Electronic Notes in Theoretical Computer Science 74 No. 4 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume74.html 13 pages

Data Reverse Engineering of Legacy Databases to Object Oriented Conceptual Schemas

⋆

J.Pérez [1], I.Ramos [2], V.Anaya [3], J.M.Cubel [4], F.Domínguez [5], A.Boronat [6], J.A.Carsí [7]

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
E-46071 Valencia - Spain

Abstract

This paper presents a solution and a methodology to recover legacy databases of most DBMS using formal-method based techniques. These formal methods (terms rewriting systems) are applied during the data reverse engineering process and allow for an automatic approach. This automatic approach reduces the time invested and the number of people involved in the data reverse engineering and data migration processes. This solution is being implemented in the RELS (Re-Engineering of Legacy Systems) tool. The RELS tool is under development in the Department of Information Systems and Computation of the Valencia University of Technology in collaboration with the industrial partner CARE-Technologies. RELS is used together with the model compiler Sosy Technology® of CARE-Technologies and provides a complete solution to the re-engineering proccess.

Key words: Re-engineering, Reverse engineering, Terms rewriting system (TRS), Data reverse engineering, Rewriting rules, Algebraic expressions, ADT(Abstract Data Type).

⋆ This work has been financed by the project CICYTof the program SIGLO with ref. TIC2000-1673-C06-01.

[1] Email: jeperez@dsic.upv.es
[2] Email: iramos@dsic.upv.es
[3] Email: vanaya@dsic.upv.es
[4] Email: jcubel@dsic.upv.es
[5] Email: fdominguez@dsic.upv.es
[6] Email: aboronat@dsic.upv.es
[7] Email: pcarsi@dsic.upv.es

1 Introduction

Technologies have a dynamic nature and they are continually evolving. For this reason, the systems must be able to adapt to new requirements. However, their development tools, DBMS or programming languages rapidly become obsoletes. As a consequence, these information systems become *Legacy Systems*. The recovery of legacy systems is a big problem for companies due to the cost of technological adaptation. In this paper, we propose a methodology and a tool to reduce this expense.

Our work is focused in legacy databases. For this reason, we apply *Data reverse engineering (DBRE)* to recover the obsolete databases. Data reverse engineering is a part of the Reverse engineering process. "Data Reverse Engineering deals with the tasks of understanding legacy databases and extracting their design specifications (domain semantics)" [2].

Nowadays, existing CASE tools are able to generate applications following the paradigm of automatic code generation. These CASE tools are called model compilers. They automatically generate the application code and the database schema starting from the conceptual schema of an information system.

Our tool recovers a legacy database obtaining an equivalent UML-like OASIS OO conceptual schema using formal methods and then, automatically generates a new SQL/Server database from the schema obtained. Finally, the data from the legacy database are correctly migrated to the new one (see Figure 1).

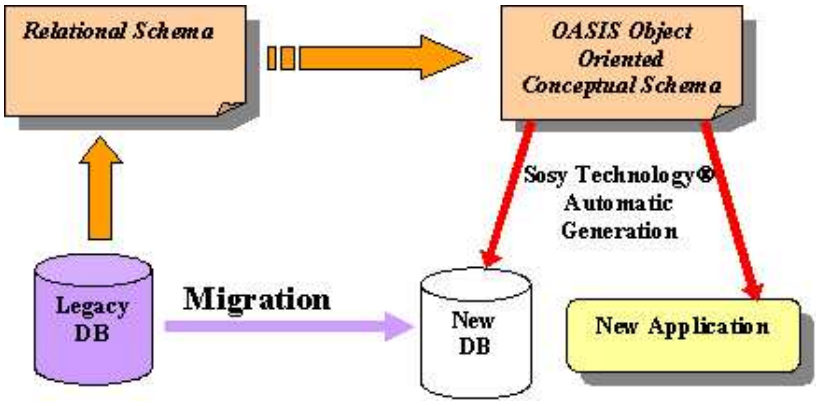

Fig. 1. Re-engineering process of RELS

This work is being developed in the Department of Information Systems and Computation of the Valencia University of Technology in collaboration with the industrial partner CARE Technologies. The data reverse engineering process and the data migration process followed by our approach will reduce the time invested and the number of people involved in the data evolution process. This maintenance improvement is due to the automatic tasks performed by the three phases of the tool. Despite the fact that these phases are

performed automatically, the results can be freely modified by the analyst. In this case, the process is semi-automatic. The three phases involved are the following:

