On the implication of application's requirements changes in the persistence layer: an automatic approach

Jennifer Pérez, José Ángel Carsí, Isidro Ramos

Department of Information Systems and Computation
Valencia University of Technology
Camino de Vera s/n
E-46071 Valencia – Spain
{jeperez | pcarsi | iramos}@dsic.upv.es

Abstract. This paper presents a solution to the data evolution problem of information systems. This solution follows an automatic approach that reduces the number of people and the time invested in the software maintenance process. The information captured by the UML-like OASIS OO conceptual schemas, representing system evolution, is used in order to generate automatically the first version of a data migration plan. This paper is focused in the process of automatic generation of the structure and the content of this migration plan using patterns. A real migration example is used to explain the designed patterns. The data migration plan execution evolves and migrates the data from the initial conceptual schema database to the evolved conceptual schema database. The solution has been implemented in the migration tool ADAM (Automatic DAta Migration)[1].

Key words: Data migration plan, migration order, correspondences, migration patterns and migration expressions patterns

1. Introduction

Information systems have a dynamic nature and they frequently undergo changes. These changes are due to volatile business rules or to external factors. For this reason, system management tools must be able to deal with them in an efficient and complete way.

Existing CASE tools are able to generate applications following the paradigm of automatic prototyping. These CASE tools are called model compilers. They generate automatically the application code and the database schema from the conceptual schema of an information system. The automatic generation can be complete as Oblog Case ([Ser94]), Sosy Modeler® (OO-Method/CASE ([Pas97]), or partial, as Rational Rose ([Rat02]), System Architect ([Sys02]), Together ([Tog02]) and others.

[1] ADAM is a CONSOFT technology product.

Moreover, they can evolve the application modifying the conceptual schema and regenerating the code and the database schema from the modified conceptual schema (see figure 1).

These model compilers do not consider the data stored in the database during the evolution process. When an information system undergoes an evolution, its conceptual schema is updated and a new schema results. From the new schema, a model compiler generates a new code and a new empty database. The structure and the properties of the new database could be different from the old database. Therefore, the data remains compliant to the old database schema and the user of the application must preserve the data of his company in spite of the evolution of the information system conceptual model. For this reason, the data remaining in the old database must be correctly migrated to the new one in order to satisfy the properties of the new database.

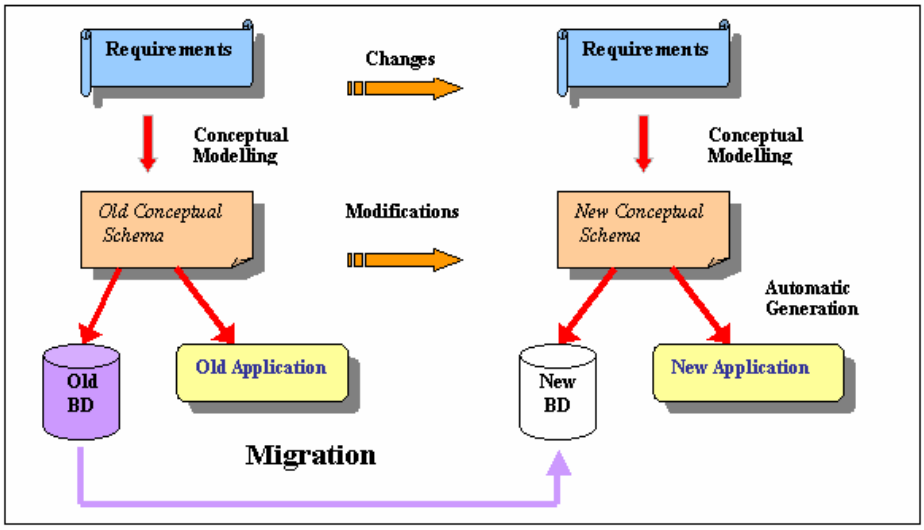


Fig. 1. Evolution of applications implemented by means of a model compiler

The migration task is necessary and normally is handmade. This task increases considerably the maintenance cost of a software product. For this reason, our work is focused on data evolution in order to improve this database maintenance process. In this paper, we present a solution for the data evolution problem following an automatic approach and a migration tool (ADAM) has been implemented. Nowadays, ADAM is used together with Sosy Modeler®, a model compiler generating automatically three layer architecture (presentation, business logic and persistence) applications. A formal language OASIS ([Pas95]) based in dynamic logic is used as the specification language of UML-like models allowing its automatic compilation.

