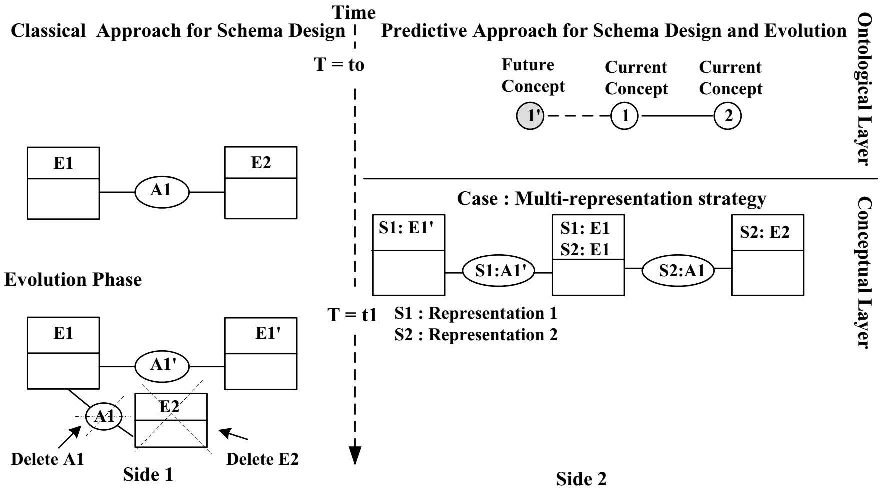

Fig. 10. Presentation of subtractive evolution on a simple example with both Classical and Predictive approaches

5.3 Descriptive Evolution

This is illustrated in the side 1 of the figure 11. Similarly to the previous, we follow the same steps for the resolution of this problem according to the predictive approach. The whole process is illustrated in the side 2 of the figure 11.

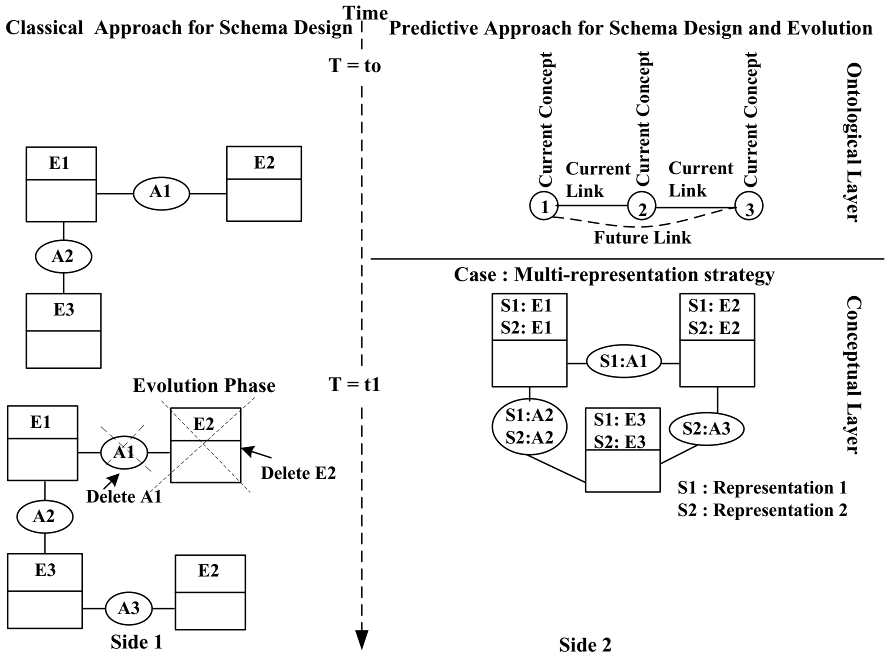

Fig. 11. Presentation of Descriptive evolution on a simple example with both Classical and Predictive approaches

6 Conclusion and Future Work

In this paper, we have presented another area where ontologies have a huge potential benefit to information systems: database schema evolution. The approach we propose belongs to a new tendency called the tendency of a priori approaches. It implies the investigation of potential future requirements besides the current requirements during the standard requirements analysis phase of schema design or redesign and their inclusion into the conceptual schema. Those requirements are determined with the help of a domain ontology called "a requirements ontology" using data mining techniques and schema repository. The advantages of this approach include: 1) new perspectives in the way requirements are inspected and integrated into the schema, 2) two categories of database designers were taken into consideration, the category of those who design a schema from scratch and the category of those who redesign the schema from existing schemas using reverse engineering and dependency graphs, 3) the reinforcement of the conceptual schemas, 4) and finally the compatibility of the approach with any data model.