1.- An OASIS OO conceptual schema is obtained by applying a data reverse engineering process in order to recover a legacy database. The relational and OO conceptual schemas are represented as terms which belong to two different term languages of two different ADTs. The correspondences between terms are specified using term rewriting rules. This paper is focused in this first phase.

2.- The rewriting rules applied in the first phase and the patterns used by Sosy Technology® to generate the new relational database are used to generate a data migration plan which is specified using a declarative language (a language which expresses properties of WHAT to do and not HOW to do it).

3.- The data migration plan is translated to *Data Transformation Service* packages of Microsoft whose execution automatically migrates data from the old database to the new database. These packages are finally executed by *SQL Server*.

Our work emphasizes the use of formal methods because they allow us to provides the user an automatic recover solution that is used by an industrial model compiler to improve the software maintenance process.

The structure of the paper is the following: Section 2 describes the data reverse engineering process. Section 3 presents how we mix the formal methods with current technologies in our implementation. Section 4 exemplifies the process of recovering a legacy database by means of an example. Section 5 provides a brief summary of related work. Finally, section 6 presents the future work and conclusions.

2 Data Reverse Engineering Process of RELS

The Data Reverse Engineering Process of RELS takes the relational model of the legacy database as input and generates an object-oriented model which is equivalent with the previous relational one. These models are represented as terms of the ADTs (Abstract Data Types) defined for the relational and object-oriented models.

The automatic generation of the object-oriented model is produced by applying the rewriting rules representing the correspondences between the relational elements [8] and the object-oriented elements [9]. These rewriting rules are applied by a term rewriting system (TRS) ([4]). The data reverse engineering process of RELS is constituted by three steps (see Figure 2):

[8] Elements that can be defined in a relational model

[9] Elements that can be defined in a object-oriented model

1.- The reading of the relational schema of a legacy database and its representation as a term of the defined relational ADT.

2.- The translation of the relational term into an object-oriented term using a TRS that applies the rewriting rules. The obtained object-oriented term which represents the object-oriented schema is compliant with the defined object oriented ADT.

3.- The storage of the object-oriented schema and the applied rewriting rules that will be used in the second phase of RELS.