ADAM is a migration tool developed in the Department of Information Systems and Computation of the Valencia University of Technology in collaboration with the industrial partner CONSOFT. The data migration process followed by ADAM transfers and updates information system data from the old database to the new one. It reduces the time invested and the number of people involved in data evolution process

up to an 80%. This maintenance improvement is due to the automatic tasks that constitute the three steps of the migration process. Despite the fact that these steps are performed automatically, its results can be modified by the analyst in a free way. In this last case, the process will be semi-automatic. The three steps involved are the following:

1. Old and new conceptual schemas are compared using different comparison criteria ([Sil02]) to automatically obtain the matchings (correspondences) between them.
2. Changes on the data are obtained from these matchings and are automatically translated to migration expressions using an object oriented migration language. These expressions constitute a data migration plan ([Per01-b]). The migration plan is generated as XML document, used by the third step to extract easily the migration information.
3. The object oriented data migration plan is translated to a relational one. This translation process generates *Data Transformation Service* packets ([Cha00]) whose execution migrates automatically data from the old database to the new database ([Ana01]). These packets are executed by *SQL Server* ([SQL]).

The 80% reduction of the time invested and the number of people have been obtained as a result of the set of migration experiences made in a company. These experiences consisted in migrating with our tool the same databases that have been migrated manually by a set of analizers who knew the information system.

The structure of the paper is the following: first, we show a real migration example that we use to explain the automatic generation of a data migration plan. Section 3 presents the generation process, the information that it needs, the patterns used to generate a data migration plan and the applied methodology. In section 4, a brief summary of related work is given. Finally, the future work and conclusions are presented in section 5.

2. A Real Example of Migration

In the paper, the well known database example of bills-customers-products presented in Figures 1 and 2 is used. We take a part of the object model of the complete example to understand the reasons why the *Product* class of the example has been evolved. The migration plan generation is exemplified in the paper using the *Product* class evolution. The OASIS schemas capture the structure of the billing management of an information system. The new conceptual schema (figure 3) improves the previous one (figure 2). This improvement consists in:

- Partition of the initial customers. As a result, the *Customer*, the *Person* and the *Company* classes of the new conceptual schema will extract data from the *Customer* class of the old conceptual schema.
- A finer specification of the *Bill* class. As a result, the *Bill* and the *Bill_row* classes of the new conceptual schema will extract their data from the *Bill* class of the old conceptual schema.
- Currency change. As a result, all the prices of the products must be converted into euros.
- Some model errors corrections.

As a technical detail, it is worth noting that these schemas are modeled using OASIS. However, if we model them using UML, their appearance would be different. This is due to the fact that in OASIS the association is considered as a kind of aggregation and thus, the referential aggregation and the inclusive aggregation are represented by a lozenge. This is the reason why UML associations are modeled like OASIS referential aggregations. For example: The relationship between the classes *Bill_Row* and *Product* is an association represented by an OASIS referential aggregation.

