Supporting Database Evolution: Using Ontologies Matching

Nadira Lammari[1], Jacky Akoka[2], and Isabelle Comyn-Wattiau[3]

[1] Laboratoire CEDRIC-CNAM,
292 Rue Saint Martin, F-75141 Paris,
[2] Laboratoire CEDRIC-CNAM et INT,
[3] Laboratoire CEDRIC-CNAM et ESSEC,
{lammari,akoka,wattiau}@cnam.fr

Abstract. In this paper, we propose a methodology for managing database evolutions by exploiting two ontologies. The first ontology describes the changes occurring in a database application. The second one aims at characterizing techniques and tools useful for database change management. We propose an algorithm performing ontologies matching and its application to identify appropriate techniques and tools for a given database change.

1 Introduction

Whilst considerable research is currently focused on software evolution, less attention has been devoted to the practical problem of identifying appropriate techniques and tools for managing database evolutions. The high cost incurred in software investments is mainly attributed to the maintenance efforts required to accommodate changes [1]. Database systems play a central role in the software evolution. Many reasons can explain the need for database evolution: volatile functional requirements, changing users' requirements, modification of organizational processes and business policies, evolving technological and software environments, architecture constraints, etc. Moreover, changes can affect many aspects of database applications. Failing to fully taking into account these changes will negatively impact the application. An efficient way to accommodate these changes is to provide a systematic process allowing the maintenance engineer to update the system. This paper reports the development of an algorithm for ontologies matching which addresses one of the central problems of database evolution: identifying techniques and tools which are relevant to database changes. This algorithm utilizes three structures: an ontology of changes, an ontology of techniques and tools and a database recording change evolutionary histories.

The remainder of the paper is organized as follows. In the next section, we describe the change ontology. Section 3 is devoted to the presentation of the tools and techniques ontology. Section 4 presents the main steps of the approach. Finally, Section 5 concludes the paper and gives some perspectives.

D. Konstantas et al. (Eds.): OOIS 2003, LNCS 2817, pp. 284–288, 2003.

2 An Ontology of Changes

Past work on change typology has been initiated by Lientz and Swanson [2]. They propose a maintenance taxonomy distinguishing between perfective, corrective and adaptive maintenance activities. This classification has been extended into a taxonomy of twelve different types of software evolution: evaluative, consultive, training, updative, reformative, adaptative, performance, preventive, groomative, enhancive, corrective and reductive [3]. A complementary view has been taken by [4] focusing more on the technical aspects: the when, where, what and how of software changes. Felici reviews a taxonomy of evolution identifying a conceptual framework to analyze software evolution phenomena [5]. Finally, in an earlier paper, we proposed a framework for database evolution management based on a typology of changes encompassing three dimensions: the nature of change, the significance of change and the time change frame [6]. In this paper, we extend this typology of change to an ontology unifying past approaches. It can be used as a surrogate for the meaning of concepts in database application changes. The purpose of the ontology is to provide information regarding the semantics of new database change concepts, and to compare two concepts in order to determine if they might be related. This ontology classifies an evolution (or a change) into one or more categories depending on the significance of change, its semantic preservation, the information and application dependency, etc. The ontology consists of a semantic network describing the different categories into which change terms can be classified (Fig 1).