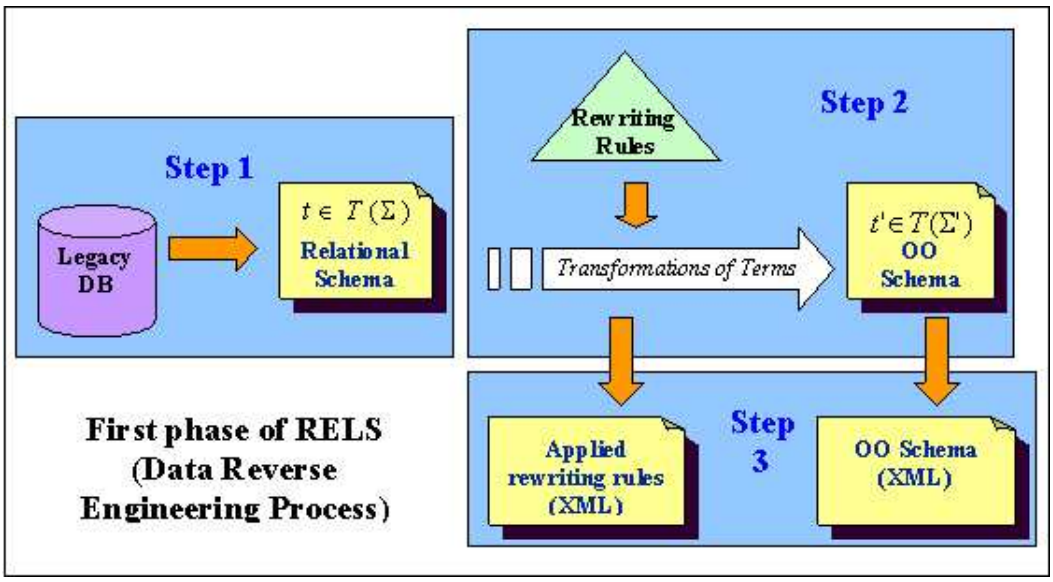

Fig. 2. Data Reverse Engineering Process of RELS

2.1 Relational Model

The relational conceptual schema of the legacy system is captured from the stored database of a specific DBMS. This structure is obtained by using a set of operations that allows it to read a relational schema specification. The relational conceptual schema is represented as an algebraic term which is based on the syntactic and semantic rules provided by an ADT. For this reason, an ADT which expresses the relational model is proposed. This ADT is composed by a group of specification modules which contain operations (constructors or functions) and axioms that determine which relational elements are considered and how they can be combined to express a correct relational schema. A specification module is associated with any relevant element of the relational model. There is a natural compositional order between elements of the relational model (for example, tables are composed by columns), so there is a compositional relation between the specification modules of the relational ADT. Therefore, a well structured ADT is obtained.

In each specification module, a set of constructors, functions and axioms

are provided. The constructors are applied to form a term. A module, which is composed of other modules, combines all the subterms and carries out the term that expresses the relational conceptual schema.

Once the relational conceptual model term is obtained, some user interaction may be required. Some legacy systems are very old and were constructed using DBMS or other repository forms which did not allow for the definition of constraints (either integrity or reference constraints). These constraints were implemented by code, so they are not explicitly presented in the legacy database structure. User interaction may be necessary to provide additional information to obtain a complete relational conceptual schema. Users can also express the formulae of derived attributes. The subprocess of enriching the conceptual schema is done over the relational term, but our tool has an interface that hides the algebraic notation, providing a friendly graphical user interface.

```
SPEC   m-rel
USING  table + column + data_type + nut_nul + ...
SORTS  m-rel
OPS    create_database: --> m-rel
       add_table: table m-rel --> m-rel
       add_column: column data_type not_null table m-rel --> m-rel
       add_ctr_pk: col_list table m-rel --> m-rel ...
ENDSPEC
```

Fig. 3. Part of the Relational ADT

2.2 Object-Oriented Model

After applying the rewriting rules to the relational conceptual schema term, an object-oriented term is obtained. This OO term is expressed by means of the constructor operations and by using the axioms provided by the ADT that represent the OASIS OO Model. The OO ADT specification modules correspond to OO elements and properties of these elements. A module that is composed of other modules expresses how to generate terms that combine the rules of its components.

Hence, the m-oo (OO model specification module) provides the rules to generate OO conceptual schema terms. The OO conceptual schema term is then translated to a XML document that can be interpreted by the SOSY Technology® model compiler.

2.3 Correspondences between the Relational Model and the Object-Oriented Model

A detailed study has been done to obtain the correspondences between the relational ADT and the object-oriented ADT. We have determined for each element of the relational ADT, the different elements of the Object-Oriented ADT that can be equivalent. This has allowed us to define the transformations

```
SPEC   m-oo
USING  class + aggregation + specialization + ...
SORTS  m-oo
OPS    create_schema: --> m-oo
       add_class: class m-oo --> m-oo
       add_identif: attr_list class m-oo --> m-oo
       add_ctt_att: attribute data_type bool class m-oo --> m-oo
       add_vbl_att: attribute data_type bool class m-oo --> m-oo
       add_aggregation: aggregation class class rol rol nat nat
                        nat nat bool bool bool bool m-oo --> m-oo ...
ENDSPEC
```

Fig. 4. Part of the Object Oriented ADT

(set of rewriting rules) to convert a relational term into an object-oriented term.

