RDB2 Model: A Relational Databases Retro-Design Tool Enriched by the Extraction of Constraints

Widad Jakjoud*, Mohamed Bahaj
Laboratory LITEN, University Hassan 1, 26000 Settat, Morocco.

* Corresponding author. Email: jakjoudw@gmail.com
Manuscript submitted March 13, 2017; accepted May 24, 2017.
doi: 10.17706/jsw.12.6.426-438

Abstract: In this paper, we present a method that retro-designs a relational database to generate a conceptual model by extracting data structures and constraints: Our method extracts a conceptual model directly from an existing relational database without using the ER model (Entity / Relationship). The method is proposed as a two-step process, the first is to extract the data structure of the relational database and the conceptualization of this data structure by generating a conceptual model. The second stage involves the extraction of relational constraints and the semantic enrichment of the conceptual model by improving the understanding of relationships between data structures.

Key words: RDB, model, meta-model, constraint, transformation, semantic.

1. Introduction

The reverse engineering is the discipline that combines all of the techniques and tools related to the understanding of an existent software system (or equipment) [1]. Its goal is to recover the design model in order to recreate the abstraction model from the source of information (physical schema of data source, source code, documentation,). The reverse engineering is based on two essential steps: the identification of the system components and their relationships then the representation at a higher level of abstraction [2].

For the data sources, the reverse engineering allows the analysis and understanding their structure and content for the production of conceptual components (concepts and relationships between concepts). These conceptual components will be used to reproduce the conceptual model. The structure of the relational database is a challenge for the reinterpretation of these components into conceptual components.

In the literature, several approaches that distinguish among the components of the data source, components which represent conceptual entities, properties, the manner of grouping them, and finally those that represent links of associations, and links of generalization / specialization. We can classify the works in this domain in three families of approaches following the levels operated in analysis; we quote the methods based on the analysis of data structure description (Data Description Level) such as [3],[4]. The methods which extend the analysis of data manipulation mechanisms (Data Manipulation Level) like the joints in requests, the triggers, and stored procedures [5]-[8].We quote another method [9] which mainly interested on data manipulation mechanisms.

We also include methods that exploit the two levels already described and expand the analysis to the data tuple (Data Level) [10], [11]. The most of these methods are implemented in modeling environments that allow the extraction of data structures from existing applications and then generate a conceptual model using interactivity between the environment and an expert user who validate the extracted structures.

2. Approach Description

We present an approach to retro design a relational database for generating a conceptual model which is instance of our proposed conceptual meta model.

Our approach is implemented as a tool named RDB2MODEL. It consist to define generation rules allowing to instantiate the conceptual meta model directly from the relational schema of database.

Therefore, we propose the mapping rules from a technological space to another. The Retro design, thus described, ensures the preservation of semantics through the generation of various constraints that may include the relational schema.

To manipulate the conceptual meta model, we generate it as a plug-in named mmconceptual. We use the interface mmconceptualFactory to instantiate meta conceptual elements: Given a relational database, we extract the relational schema from which the Factory mechanism instantiates the meta model to generate an empty model.

From the elements of the relational schema, conceptual elements will be generated and added to the model thus created

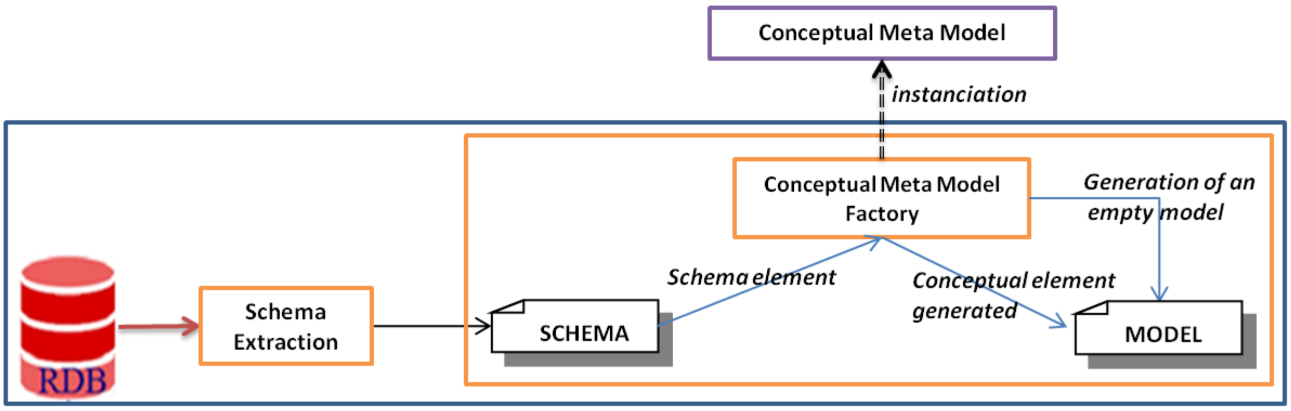
Conceptual Meta Model
instanciation
Conceptual Meta Model Factory
Generation of an empty model
Schema element
Conceptual element generated
Schema Extraction
SCHEMA
MODEL
RDB