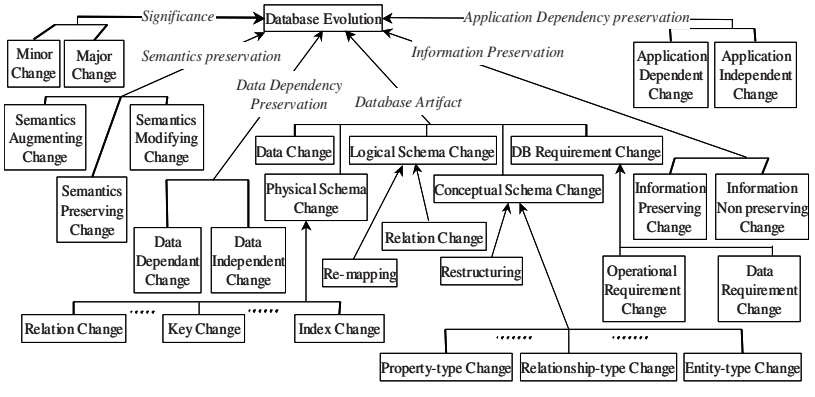

Fig. 1. Change ontology

The semantic network has about twenty-five leaf nodes corresponding to each of the possible categories into which a change can be classified. Sub-nodes are connected via IS-A links as indicated by the arrows (Fig. 1). Additional links such as "may generate", "is synonym of" and "is instance of" allow us to describe respectively: potential change that can be generated by a change, concept naming, and instance list of a given change. Figure 1 describes a small part of the semantic network representing the change ontology.

3 An Ontology of Techniques and Tools

The literature supplies a set of approaches and techniques for schema evolution. Approaches for database transformations have been proposed in [7]. At the design level, specific methodologies for Entity-Relationship (ER) schema transformations have been elaborated [8,9,10]. Some approaches independent of database content can be found in [11]. Dependency constraints are used in some transformations approaches [12]. Finally, Mens lists and compares several tools: Refactoring Browser, Concurrent Versions System, and eLiza [4].

Our aim is to extend and structure these techniques and tools into an ontology enabling maintenance engineers to select the most appropriate tools and/or techniques for a given change. This ontology is partially described below (Fig 2).

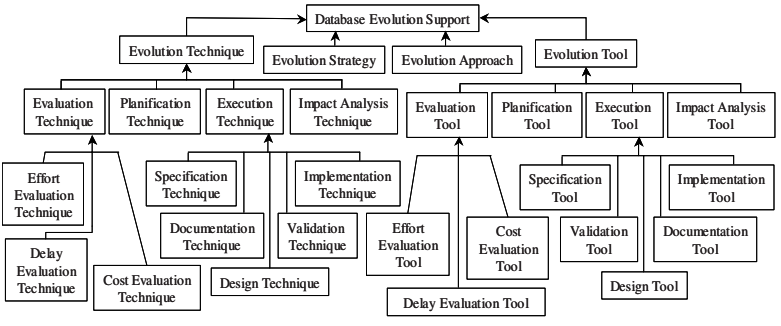

Fig. 2. Techniques and tools ontology

4 Main Steps of the Guidance Approach

Given these two ontologies, a matching process can be defined. For example, a major change implies a backward approach if the change occurs at the requirement phase. Such an example illustrates the fact that the relationship between the two ontologies cannot be represented by simple links between both ontologies' leaf nodes. Therefore, we need to refine further the changes. They can be characterized by a vector whose components materialize the location of the change in the first ontology. The vector components are: significance, time frame, semantic preservation, data dependency preservation, information preservation, application dependency preservation, database artifact. As a consequence, we associate to each leaf node of the database evolution support ontology the corresponding vectors values by a special link "may-be-realized-with".

The general architecture of the approach is composed of four phases. The first phase consists in characterizing the change being considered, such as a business rule modification leading to a change in the logical schema and even in the code. The characterization allows us to locate the change within the first ontology. This leads us to associate values to the vector components. Therefore the change vector is fully computed. In the second phase, we perform a search allowing us to locate this vector in the second ontology. This vector is associated to a set S of approaches, techniques,

strategies and tools by means of "may-be-realized-with" links. The third phase will compare the proposed techniques and tools contained in S to the ones that have already been used, by checking the change history database. The latter describes numeric values characterizing these changes, and data about cost, duration and quality needed to implement each change. By confronting the considered change with the change history, we generate a sorted list of candidate techniques and tools. The maintenance engineer can perform the choice inside this list using the appropriate criteria, such as cost, duration and/or quality.The aim of the last phase is to update the change history database. In particular, cost, duration and quality needed to perform the changes are recorded.