Our TRS is finite and non-confluent, because we can obtain several object-oriented terms from a relational term (different possible representations). The initial term is constituted by subterms of the relational ADT and the following terms (in the rewriting process) include terms of the object-oriented ADT. However, subterms of both ADTs coexist in the intermediate terms. Those intermediate terms belongs to the mixed ADT m-rel-oo.

The application of the rules is automatic, but the user's knowledge is necessary in order to validate whether the applied rule is semantically correct. This is due to the fact that an element of the relational model can be represented by several elements of the OO model. For this reason, the RELS' proposal of applying a specific rewriting rule can be changed by the user. This change is controlled by the tool; the user can only modify the proposed rewriting rule by using another rule from the set of rewriting rules that are syntactically correct in the following rewriting step. We take into account the criterion that legacy databases were usually designed to improve the efficiency of access to the data. This reduces the interactions required by the users.

3 Implementation

The data reverse engineering process of RELS uses several technologies in order to develop the different phases. This tool is under development using Visual Studio .NET because of its advantages and is one of the programming platforms for the future.

Moreover, an API has been defined to read the relational schema. This API has been created based on ADOX Object Model ((Microsoft ActiveX Data Objects Extensions for Data Definition Language and Security). From the services provided by the API, the application extracts the necessary information to obtain the relational term. RELS uses this API because it provides independence from the DBMS. As a result, RELS reads the relational schema from most legacy DBMS.

The relational and the object-oriented ADTs, the set of rewriting rules which can be applied, and the algebraic terms are specified using MAUDE

[7] (an OBJ dialect). MAUDE also provides the rewriting mechanism that executes the rules in order to transform the relational term into the object-oriented term. We chose HASKELL to implement this phase of the data reverse engineering process because is a functional language and is supported by Visual Studio .NET.

Finally, the outputs of this phase are stored as XML documents to be used by the second phase of RELS. Thus, there are two different XML DTDs to generate each document:

1.- The Sosy Technology® XML DTD: The generated object-oriented term is specified in a XML document which is compliant with this DTD.

2.- The rewriting rules XML DTD: The applied rewriting rules in a specific data reverse engineering process are stored in a XML document that follows this DTD.

The XML document that contains the object-oriented term is used by

```
SPEC   m-rel-oo
USING  m-rel + m-oo + ...
SORTS  m-rel-oo
SUBSORTS
       m-rel < m-rel-oo
       m-oo < m-rel-oo
OPS    ...
FOR ALL
       c, cr, cs, c1, c2, c3, c4, c5, c6, c7, c8: column ...
AXIOMS create_database() --> create_schema()
       add_column(c, d, n, t, m) -->
          add_vbl_att(c, d, n, cat("AGGR-",t, "-1"), m) IF aggregation(cat("AGGR-",t),m)
       add_column(c, d, n, t, m) -->
          add_vbl_att(c, d, n, cat("AGGR-",t, "-2"), m) IF aggregation(cat("AGGR-",t),m)
       add_column(c, d, n, t, m) -->
          add_ctt_att(c, d, n, cat("AGGR-",t, "-1"), m) IF aggregation(cat("AGGR-",t),m)
       add_column(c, d, n, t, m) -->
          add_ctt_att(c, d, n, cat("AGGR-",t, "-2"), m) IF aggregation(cat("AGGR-",t),m)
       add_unique(cr, t, add_unique(cs,t,add_column(c1, d1, n1, t, add_column(c2, d2, n2, t,
       add_column(c3, d3, n3, t, add_column(c4, d4, n4, t, add_column(c5, d5, n5, t,
       add_column(c6, d6, n6, t, add_column(c7, d7, n7, t, add_column(c8, d8, n8, t,
       add_table(t,m)))))))))) -->
          add_unique(cr,t, add_unique(cs, t, add_column(c1, d1, n1, t, add_column(c2, d2, n2, t,
          add_column(c3, d3, n3, t, add_column(c4, d4, n4, t, add_column(c5, d5, n5, t,
          add_column(c6, d6, n6, t, add_column(c7, d7, n7, t, add_column(c8, d8, n8, t,
          add_class(t, m)))))))))))
       add_unique(cr,t, add_unique(cs,t, add_column(c1, d1, n1, t, add_column(c2, d2, n2, t,
       add_column(c3, d3, n3, t, add_column(c4, d4, n4, t, add_column(c5, d5, n5, t,
       add_column(c6, d6, n6, t, add_column(c7, d7, n7, t, add_column(c8, d8, n8, t,
       add_table(t,m)))))))))) -->
          add_unique(cr,t, add_unique(cs,t, add_column(c1, d1, n1, t, add_column(c2, d2, n2, t,
          add_column(c3, d3, n3, t, add_column(c4, d4, n4, t, add_column(c5, d5, n5, t,
          add_column(c6, d6, n6, t, add_column(c7, d7, n7, t, add_column(c8, d8, n8, t,
          add_aggregation(cat("AGGR-",t), cat(t,"-1"), cat(t,"-2"), cat("rol",t,"-1"),
          cat("rol", t,"-2"), 1, 1, 1, 1, false, true, false, false, add_class(cat(t,"-2"),
          add_class(cat(t,"-1"),m))))))))))))) IF notnull(cr,cs,t) ...
ENDSPEC
```