Fig. 1. Approach description.

2.1. Conceptual Meta-Model

We define a conceptual Meta model in Fig.2, based on the UML Meta model taking into account some simplifications and modifications to in order to tailor it to our study context:

Our meta model is a class diagram composed of a set of conceptual elements : classes, types, associations, constraints, etc.

A class is characterized by an identifier and a name. It can be composed by several classes and can inherit several classes A class inherits from a data type composed by a set of properties. Each property have a name and a type.

An association is composed by arities (end of association), each arity connect a source class to a target class. An arity is characterized by its identifier and its role and it inherits the conceptual element ElementMultiplicity which defines a maximal and minimal cardinality.

The arties are in number of participating classes in the association.

We can define constraints on the elements classes, properties and arities. :

The Constraint element is the meta class that represents the constraints. The meta class ConstraintElement represents a conceptual element on which the constraint applies. In our case, we focus on the conceptual elements class, property and arities.

DefaultConstraint represent the usual constraints such as : key, refkey, nullable, exclusivity (XOR), frozen, subset,.

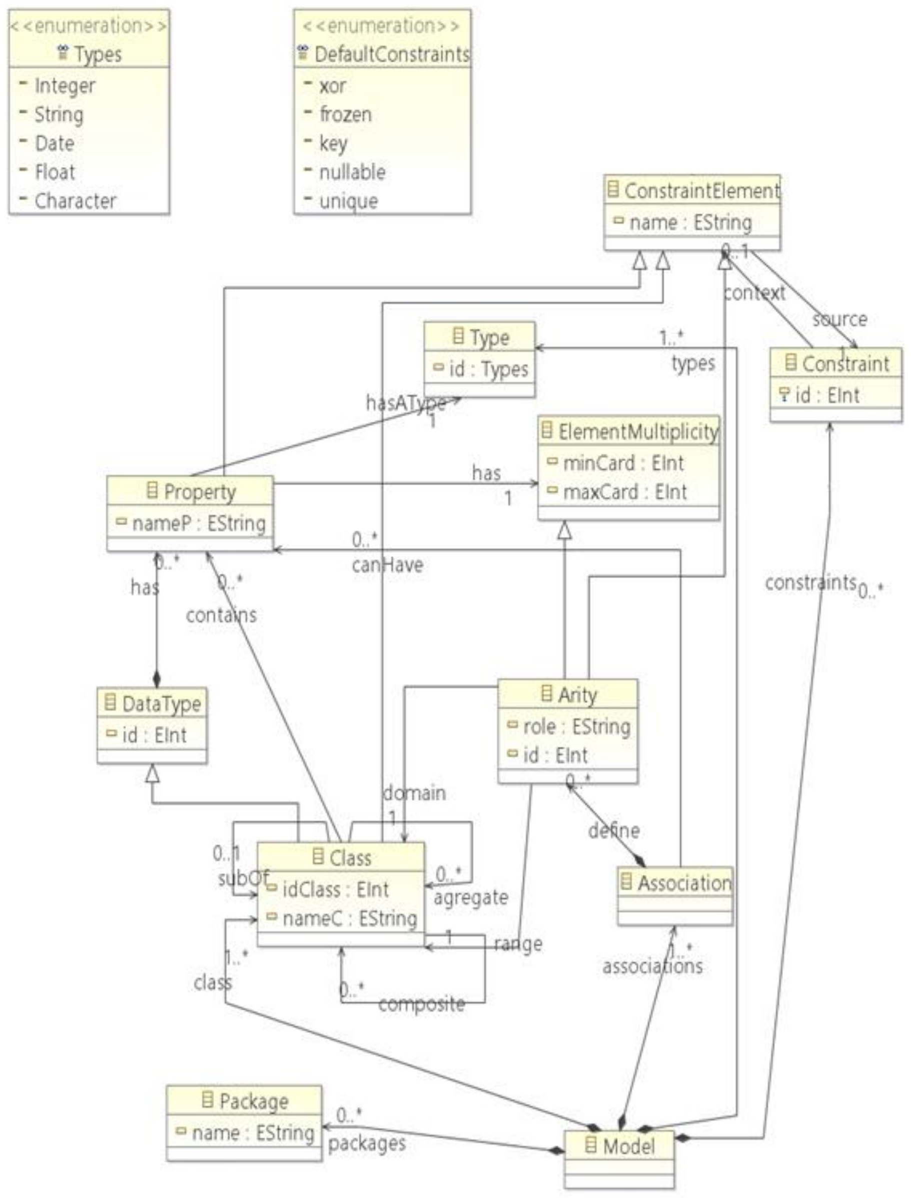

Fig. 2. The Conceptual meta-model.

2.2. Description of the Relational SCHEMA

In the relational model, a database is described by its schema. A schema as defined by Codd is represented by the following concepts:

An attribute is an identifier describing an information.

A domain of an attribute is a set of values having a syntactic type and a semantic

A relation is a Cartesian sub-set of domains and a set of integrity constraints on the constituents of the relation.

Relation = {{Di/i=1..n}, {D_CIj/j=1..m}}