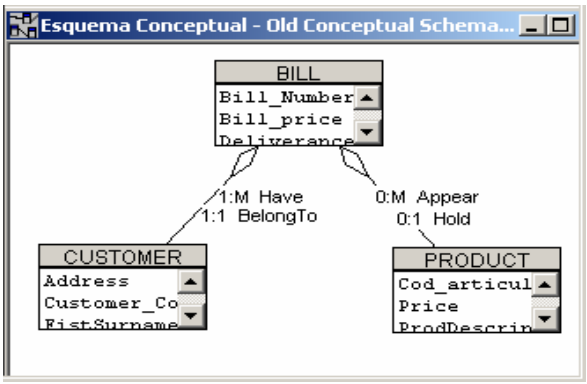


Fig. 2. Old Conceptual Schema

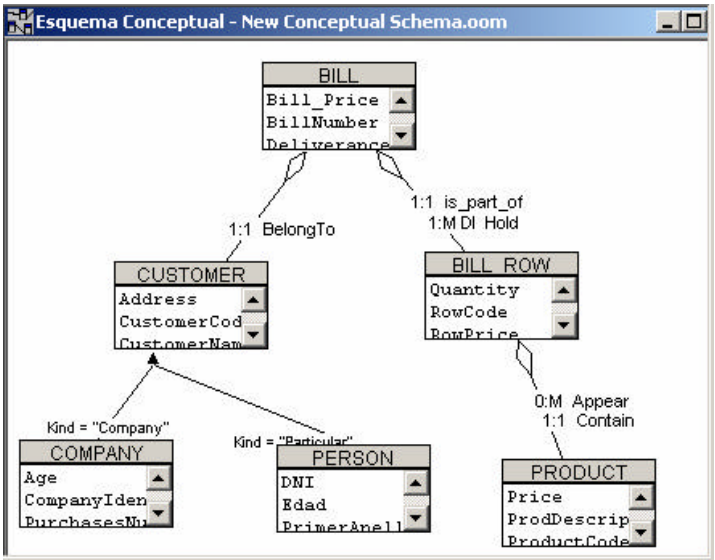


Fig. 3. Automatic Generation of a Data Migration Plan

3. Automatic Generation of a Data Migration Plan

A data migration plan is a set of data changes that are specified using a migration language (ADML). Its execution allows us to transfer data from an old database to a new database. A data migration plan must include all the necessary changes in order to perform a correct migration and in the right order. It is structured as the composition of the following different elements: migration expressions, changes and modules. A data migration plan is structured as a set of ordered modules, each module includes the neccesary migration expressions to migrate the data of a conceptual schema element. The highest abstraction level of this composition is constituted by migration modules whereas migration expressions are the lowest level. The definition of these elements is as follows:

- *Migration expressions:* Migration expressions are those expressions that can be specified in a data migration plan. Each type of migration expression has a different semantic and follows different syntactic restrictions. Examples of migration expressions are: Data Source, Transformation Function, Filters, etc.
- *Changes:* A change is the set of migration expressions that specify the updates undergone and the filters applied on the old database's instances.
- *Migration modules:* A migration module has a transactional behavior when it is executed by the ADAM tool. The composition of modules forms a data migration plan. Finally, it is important to note that there is one migration module for each class, aggregation and specialization of the new conceptual schema.

Each one of the changes that can be undergone in the old database are specified using a declarative language (a language which express properties of WHAT to do and not HOW do it). This migration language (ADML) ([Per02]) follows the object-oriented model. The main advantage of ADML is the independence from any DBMS due to its high abstraction level.

This language makes use of path expressions notation to specify changes that have been undergone in a conceptual schema element[2]. Conceptual schema elements are a set of origins of the data, transformations of the data and filters that will be applied on the data at the execution moment.

3.1. Data sources

The generation process of a migration plan needs several data sources to generate automatically the structure and the contents of the plan. The inputs of this process are the following:

- *Matchings between conceptual schemas:* A matching defines the old element from which must be obtained the data to provide to a determinate new element. This

[2] The conceptual schema elements that have been considered for the generation of a data migration plan are: classes, attributes, aggregations, associations and specializations.

information is available because the previous step of ADAM finds the matches between elements of both conceptual schemas. This step executes an algorithm that uses different comparison criteria in order to find the most suitable matches ([Sil01]). As a result, this process produces the correspondences between the elements of the conceptual schemas that have been compared. For example: The *Product* class of the new conceptual schema obtains its data from the *Product* class of the old conceptual schema (matches at class level) and the matches between their attributes are the following:

OLD CONCEPTUAL	NEW CONCEPTUAL
Cod_Product	ProductCode
ProductName	ProductName
ProdDescription	ProdDescription
UnitPrice	Price

Table 1. Matches between the attributes of the old Price class and the new Price class

– *Properties of the conceptual schema elements:* The properties of the conceptual schema elements are necessary in order to know the changes between the elements of a match instance and to generate the implied transformations. These transformations must be applied on the data of the old element in order to be compliant with the new element. This information is afforded by the conceptual schemas where only the properties of each element are defined. In our example we are going to focus on the following correspondence:

- The *Price* attribute of the *Product* class of the new conceptual schema obtains its data from the *Price* attribute of the *Product* class of the old conceptual schema. The properties of both attributes are included in the specification of the classes (see figures 4 and 5).