Fig. 5. Part of the mixed m-rel-oo ADT

Sosy Technology® model compiler, and the XML document of the applied rewriting rules is used by the second phase of RELS.

4 An Example

In this section we present an example of recovering a legacy database. The example takes a part of the database to exemplify the process followed by the RELS tool. The legacy database belongs to a car insurance company and the example is focused on the relation between the insurance policy and the car.

The insurance company manages its insurance policies in the following way:

- An insurance policy insures only one car and at least one.
- A car, which is stored in the database, must be insured by one insurance policy and only one.

The relations between cars an insurances policies are stored in one table in the legacy database. The SQL relational schema which specifies this table is presented in the figure 6.

```
CREATE TABLE car_insurance_policy(
  insurance_num  int        NOT NULL  UNIQUE,
  enrol_date     date       NOT NULL,
  deadline       date       NOT NULL,
  type           char(4)    NOT NULL,
  price          int        NOT NULL,
  licence        int        NOT NULL  UNIQUE,
  make           char(10)   NOT NULL,
  model          char(10)   NOT NULL,
  PRIMARY KEY (insurance_num, licence))
```

Fig. 6. The SQL relational schema of the example

The relational term, who specifies the relational schema in figure 6, is constructed by RELS taking into account the designed relational ADT. The API, which read the relational schema, is used in the process of term construction. As a result of this process, RELS obtains the following term:

```
add_ctr_pk(insurance_num, licence, car_insurance_policy,
add_ctr_unique(insurance_num, car_insurance_policy,
add_ctr_unique(licence, car_insurance_policy,
add_column(model, string, false, car_insurance_policy,
add_column(make, string, false, car_insurance_policy,
add_column(licence, int, false, car_insurance_policy,
add_column(price, int, false, car_insurance_policy,
add_column(type, char, false, car_insurance_policy,
add_column(deadline, date, false, car_insurance_policy,
add_column(enrol_date, date, false, car_insurance_policy,
add_column(insurance_num, int, false, car_insurance_policy,
add_table(car_insurance_policy, create_database()))))))))))))
```

We must take into account that the table represents in the OO model an aggregation between the *Insurance_policy* class and the *Car* class (see figure 7). The *Car* class is the component of the aggregation and the *Insurance_policy*

class is the composed of the aggregation. As we can see, this information will be provided by the user during the re-writing proccess, which transform the relational term into the OO term.