The relational schema is a set of relations and integrity constraints on those relations

Schema = {{Ri/i=1..n}, {R_CIj/j=1..m}}

We propose a simplified meta model of relational schema

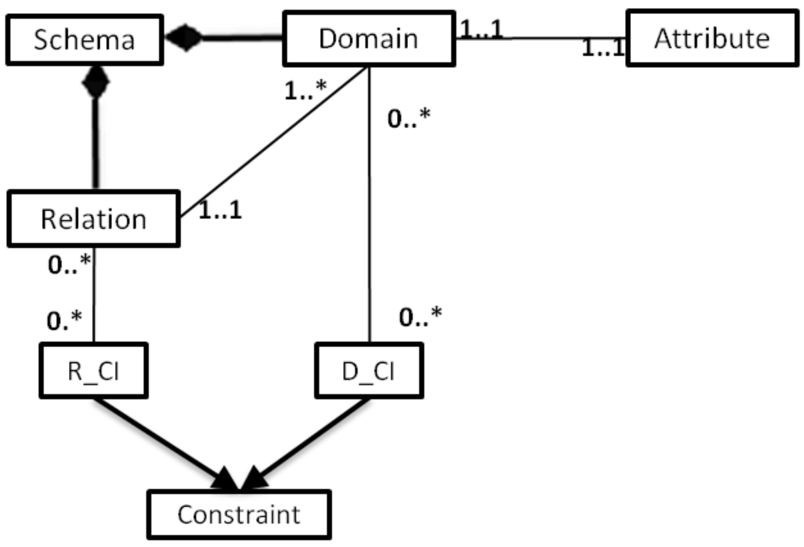

Fig. 3. The simplified meta model of relational schema.

2.3. Extraction of Constraints

2.3.1. Relational integrity constraints

For information systems, it is essential to ensure the integrity and consistency of the database, it is a permanent concern on which the reliability of the entire system [12], [13]. The concept of integrity includes several aspects, namely the technical aspect (the competition control, data security...) that serves to maintain the database consistency in the case of technical problems. Also ensures the semantic aspect responsible for the preservation of the quality of data stored in the database by mutual validation of semantic values and structures of the data. If it happens that a change may disrupt data integrity, the change must be banned. In fact, integrity is an assertion expressed at the conceptual level (Conceptual Model Data) and then implemented at the physical level. This is called semantic rules commonly known: integrity constraints. An integrity constraint is the assurance of the data concordance with the system that database represents. A database is semantically integrated when the constraints of integrity is respected by the entire database. Several classifications of integrity constraints exist in the literature, these classifications depend on the classification criteria used [14]. We will adopt the following classification:

The D_CI constraints: which include the structural constraints (namely the uniqueness of the identifier value of an entity, not the nullity of the identifier,..) and the functional constraints associated with the nature of the functionality (the age of an employee should not exceed 60 years, the stock quantity cannot be null,...)