Atributos

Nombre	Tipo atributo	Tipo dato	Id	Tamaño	Valor defecto	Pedir al crear	Nulos
Cod_Product	Constante	Autonumerico	0			No	No
ProductName	Variable	String		15		Sí	Sí
ProdDescription	Variable	String		100		Sí	Sí
Price	Variable	Nat				Sí	No

Fig. 4. Attributes of the Product class of the old conceptual schema

Atributos

Nombre	Tipo atributo	Tipo dato	Id	Tamaño	Valor defecto	Pedir al crear	Nulos
ProductName	Variable	String		15		Sí	Sí
ProdDescription	Variable	String		80		Sí	Sí
Price	Variable	Real				Sí	No
ProductCode	Constante	Autonumerico	0			No	No

Fig. 5. Attributes of the Product class of the new conceptual schema

We can see at the figures 4 and 5 that the prices of the products were in pesetas, and now they must be in Euros. As a result, the data type of the *Price* attribute was *natural* in the old schema and is *real* in the new schema and also the *Price* value must be converted to the equivalent in euros. In addition, the attribute *UnitPrice* allowed null values and now the attribute *Price* does not allow null values.

– *Migration order:* Migration order is the order in which the data should be migrated. This preserves the database not to be in inconsistent states during the migration process. Moreover, this order affords the migration modules composition order and also the migration order between non-related modules. The computation of this order is made by means of an algorithm. The migration order algorithm analyses the structure of the new conceptual schema in order to obtain the relationships between elements. These relationships imply dependences determining the order to be followed in the data migration process ([Aba01]). The migration order obtained by the algorithm for the new conceptual schema of our example is the following: Customer, Bill, Product, Company, Person, Row_Bill.

3.2. Patterns

The automatic generation of a data migration plan implies generating its structure and its contents. First of all, ADAM generates the structure creating an empty migration module for each element of the new conceptual schema and includes the modules in the data migration plan following the computed migration order. Next, the module's content is automatically generated giving the migration expressions of each migration module and including them into their modules. Finally, a complete migration plan results. This automatic and complete generation of the migration plan is performed using two types of patterns: migration patterns and migration expressions patterns. We have specified them using the patterns design criteria proposed by [Ale79] and [Gam94]. They are available in a patterns catalog ([Per01-a]), where they can be identified by a number of pattern (**P-number**) and a title. Each pattern is composed of several sections that explain different qualities of the pattern. In the example patterns (see tables 2 and 3), the solution and the example sections are presented.
There is a migration pattern for each element and a migration pattern expression for each element's property. For this reason, it is possible to select the correct pattern by means of the property or element and to include the patterns in the code of the tool.

– **Migration expressions patterns**

There is a pattern for each one of the element's properties that can be changed by the schema evolution process and for any of the possible combinations of them. Each pattern contains a migration expression or a set of migration expressions specifying the correct transformation of data.

The generation of the migration expressions for a new element consists of determining which old element is related to it in the mapping and consulting their different properties. Next, the specific element pattern that specifies the migration expression code for the updated properties is instanciated and the resulting migration expressions are generated. Finally, these expressions are included in the new element module.

When the data migration plan is executed, the generated migration expressions of an element will be evaluated and the instances migrated to the new database. An example of a migration expression pattern is the one for an *attribute* when the "*name*", the "*data type*" and the "*not null value*" properties change (**P-12).**

In our example, ADAM uses the necessary patterns for each one of the established correspondences between the attributes of the new *Product* class and the old *Product* class. Moreover, we need take into account that the transformation function generated by the pattern (NatToReal(OldCS.Product.Price)) must be modified by the user in order to add the currency conversion function[3]. As a result, the transformation function that will be included in the data migration plan is PtsToEuro(NatToReal(OldCS.Product.Price)).