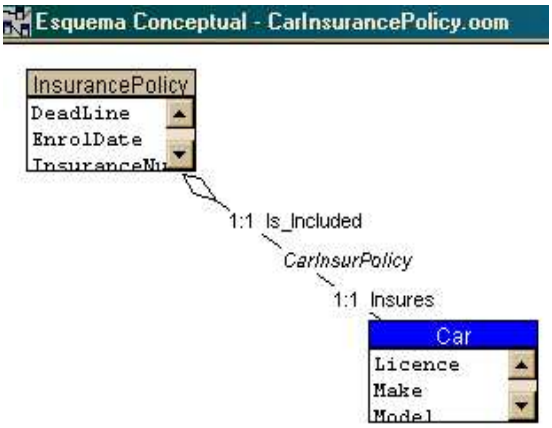

Fig. 7. OO Model

Following is presented the process that applies the rewriting rules in order to translate the relational term into a object oriented term of our example. In this paper, we provides some of the rules which will be presented to the user by means of a graphical user interface (GUI). In this example, we emphasize the changes of the user, it is the moment in which the user change the default rule by another one.

(i) First rewriting rule that it is proposed and applied by RELS.

```
create_database() --> create_schema()
```

(ii) Second rewriting rule that it is proposed by RELS.

```
add_ctr_unique(insurance_num, car_insurance_policy, add_ctr_unique(licence, car_insurance_policy,
add_column(model, string, false, car_insurance_policy, add_column(make, string, false, car_insurance_policy,
add_column(licence, int, false, car_insurance_policy, add_column(price, int, false, car_insurance_policy,
add_column(type, char, false, car_insurance_policy, add_column(deadline, date, false, car_insurance_policy,
add_column(enrol_date, date, false, car_insurance_policy, add_column(insurance_num, int, false,
car_insurance_policy, add_table(car_insurance_policy, m)))))))))) -->
add_ctr_unique(insurance_num, car_insurance_policy, add_ctr_unique(licence, car_insurance_policy,
add_column(model, string, false, car_insurance_policy, add_column(make, string, false, car_insurance_policy,
add_column(licence, int, false, car_insurance_policy, add_column(price, int, false, car_insurance_policy,
add_column(type, char, false, car_insurance_policy, add_column(deadline, date, false, car_insurance_policy,
add_column(enrol_date, date, false, car_insurance_policy, add_column(insurance_num, int, false,
car_insurance_policy, add_class(car_insurance_policy, m)))))))))))
```

(iii) Second rewriting rule that it is applied by RELS. The rule is selected by the user between the rules that can be executed.

```
add_ctr_unique(insurance_num, car_insurance_policy, add_ctr_unique(licence, car_insurance_policy,
add_column(model, string, false, car_insurance_policy, add_column(make, string, false, car_insurance_policy,
add_column(licence, int, false, car_insurance_policy, add_column(price, int, false, car_insurance_policy,
add_column(type, char, false, car_insurance_policy, add_column(deadline, date, false, car_insurance_policy,
add_column(enrol_date, date, false, car_insurance_policy, add_column(insurance_num, int, false,
car_insurance_policy, add_table(car_insurance_policy, m)))))))))))) -->
add_ctr_unique(insurance_num, car_insurance_policy, add_ctr_unique(licence, car_insurance_policy,
add_column(model, string, false, car_insurance_policy, add_column(make, string, false, car_insurance_policy,
add_column(licence, int, false, car_insurance_policy, add_column(price, int, false, car_insurance_policy,
add_column(type, char, false, car_insurance_policy, add_column(deadline, date, false, car_insurance_policy,
add_column(enrol_date, date, false, car_insurance_policy, add_column(insurance_num, int, false,
car_insurance_policy, add_aggregation(AGGR-car_insurance_policy, car_insurance_policy-2,
car_insurance_policy-1, rol_car_insurance_policy-2, rol_car_insurance_policy-1, 1, 1, 1, 1, false, true,
false, false, add_class(car_insurance_policy-2), add_class(car_insurance_policy-1), m)))))))))))
/* IF notnull(insurance_num, licence, car_insurance_policy) */
```