The R_CI constraints:

The referential constraint which concern the foreign key (a field is forced to take the values from another relation field).

The inclusion constraint that determines that an association is included in another association, that said, the occurrences of the identifier of an association exists as occurrence of the dependent association.

The equality constraint which reflects a mutual inclusion between occurrences of the concerned associations. Indeed, two associations are equal if the combinations of the primary key occurrences are identical in both associations.

The exclusivity constraint: for two associations which constrained by the exclusivity, any occurrence of identifier present in an association cannot be present in the other.

The generalization constraint which results from the presence of a master entity that contains the common properties (factorization properties) and child entities that contain their own properties.

2.3.2. Conceptual Constraints

In the conceptual model, the constraint is a Boolean expression that it can be attached in any conceptual element. In general, it allow to indicate a restriction or to provide complementary information about a model. It specifies the role and scope of the conceptual element to extend or clarify its semantics.

A constraint is always attached to a conceptual element that constitute the context of the constraint. Several types of constraints can be specified, we quote:

The constraints of type that concern the type of properties, for each property, the set of values that can be attributed depends of a specific kind,

The structural constraints: these are properties of a class and the different types of relations between classes (Generalization, Aggregation, Composition, dependence and association ...),

The constraints of visibility: this is the scope of a conceptual element (property, class ...). Visibility is a mechanism of collecting the data and methods into a structure by concealing the location of bodies (public, private, protected ...),

The constraint of abstraction that constrained the class as not be instantiated and the operation to be redefined in the classes, which inherits the abstract class,

The constraints of relationships, such as: the limit of the number of involved instances (navigation expressions), the exclusivity (XOR), the invariance of the number of instances (frozen),

The constraints of cardinality which express the degree of participation of a class in a relationship by defining two scalar values for each constraint: the minimum number of instances of the classes must participate in the relationship and the maximum number of instances that can participate in the relationship.

2.4. Generation of Conceptual Model from Relational Schema

2.4.1. Description of the generation's rules

Rule 1: generation of classes: Every relation of physical schema whose primary key contains no foreign key generate a class in the conceptual model.

Rule 2: generation of properties: Every attribute of the relation becomes a property in the class generated from the relation in question.

Rule 3: generation of Associations:

Generalization constraint: Every relation whose primary key is both foreign key will be a generalization link (SubOf) between the two classes.

Many to many associations : Every relation whose primary key is made entirely of foreign keys that reference multiple primary keys, generates a class (association class) whose properties are possibly the properties of the relation which not constitute the key.

For each foreign key, an arity is generated such that the domain is the class generated from the referenced relation, the range is the class generated from the reference relation, the role is the concatenation of the domain and range names, the maximum cardinality is more (*), the minimum cardinality is 1.

One to many associations : for each relationship with a foreign key attribute that references no primary key an arity is generated. The domain of this arity is the class generated from the referenced relation. The range is the generated class from the reference relationship. The role is the concatenation of the names of the domain and range. The maximum and minimum cardinalities are 1.

In both cases, for each arity, a second arity opposite is generated such that :The domain is the class generated from the reference relation, the range is the class generated from the referenced relation, the role is the concatenation of the names of the range and domain, the maximum cardinality is more (*), the minimum cardinality is 1. Both arities generate an association.

Rule 4: generation of constraints between attributes (D_CI): For each property, it is tested whether it is primary key, foreign key, unique, can be NULL. For each test, a constraint is generated with the Boolean value True or False.

Rule 5: generation of constraints between associations (R_CI): Only inter-associations constraints will be treated. Those constraints express the nature of the participation of a class to a set of associations. These constraints are difficult to find in the physical schema because they do not affect the translation of the model to physical schema. This Rule focuses on the relation of the physical schema whose primary key is made entirely of foreign keys that reference multiple primary keys.