Prediction in schema evolution means to envision the potential changes that could occur over time on the schema. However, prediction does not work all the time; i.e. it is not always possible to detect the changes that are plausible to occur on a database schema and are important for the database users because of the complex perception of the real world. Therefore, the predictive approach operates according to three main scenarios which are respectively:

***1- Scenario 1*: Avoid redesign because better designed schema ready for evolution**
The first scenario involves the case in which the predicted schema of the database does not need to be evolved or in other words, the schema has already evolved because the required changes are already built-in and the database designer or the database administrator does not have to adjust the schema for them at the evolution time.

***2- Scenario 2:* Redesign slightly because already planned so easier**
The second scenario involves the case in which the predicted schema needs to evolve. However this evolution is straightforward to realize because the required changes have already been planned and therefore need just to be added in the database schema. Consequently, in the evolution time, the schema is redesigned slightly.

***3- Scenario 3:* Redesign from scratch**
The third scenario concerns the case in which the anticipated changes are not accurate for the evolution of the database scheme because somehow the potential detected future changes are not appropriate and sufficient. Consequently, in the evolution time, the database schema needs absolutely to be redesigned from scratch in order to include all the adequate changes that have occurred on it over time. This scenario raises the problem that prediction is not feasible each time and it therefore implies 1) Maybe more work since the requirements ontology needs to be updated because it does not contain the needed information. 2) However, Updating the requirements ontology may help for other future schemas redesign and evolution. For example, a case in which the government legislation means radical changes in the way tax is paid on investment interest involves changes to the investment file.

Another problem of this approach is that the effectiveness of this approach for evolution is limited by the amount and the quality of the knowledge accumulated inside the requirements ontology. Therefore, we have taken into consideration the problem of the evolution of the requirements ontology as well. For this purpose, we have adopted the multi-representation strategy based on stamping mechanism. In [13] a multi-representation solution for ontologies is presented. This solution develops a language based on description logic (DL) [14] to implement the stamping mechanism. Unfortunately, this new approach is not without problems.

Future work will proceed in both theoretical and practical directions. The theory will focus on extending the idea behind the requirements ontology and the stamping mechanism. The practical work consists in testing this approach significantly through several case studies with the use of a prototype that is under development.

References

1. Connor, R.C.H., Cutts, Q.I.: Using Persistence Technology to Control Schema Evolution. In: Proceedings of the ACM symposium on Applied computing Phoenix, Arizona. ACM Press, New York (1994)
2. Roddick, J.F: A survey of schema versioning issues for database systems. Information and Software Technology 37(7), 383–393 (1995)

3. Borgida, A., Williamson, K.E.: Accommodating Exceptions in Databases, and Refining the Schema by Learning from them. In: 11th International Conference on Very Large Data Bases, Stockholm, Sweden. Morgan Kaufmann, San Francisco (1985)
4. Benatallah, B.A: Unified Framework for Supporting Dynamic Schema Evolution in Object Database. In: Akoka, J., Bouzeghoub, M., Comyn-Wattiau, I., Métais, E. (eds.) ER 1999. LNCS, vol. 1728, Springer, Heidelberg (1999)
5. Tan, P.-N., et al.: Introduction to Data mining Pearson. Addison Wesley, Pearson (2006)
6. Sugumaran, V., Storey, V.C.: Ontologies for conceptual modeling: their creation, use, and management. Data Knowledge. Engineering 42(3), 251–271 (2002)
7. Sugumaran, V., Storey, V.C.: An Ontology-Based Framework for Generating and Improving Database Design. In: Andersson, B., Bergholtz, M., Johannesson, P. (eds.) NLDB 2002. LNCS, vol. 2553, pp. 1–12. Springer, Heidelberg (2002)
8. Kroenke, D.M.: Database Processing: Fundamentals, Design and Implementation. In: Acevedo, G.S.d. (ed.), pp. 265–275. Prentice Hall, Pearson (2004)
9. WordNet: http://wordnet.princeton.edu/
10. Lacy, L. W.: OWL: Representing Information Using the Web Ontology Language (2005)
11. Bounif, H., Pottinger, R.: Schema Repository for Database Schema Evolution. In: 2nd international workshop on Data Management in Global Data Repositories (GREP) 2006 at International Conference on Database and Expert Systems Applications 06 (2006)
12. Spaccapietra, S., et al.: Supporting Multiple Representations in Spatio-Temporal databases. In: Proceedings of the 6th EC-GI & GIS Workshop, Lyon, France, June 28-30 (2000)
13. Benslimane, D., et al.: Multi-representation in ontologies. In: Kalinichenko, L.A., Manthey, R., Thalheim, B., Wloka, U. (eds.) ADBIS 2003. LNCS, vol. 2798, pp. 4–15. Springer, Heidelberg (2003)
14. Baader, F., Nutt, W.: Basic Description Logics. In: Baader, F. (ed.) The Description Logic Handbook: theory, implementation and application, pp. 43–95. Cambridge University Press, Cambridge (2003)