(iv) Third rewriting rule proposed by RELS.

```
add_column(insurance_num, int, false, car_insurance_policy, m) -->
add_vbl_att(insurance_num, int, false, car_insurance_policy-1, m)
/* IF aggregation(AGGR_car_insurance_policy,m) */
```

(v) Third rewriting rule applied by RELS. The rule is selected by the user between the rules that can be executed.

```
add_column(insurance_num, int, false, car_insurance_policy, m) -->
add_ctt_att(insurance_num, int, false, car_insurance_policy-1, m)
/* IF aggregation(AGGR_car_insurance_policy,m) */
```

As a result of this process RELS obtains the object oriented term (see figure 8) and the XML document with the applied rules (see figure 9).

```
add_identif(licence, car_insurance_policy-2,
add_identif(insurance_num, car_insurance_policy-1,
add_unique(licence, car_insurance_policy-2,
add_unique(insurance_num, car_insurance_policy-1,
add_vbl_att(model, string, false, car_insurance_policy-2,
add_vbl_att (make, string, false, car_insurance_policy-2,
add_ctt_att (licence, int, false, car_insurance_policy-2,
add_vbl_att (price, int, false, car_insurance_policy-1,
add_vbl_att (type, char, false, car_insurance_policy-1,
add_vbl_att(deadline, date, false, car_insurance_policy-1,
add_ctt_att(enrol_date, date, false,car_insurance_policy -1,
add_ctt_att(insurance_num, int, false, car_insurance_policy-1,
add add_aggregation(AGR-car_insurance_policy, car_insurance_policy-2,
car_insurance_policy-1, rol_car_insurance_policy-2,
rol_car_insurance_policy-1, 1, 1, 1, 1, false, true, false, false,
add_class(car_insurance_policy-2),
add_class(car_insurance_policy-1),
create_schema()))))))))))))))
```

Fig. 8. The object oriented term

For each applied rule, the XML document only includes the subterm of the rule that has been modified. The figure 9 presents the XML document generated for the second rewriting rule (step 3) that has been applied by RELS.

```
<MERToMOO>
   <rule>
      <m-er>
         <table> car_insurance_policy </table>
      </m-er>
      <m-oo>
         <aggregation>
            <name> AGGR-car_insurance_policy </name>
            <composed> car_insurance_policy-1 </composed>
            <component> car_insurance_policy-2 </component>
            <rol_composed> rol-car_insurance_policy-1 </rol_composed>
            <rol_component> car_insurance_policy-1 </rol_component>
            <min_composed> 1 </min_composed>
            <max_composed> 1 </max_composed>
            <min_component> 1 </min_component>
            <max_component> 1 </max_component>
            <dynamic> false </dynamic>
            <inclusive> true </inclusive>
            <id_dependency_composed> false </id_dependency_composed>
            <id_dependency_component> false </id_dependency_component>
         </aggregation>
         <class> car_insurance_policy-1 </class>
         <class> car_insurance_policy-2 </class>
      </m-oo>
   </rule>
</MERToMOO>
```

Fig. 9. The XML document of the third rule applied in the example by RELS

5 Related Work

There are many works that propose database reverse engineering approaches. However, none of them provide a specific industrial tool that offers anything more than just a theoretical result. The common aim of the related works is to obtain an abstract description of the information system with different final purposes (documenting, migrating, etc). Thus, [1] and [6] propose two reverse engineering processes where an entity relationship schema is obtained from a legacy relation schema. This ER schema constitutes the abstract description, but is poorer than an OO model because it cannot be completed by system behaviour. Other works ([3,9,10]) provide OO models from relation schemas. [2] and [8] attempt to reduce the user interaction but require imposing assumptions (3NF relational schemas) and require more information to induce some knowledge (such as instances of legacy database, application source code, etc). We prefer a stronger user interaction because then it is not necessary to impose assumptions or require many inputs and mainly because it is more likely to achieve an accurate model of legacy system.