We define:

- listRelation: the list of relations extracted from the schema,
- listPK (Relation rel) the list of attributes which are the primary key of rel,
- upletslistValuesPK(Relation rel) the list of primary key values of relation rel.

The extraction of R_CI Constraints is based on the following algorithm:

```
List listRelation; // list of relations,
List listPK (Relation rel); // list of attributes which constitute the primary key of the relation
rel,
List upletslistValuesPK(Relation rel); // list of tuple of values of attributes which compose the
                                       //primary key of the relation rel.
ExtractR_CI(Relation rel1){
  foreach(rel2 in listRelation){
    if (rel2 ≠rel1){
   if(listPK(rel2) Include in listPK(rel1)){
     if (ALL listValues(rel1) include in listValues(rel2)){
         if (ALL listValues(rel2) include in listValues(rel1)) {
             //generation of equal constraint on both corresponding associations
                 rel1.R_CI.add(CreateEqualConstraint(rel2));
          }else{
// generation of inclusion constraint on the two corresponding associations
                 rel1.R_CI.add(CreateIncludeConstraint(rel2);
          }
     }elseif((ANY listValues(rel1) include listValues(rel2))&&(ANY listValues(rel2)
   include listValues(rel1))){
             //generation of exclusive constraint on the two corresponding associations
                 rel1.R_CI.add(CreateXORConstraint(rel2);
  }}}
}

ExtractR_CI(){
  foreach(rel in listRelation){
       ExtractR_CI(rel);
}
```

Fig. 4. The Extracting R_CI constraints algorithm.

2.4.2. Description of the generation's algorithm

The generation of the conceptual elements from the elements of the relational schema is based on the following general algorithm:

We define :

Hash listProp; : a Dictionary of properties created from foreign key's attributes. the key of the listProp is the property and the value is composed by a vector of the name of attribute relation, attribute referenced relation and a boolean which determinates the generated association type.

```
List listRelation;
List listAttributes(Relation rel);

createModel(){
        Hash listProp;
        aModel = mmconceptual.Factory.createModel();
        createClass();
       createAssociations();
       createConceptualConstraints(ExtractAssociationsConstraints());
 }
createClass(){//rule 1
      ForEach (rel in listRelation){
           aClass = mmconceptual.Factory.CreateClass();
           aClass.setName(rel.name);
           ...
          // for each attribute in the relation rel
          Foreach(att in listAttributes(rel)){
               Property aProperty = crreateProperty(att);
               if(aProperty.Constraints.Contains(RefKey)
                   {
                       if (aProperty.Constraints.Contains(Key) {
                           if (rel==referencedRealtion)
                               ExtractGeneralizationConstraint();//rule 2
                            else{
                         // property created from a foreign key attribute which is part of
                        //association's primary key
                          listProp.add(aProperty,{rel,referencedRelation,true});
                            }
                       listProp.add(aProperty,{rel,referencedRelation,false});
                   }
               else    aClass.add(aProperty);
           }
       if (all att in rel are foreign key) // aClass will not be added on the aModel
       else
           aModel.add(aClass);
       }
  }

  createProperty(Attribute att){    //rule 2
          aProperty = mmconceptual.Factory.CreateProperty();
          aProperty.CreateType();
          aProperty.CreatePropertyConstraints(att);
       }

createPropertyConstraints(Attribute att){    //rule 4
           if (isPrimaryKey(att))
               Constraints.add(mmconceptual.Factory.CreateConstraints(Key);
                if (isForeignKey(att))
                        Constraints.add(mmconceptual.Factory.CreateConstraints(RefKey);
                if (isUnique(att))
                        Constraints.add(mmconceptual.Factory.CreateConstraints(Unique);
                ...
}

createArities(Class c1,c2; int min,max){ //rule 3
            anArity = mmconceptual.Factory.CreateArity();
            anArity.SetDomain(c1); anArity.SetRange(c2);
            anArity.setMinCard(min);anArity.setMaxCard(max);
}

createAssociations(){//rule 3
             forEach(prop in listProp){
                    if (listProp(prop)[2] == true) {
                            createArities(ClassOf(listProp(prop)[0]), ClassOf(listProp(prop)[1]),1,n);
                            createArities(ClassOf(listProp(prop)[1]), ClassOf(listProp(prop)[0]),1,n);
                    else {
                            createArities(ClassOf(listProp(prop)[0]), ClassOf(listProp(prop)[1]),1,1);
                            createArities(ClassOf(listProp(prop)[1]), ClassOf(listProp(prop)[0]),1,n);
                            }
}

createConceptualConstraints(){//rule 5
forEach(assoc in associations){
    createConstraints(assoc,ExtractR_CI());
}
```