P-12: Pattern for an attribute when the "name", the "data type" and the "not null value" properties change.
Solution

The solution presents the generic migration expressions that specify the attribute changes of "name", "data type" and "not null value" properties. In this case, as in the **P-04**[4] and **P-08**[5] patterns, it is necessary to perform a type conversion in the transformation function as follows:
`old_data_typeTOnew_data_type (old_attribute)`
This pattern is a composition of the "name", the "data type" and the"not null value" property patterns (**P-03**[6]**, P-04** and **P-06**[7] or **P-08** and **P-10**[8] patterns). The migrations expressions that express these changes are the following:
`Filter: IDENT_clase '.' IDENT_attr`
 	
`Transformation_Function: generic_func '(' IDENT_clase '.' <IDENT_attr> ')'`

[3] The PtsToEuros conversion function is included in the ADAM's set of built-in transformation functions. If the user needs a function that is not provided by the tool, he can import the code of the new function and use it .

[4] **P-04:** Pattern for an attribute when the "data type" property change.

[5] **P-08:** Pattern for an attribute when the "name" and the "data type" properties change.

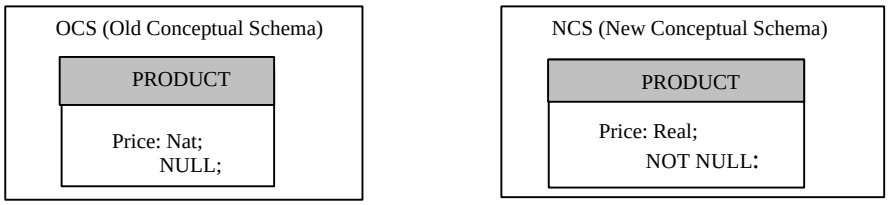
[6] **P-03:** Pattern for an attribute when the "name" property change.

[7] **P-06:** Pattern for an attribute when the "not null value" property change.

[8] **P-10:** Pattern for an attribute when the "name" and the "not null value" properties change.

Example

The prices of the products were in pesetas, and now they must be in euros. As a result, the data type of thw *Price* attribute was natural in the old schema and is real in the new schema and the *Price* value must be converted to €. In addition, *Number* allowed null values and now does not allow null values.



Text Format

```
Filter: OldCS.Product.Price
   Transformation_Function: NatToReal(OldCS.Product.Price)
```

XML Format

```
<Filter> OldCS.Product.Price <> NULL </Filter>
 <Transformation_Function> NatToReal(OldCS.Product.Price)
  </Transformation_Function>
```

Table 2. Solution and Example sections of the pattern P-12: Transformation function for the change of name, data type and not null value properties

– **Migration pattern**

Migration patterns establish the way of migrating data for each type of conceptual schema element together with the necessary actions to migrate each type of conceptual schema element and the allowed migration expressions for each one. We must take into account, at the design process of a migration pattern, the type of conceptual schema element that will use the pattern, because the transformations that can undergo each conceptual schema element are different. Moreover, migration patterns specify the expression place inside the module for each type of conceptual schema element, in the internal structure of a module.

A type of a conceptual schema element can have different associate patterns because there are different properties that influence the migration process. For example, the migration of a specialization relationship is different if its condition is more restrictive or less restrictive than the previous one or different because we must apply different types of filters on the data and different migration expression in a different place.

An example of a migration pattern is the pattern of the elemental class (**P-01)**:

P-01.Pattern: Elemental class

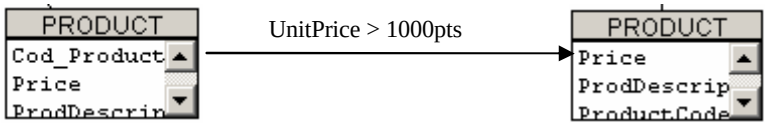
Solution

Let S be a set of schemas, C be an alphabet of classes, A be a set of attributes, G be a set of filters that are applied on old class population, GC be a set of conditions that are applied over old attributes, F be a set of transformation functions, SM be a set of matches between conceptual schemas, CM be a set of matches between classes of new and old conceptual schemas, and AM be a set of matches between attributes of conceptual schemas.