As an example, let us consider a payroll system. During the maintenance phase, a major business rule is modified. In the current system, an employee is assigned to a unique job. The new rule stipulates that any employee can be assigned to several different jobs. Its salary is composed of the aggregation of partial earnings. The aggregation rules are specific to each employee. Applying our algorithm, we obtain the following result:

1. Change characterization: It can be considered as a minor change (significance=1), occurring at the maintenance phase of the application (time frame=3). This modifies the semantic of the database (semantic preservation=2). It preserves information (information preservation=2) and data dependency (data dependency preservation=2). It is an application-dependent change (application dependency preservation=4). The target database artifact change is a relationship-type change (database artifact=1). Thus the resulting vector is (1, 3, 2, 2, 1, 4, 1).
2. Mapping algorithm: This vector leads to a backward approach, a forward and/or a reverse engineering technique and implementation tools.
3. History confrontation: In the change history database, reverse engineering leads to costly implementation for the change. However, it is considered as a high quality change implementation. The final choice should be made by the maintenance engineer considering a trade-off between quality and cost adding the duration constraint.
4. Change historization: We record this new change in the change history database, including the cost, duration and quality information observed by the maintenance engineer.

5 Conclusion and Further Research

This paper describes an ontology-based approach allowing us to guide the maintenance engineer in the process of choosing the relevant techniques and tools for database application change. To perform such a choice, we provide an ontology of change, an ontology of techniques and tools and an algorithm for the mapping between them. The approach is enriched by an evolutionary histories process updating the changes performed in the past.

Further research will consist in designing and developing a prototype guiding the choice of techniques and tools for a given change. By prototyping this approach, we will be able to validate both our ontologies and their use in the context of database evolution. In parallel, we will investigate techniques and tools to enrich the ontology and the mapping.

References

1. Cremer K., Marburger A., Westfechtel B.: Graph-Based Tools for Re-engineering. Journal of software maintenance and evolution: research and practice, Vol 14, 2002, 257–292.
2. Lientz, Swanson**,** Distributed Systems Development and Maintenance. In Management of Distributed Data Processing, Akoka, J. (ed.), North-Holland, Publishing Company, Amsterdam, 1982.
3. Chapin N, Hale J, Khan K, Ramil J, Than W.G.: Types of Software Evolution and Software Maintenance, Journal of Software Maintenance and Evolution, 2001.
4. Mens T., Buckley J, Zenger M, Rashid A.: Towards a Taxonomy of Software Evolution, USE03, 2nd International Workshop on Unanticipated Software Evolution, in conjunction with ETAPS03, Warsaw, 2003.
5. Felici M: Taxonomy of Evolution and Dependability, USE03, 2nd International Workshop on Unanticipated Software Evolution, in conjunction with ETAPS03, Warsaw, 2003.
6. Comyn-Wattiau I, Akoka J, Lammari N, A Framework for Database Evolution Management, USE03, 2nd International Workshop on Unanticipated Software Evolution, in conjunction with ETAPS03, Warsaw, 2003.
7. Davidson S., Buneman P., Kosky A.: Semantics of Database Transformations. LNCS 1358, Edited by L. Libkin and B. Thalheim, 1998.
8. Batini C., Lenzerini M., Navathe S.: A comparative Analysis of methodologies for Database Schema Integration. ACM Computing Survey Journal 18 (4), 1986.
9. Kim W., Choi I. Gala S., Scheeval M.: On Resolving Schematic Heterogeneity in Multidatabase Systems. Modern Database Systems, ACM Press, 1995.
10. Kashyap V., Sheth A.: Semantic and Schematic Similarities Between DataBase Objects: A context-based Approach. VLDB journal, 5 (4), 1996.
11. Miller R. J., Ioannidis Y. E., Ramakrishan R.: Schema Equivalence in heterogeneous Systems: Bridging Theory and Practice. Information System Journal, 19 (1), 1994.
12. Biskup J., Convent B.: A formal View Integration Method. Proceedings of ACM SIGMOD International Conference on Management of Data, Washington, 1986.