Fig. 5. The generation algorithm.

3. Implementation

RDB2MODEL is implemented as an Eclipse plug-in, it allow the extraction of schema from a relational database and the generation of a conceptual model.

To test our plug-in we use the following relational database example:

Customers		
idCust	nameCust	telCust

Product		
refP	unitPrice	intitled

Order		
NOrder	dateOrder	idCst

Products_Per_Orders		
numOrder	referenceProd	quantity

Pershable_Products	
referenceP	dateExpiration

Fig. 6. The RDB example.

With:

- The properties idCust, refP, NOrder, reference, and (numOrder, referenceProd) are primary keys,
- The properties idCst, numOrder and referenceProd are foreign keys,

- The properties unitPrice and dateExpiration cannot be NULL,
- The property nameCust is unique.

The execution of RDB2MODEL generate the following conceptual model expressed in XMI (XML Metadata Intechange)[1] :

The conceptual model expressed in XMI and a log file that are detailed as follows:

Table 1. Details of Generated Log File

RDB Table	**Generated log**
Customer	Class-gencustomer Property- Name :idCust Type : mmconceptual.impl.TypeImpl@120b476 (idType: Integer) Constraints : [mmconceptual.impl.ConstraintImpl@27e2d8 (id: 0, name: notnull)] Property- Name :nameCust Type : mmconceptual.impl.TypeImpl@9df6e3 (idType: String) Constraints : [mmconceptual.impl.ConstraintImpl@bbe4b0 (id: 0, name: unique)] Property- Name :telCust Type : mmconceptual.impl.TypeImpl@1a2fefd (idType: String) Constraints : []
Order	Class-genorders Property- Name :NOrder Type : mmconceptual.impl.TypeImpl@12a54b (idType: Integer) Constraints: [mmconceptual.impl.ConstraintImpl@220a0c (id: 0, name: key), mmconceptual.impl.ConstraintImpl@c1a427 (id: 0, name: notnull)] Property- Name :dateOrder Type : mmconceptual.impl.TypeImpl@12a54b (idType: Date) Constraints : []
Product	Class-genproduct Property- Name :refP Type : mmconceptual.impl.TypeImpl@1d69794 (idType: Integer) Constraints: [mmconceptual.impl.ConstraintImpl@1799a64 (id: 0, name: key), mmconceptual.impl.ConstraintImpl@874920 (id: 0, name: notnull)] Property- Name :intitled Type : mmconceptual.impl.TypeImpl@1d1f3f2 (idType: String) Constraints : [] Property- Name :unitPrice Type : mmconceptual.impl.TypeImpl@1d1f3f2 (idType: Float) Constraints: [mmconceptual.impl.ConstraintImpl@1736b50 (id: 0, name: notnull)]