$S1, S2 \in S \wedge S1.C1, S2.C2 \in C \wedge S1.C1.a1, S2.C2.a2 \in A \wedge f_1, .., f_n \in F \wedge g_1,...,g_n \in G \wedge gc_1,...,gc_n \in GC \wedge SM1 \in SM \wedge CM1 \in CM \wedge AM1 \in CA \wedge SM1.old=S1 \wedge SM1.new=S2 \wedge CM1.old=S1.C1 \wedge CM1.new=S2.C2 \wedge AM1.old=S1.C1.a1 \wedge AM1.new=S2.C2.a2 \rightarrow$ data (S2.C2) = $\{y \mid \exists x \in data(S1.C1) \wedge \forall i ?_x g_i \; i=1,..,n \wedge ((y.a2 = f_n of_{n-1}...of_1(x.a1) \vee y.a2 = cte) \wedge \forall i ?_{x \downarrow a1} gc_i)\}$

Example

The *Product* class of the new conceptual schema obtains its data from the *Product* class of the old conceptual schema. However, the analyst of this system is only interested in the products that have a price higher than 1000. Moreover, all its attributes must be migrate with its transformations and conditions.



Text Format:

S1, S2 ∈ S ∧ S1.Product, S2.Product ∈ C ∧ S1.Product.Cod_Product, S1.Product.ProductName, S1.Product.ProdDescription, S1.Product.UnitPrice, S2.Product.ProductCode, S2.Product.Price, S2.Product.ProductName, S2.Product.ProdDescription ∈ A ∧ RealTONatural, RightTrunc ∈ F ∧ {S1.Product.UnitPrice > 1000pts}, {S1.Product.Price <> NULL} ∈ G ∧ SM1 ∈ SM ∧ CM1 ∈ CM ∧ AM1, AM2, AM3.AM4 ∈ CA ∧ SM1.old=S1 ∧ SM1.new=S2 ∧ CM1.old=S1.Product ∧ CM1.new=S2.Product ∧ AM1.old=S1.Product.Cod_Product ∧ AM1.new=S2.Product.ProductCode ∧ AM2.old=S1.Product.ProductName ∧ AM2.new=S2.Product.ProductName ∧ AM3.old=S1.Product.ProdDescription ∧ AM3.new=S2.Product.ProdDescription ∧ AM4.old=S1.Product.UnitPrice ∧ AM1.new=S2.Product.Price → data (S2.Product) = { y | ∃x ∈ data(S1.Product) ?$_x$ (S1.Product.Price > 31/12/1999 AND S1.Product.ProductName <> NULL AND S1.Product.ProdDescription <> NULL AND S1.Product.Price <> NULL ∧ (y.ProductCode = x.Cod_Product) ∧ (y.ProductName = x.ProductName) ∧ (y.ProdDescription = RightTrunc(x.ProdDescription) ∧ PtsToEuro(NatToReal(OldCS.Product.Price)) }

XML Format:

```
<New_Conceptual_Schema>
  <Class>
    <Name> Product </Name>
    <Origin>
      <Name> Product </Name>
      <Filtered>
        <Filter>
            OldCS.Product.Price > 31/12/1999 AND
            OldCS.Product.ProductName <> NULL AND
            OldCS.Product.ProdDescription <> NULL AND
            OldCS.Product.Price <> NULL
        </Filter
        <Attribute>
          <Name> ProductCode </Name>
          <OriginAttribute> OldCS.Product.Cod_Product </OriginAttribute>
          <Transformation_Function>
              OldCS.Product.Cod_Product
          </Transformation_Function>
        </Attribute>
        <Attribute>
          <Name> ProductName </Name>
          <OriginAttribute> OldCS.Product.ProductName </OriginAttribute>
          <Transformation_Function>
              OldCS.Product.ProductName
          </Transformation_Function>
        </Attribute>
        <Attribute>
          <Name> ProdDescription </Name>
          <OriginAttribute> OldCS.Product.ProdDescription </OriginAttribute>
          <Transformation_Function>
              RightTrunc(OldCS.Product.ProdDescription)
          </Transformation_Function>
    </Attribute>
        <Attribute>
          <Name> ProductName </Name>
          <OriginAttribute> OldCS.Product.Price</OriginAttribute>
          <Transformation_Function>
              PtsToEuro(NatToReal(OldCS.Product.Price))
          </Transformation_Function>
        </Attribute>
      </Filtered>
    </Origin>
  </Class>
</New_Conceptual_Schema>
```