There are other tools as DB-Main [11] that apply a data reverse engineering process, but their aim is different to RELS because they do not perform a conversion between different models and they do not migrate the data from the legacy database to the new database. These tools recover the conceptual schema from the logic schema in order to obtain traceability between different layers of the database, to create new databases in other DBMS and to reduce

the dependence on the technology.

In [5] the Varlet Database reverse engineering process is explained. The Varlet tool transforms a relational schema into an OO conceptual schema, and migrates the legacy data to the new OO database. Our approach is different: a relational database schema, implementing the persistence layer of an OASIS object society is the migration target. Moreover, in Varlet, the legacy relational schema is enriched with semantic information which are extracted from several sources as the application source code. However, in our approach, this semantic information is given by the user in an interactive way. The last difference between both approaches is the moment in which the transformation rules, for obtaining the desired conceptual schema, are applied. The Varlet approach changes the initial OO conceptual schema produced by the generation process and our approach changes the proposed rules during the generation process.

6 Conclusions and Future Works

This paper presents a solution and a methodology to recover legacy databases of most DBMS using formal-methods based techniques. The presented solution blends formal methods (Abstract Data Types, Term Rewriting System (MAUDE, OBJ)) with current technologies (HASKELL, XML, SQL-Server) following an automatic approach. As a result, it reduces the time invested and the number of people involved in the data reverse engineering and data migration processes. Moreover, this methodology recovers the legacy databases of most legacy DBMS due to the high abstraction level used and the technologies that has been applied. The solution is being implemented in the Department of Information Systems and Computation of the Valencia University of Technology in collaboration with the industrial partner Care-Technologies.

Finally, it is important to note that it is still possible to improve the data reverse engineering process presented. More work is necessary to define the criteria determining the application of the rewriting rules and also in increasing the usability of the user interface. In addition, more work is necessary to take into account the behavior of the legacy system (code) to automatically obtain the information that is not specified in the schema of the database (constraints, derived expressions, etc).

References

[1] Andersson, M., *Extracting an entity relationship schema from a relational database through reverse engineering.* In Proceedings of ER'94, LNCS, 403–419. Springer-Verlag, 1994.

[2] Chiang, R. H. L., Barron, T. M., Storey, V. C., *A framework for the design and evaluation of reverse engineering methods for relational databases*, Data &

Knowledge Engineering **12** (1997), 107–142, Elsevier Science.

[3] Hainaut, J. L., Henrard, J., Roland, D., Englebert, V., Hick, J. M., *Structure Elicitation in Database Reverse Engineering*, WCRE'96 **3** (1996), 131–140.

[4] Haskell, URL: `http://www.haskell.org/`.

[5] Jahnke,J.H., Zundorf,A. *Using Graph Grammars for Building the Varlet Database Reverse Engineering Environment*. Theory and Application of Graph Transformations (TAGT'98), Paderborn, Germany, Technical Report tr-ri-98-201, University of Paderborn,1998.

[6] Johannesson, P., *A Method for transforming Relational Schemas into Conceptual Schemas*, in Proc. Of the 10 th International Conference on Data Engineering, Rusinkiewicz (Ed.), 115–122, Houston, IEEE Press, 1994.

[7] The Maude System, URL: `http://maude.csl.sri.com/`.

[8] Petit, J. M., Toumani, F., Boulicaut, J.F., Kouloumdjian, J., *Towards the Reverse Engineering of Denormalized Relational Databases*, in Proc. of the 12 th International Conference on Data Engineering, New Orleans, Louisiana, USA, IEEE Press, Feb. 1996.

[9] Premerlani, W.J., Blaha, M., *An approach for reverse engineering of relational databases*, Communications of the ACM **37,5** (1994), 42–49.

[10] Ramanathan, S., Hodges, J., *Reverse Engineering Relational Schemas to Object-Oriented Schemas*. Technical Report MSU-960701, Mississipi State University, Mississipi, 1996.

[11] DB-Main, URL:
`http://www.fundp.ac.be/recherche/unites/publications/en/2987.html`.