Table 3. Solution and Example sections of the pattern P-01: Elemental Class

4. Related work

Nowadays, data evolution support is given by DBMS. Relational DBMS such as ORACLE ([Ora02]) or SQL Server ([Sql02]) and object-oriented DBMS like O2 ([Ard02]), Poet ([Poe02]) or Versant ([Ver02]).

Existing works in the schema evolution field focuses in the consistence problems between modified schemas and the corresponding databases. A set of invariants were defined by [Kim89] and several DBMS uses the same or similar invariant set. Existing solutions vary from those that use schema evolution (OTGen [Ler90] or Cocoon [Tre92]) to those that use schema versioning or views (Avance [Bjø88], CLOSQL [Mon93] or Multiperspectives [Odb95]). Our work follows the schema evolution solution due to our interest in application dynamics in order to be compliant with new requirements.

Several DBMS allow for data migration using their ETL (Extract, Transforma & Load) tools. This migration can be done by means of SQL statements or user defined scripts which can be executed on the database. However, these tools do not provide automatic support for the generation of these statements and scripts as theADAM tool does. For this reason, DB administrators must write migration code by hand. ETL tools offer a friendly interface to migrate data from an old database to a new database. However, they do not solve two important problems at the software maintenance stage: the high number of people required for the migration process and the high temporal cost.

The closest point of view to our approach is the TESS tool presented by Barbara Staund Lerner in her paper [Sta00]. Both approaches have two things in common: the processes are automatic and the tools are based on schema evolution. TESS uses an intermediate language generated from the relational schema code. This is an important difference with our approach, because we deal with directly the OO conceptual schemas and we do not have to translate them to an intermediate language. The OO conceptual schemas give us a higher abstraction level and save us from the translation process. It also implies that we do not have to take implementation details into account and so our process is more agile and has a higher abstraction level than the TESS process.

5. Further work and conclusions

The paper presents a solution to the data evolution problem of information systems. It explains the importance of this evolution problem and the need to support data changes in information systems.

The presented solution uses conceptual schemas, which represent the system evolution, to generate a data migration plan. This data migration plan is able to evolve and migrate data between the respective databases of the conceptual schemas. The automatic generation process gives us a first version of a data migration plan that can be modified by the analyst. The contents and the structure of this first version of a data migration plan are generated by means of a set of patterns.

This solution has been implemented in a migration tool whose name is ADAM. Nowadays, ADAM is being used together with Sosy Modeler®, a model compiler. ADAM has achieved to reduce the time invested in the maintenance stage and the number of employees involve in the data evolution task up to 80 % in real experiences.

The high abstraction level of the migration language ADML allows us to be independent from the underlying DBMS. In order to use a given DBMS, it is only necessary to construct its specific "driver". ADAM specifically is a compiler which generates DTS packets of SQL Server from migration plan .Using ADAM we do not care of implementation details at the comparison and plan generation processes. The final "driver" will do automatically the work.

Finally, it is important to note that it is possible to improve the automatic migration process. Therefore, there are some aspects that we must solve in the future: the migration language needs to be able to specify migration expressions that contain elements of the new conceptual schema and the migration process should take into account also the dynamics of the system (States Transition Diagrams (STD)), etc.

6. Bibliography

[Aba01] Abad, S, Carsí, J.A, Ramos, I, *How to obtain a data migration order for the elements of an OO conceptual schema*, Workshop on Evolution in the VI conference on Software Engineering and Databases , Almagro, Ciudad Real, Spain, November 2001 (in Spanish)

[Ana01] Anaya, V. *Generation of transformation modules to migrate data between databases using a data migration plan*, Msc Project, Computer Science Faculty, Valencia University of Technology, September 2001 (in Spanish)

[Ale79] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press. 1979.

[Ard] Ardent Software, *O2*, http://www.ardent.com/

[Bjø88] Bjørnerstedt A., Britts S., *Avance - An object management system*, Proc. of the Conference on Object-Oriented Systems, Languages and Applications (OOPSLA), San Diego, California, USA, pag. 206-221, September 1988.

[Cha00] Chaffin M., Knight B., Robinson T., Professional SQL Server 2000 DTS (Data Transformation Services), Wrox, 2000

[Gam94] Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addyson-Wesley.1994.

[Kim89] Kim W. and Lochovsky, F.H. (eds.), *Object-Oriented Concepts, Databases and Applications*, ACM Press, Addison-Wesley, 1989.

[Ler90] Lerner B.S., Habermenn A.N., *Beyond Schema Evolution to DatabaseReorganization*, In Proc. Of the Joint Conference on Object-OrientedSystems, Languages and Applications (OOPSLA) and ECOOP, Ottawa, Canada,pag.67-76, Octubre 1990.

[Mon93] Monk, S., *A Model for Schema Evolution in Object-Oriented Database Systems*, Ph. D. Thesis, Computing Department, Lancaster University, Febrero 1993.

[Odb95] Odberg, E., *Multiperspectives: Object Evolution and Schema Modificacion Management for Object-Oriented Databases*, Ph.D. Thesis, Department of Computer Systems and Telematics, Norwegian Institute of Technology, Febrero 1995.

[Ora] Oracle Corporation, *Oracle*, http://www.oracle.com

[Pas95] Pastor O. Et al., *OASIS versión 2 (2.2): A Class-Definition Language to Model Information Systems using an Object-Oriented Approach*, SPUPV-95.788, Universidad Politécnica de Valencia, España, 1995.

[Pas97] Pastor O. Et al, *OO-METHOD: A Software Production Environment Combining Conventional and Formal Methods*, Procc. of 9th International Conference, CaiSE97, Barcelona, 1997.

[Per02] Pérez J., Carsí J.A, Ramos I., *ADML: A Language for Automatic Generation of Migration Plans*, Lecture Notes in Computer Science (LNCS) by Springer Verlag, The First Eurasian Conference on Advances in Information and Communication Technology, Tehran, Iran, Octubre 2002 (pendiente de publicación)

[Per01-a] Pérez, J. *Generation of a data migration plan for OASIS conceptual schemas*, Msc Project, Computer Science Faculty, Valencia University of Technology, September 2001 (in Spanish)

[Per01-b] Pérez J., Carsí J.A, Ramos I., Anaya V., Silva J.F., *Data migration of conceptual schemas*, Workshop on Evolution in the VI conference on Software Engineering and Databases , Almagro, Ciudad Real, Spain, November 2001 (in Spanish)

[Poe] POET Software, *POET*, http://www.poet.com

[Rat] Rational Software, *Rational Rose*, http://www.rational.com/products/rose/

[Ser94] Sernadas A., Costa J.F., Sernadas C., "Object Specifications Through Diagrams: OBLOG Approach" INESC Lisbon 1994

[Sil02] Silva, J.F.., Carsí, J.A., Ramos, I., *Theoric analyze of the criteria of OO conceptual schemas comparison*, Ingeniería Informática Magazine, ISSN:0717-4197, Enero, http://www.inf.udec.cl/revista/edicion7/jsilva.htm

[SQL] Microsoft, *SQL Server*, http://www.microsoft.com/sql

[Sys] System Architec, http://www.popkin.com/products/sa2001/systemarchitect.htm

[Sta00] Staund Lerner, B., *A Model for Compound Type Changes Encountered in Schema Evolution*, ACM Transactions on Database Systems (TODS) Marzo 2000, Volumen 25 número 1

[Tre92] Tresch M., Scholl M.H., *Meta object Management and its applications to database evolution*, In Proc. Of the 11th International Conference on the Entity-Relationship Approach, Karlsruhe, Germany, pag.299-321, Octubre 1992

[Tog] TogetherSoft Corporation http://www.togethersoft.com/

[Ver] Versant Object Technology, *Versant*, http://www.versant.com/