[1] http://www.omg.org/cgi-bin/doc?formal/2000-11-02

Perishable-Products	Class-genperishableproduct Property- Name :referenceP Type : mmconceptual.impl.TypeImpl@10020bb (idType: Integer) Constraints: [mmconceptual.impl.ConstraintImpl@164ff8f (id: 0, name: notnull)] Property- Name :dateExpiration Type : mmconceptual.impl.TypeImpl@1c61c91 (idType: Date) Constraints: [mmconceptual.impl.ConstraintImpl@17d9553 (id: 0, name: notnull)]
Products_Per_Orders	Class-genproducts_per_orders Property- Name :quantity Type : mmconceptual.impl.TypeImpl@c7640b (idType: Integer) Constraints : [] Association n:n ARITY NAME : products_per_ordersorderCible Class order Source Class products_per_orders MIN CARDINALITY : 0 MAX CARDINALITY : -1 ARITY NAME : products_per_ordersproductCible Class product Source Class products_per_orders MIN CARDINALITY : 0 MAX CARDINALITY : -1 ARITY NAME : orderproducts_per_ordersCible Class products_per_orders Source Class order MIN CARDINALITY : 0 MAX CARDINALITY : -1 ARITY NAME : productproducts_per_ordersCible Class products_per_orders Source Class product MIN CARDINALITY : 0 MAX CARDINALITY : -1
	Association 1:n ARITY NAME : customersorderCible Class orders Source customers MIN CARDINALITY : 1 MAX CARDINALITY : -1 ARITY NAME : ordercustomersCible Class customers Source orders MIN CARDINALITY : 1 MAX CARDINALITY : 1

The conceptual model expressed in XMI

```
<?xml version="1.0" encoding="UTF-8"?>
<mmconceptual:Modelxmi:version="2.0"xmlns:xmi="http://www.omg.org/XMI"
xmlns:mmconceptual="http://mmconceptual/1.0">
//class gencustomers
<class name="GenCustomers">
        <has name="idCust" hasAType="//@types.0” source="//@constraints.0"/>
        <has name="nameCust" hasAType="//@types.1” source="//@constraints.7"/>
        <has name="telCust" hasAType="//@types.0" />
</class>
//class genorder
<class name="GenOrder">
        <has name="NOrder" hasAType=”//@types.0” source="//@constraints.0//@constraints.6"/>
        <has name="dateOrder" hasAType="//@types.2"/>
</class>
//class genproduct
<class name="GenProduct">
        <has name="refP"    hasAType="//@types.0" source="//@constraints.0"/>
```

```
          <has name="unitPrice"    hasAType=”//@types.3” source="//@constraints.6"/>
          <has name="intitled" hasAType="//@types.1"/>
</class>
//class genproducts_per_orders
<class name="GenProducts_Per_Orders">
          <has name="quantity" hasAType="//@types.0"/>
</class>
//class genperishable_products
<class name="GenPerishable_Products"subOf="//@class.2">
//generalisation between product and perishable_products
          <has name="referenceP" hasAType=”//@types.0” source="//@constraints.4//@constraints.6"/>
          <has name="dateExpiration" hasAType=”//@types.2" source="//@constraints.6"/>
</class>
//association between customers and orders
<associations>
             <define     minCard="1"     maxCard="-1"     name="CostumersOrder"     role="CostumerOrder"
          domain="//@class.0" range="//@class.1"/>
             <define     minCard="1"     maxCard="1"     name="OrderCostumers"     role="OrderCostumer"
          domain="//@class.1" range="//@class.0"/>
</associations>
// class association between product and order
<associations>
             <define          minCard="0"          maxCard="-1"          name="products_per_ordersorder"
          role="products_per_ordersorder" domain="//@class.3" range="//@class.1"/>
             <define          minCard="0"          maxCard="-1"          name="orderproducts_per_orders"
          role="orderproducts_per_orders" domain="//@class.1" range="//@class.3"/>
</associations>
<associations>
             <define          minCard="0"          maxCard="-1"          name="products_per_ordersproduct"
          role="products_per_ordersproduct" domain="//@class.3" range="//@class.2"/>
             <define          minCard="0"          maxCard="-1"          name="productproducts_per_orders"
          role="productproducts_per_orders" domain="//@class.2" range="//@class.3"/>
</associations>
<types id="Integer"/>
<types id="String"/>
<types id="Date"/>
<types id="Float"/>
<types id="Character"/>
<constraints id="0" type="xor"/>
<constraints id="1" type="frozen"/>
<constraints id="2" type="subset"/>
<constraints id="3" type="equal"/>
<constraints id="4" type="key"/>
<constraints id="5" type="refKey"/>
<constraints id="6" type="notnull"/>
```

```
<constraints id="7" type="unique"/>
</mmconceptual:Model>
```

Fig. 7. The instantiated model generated in XMI.

4. Conclusion

We present a generic method that transforms a relational database to a conceptual model consistent with our meta model.

The method is implemented as an Eclipse plug-in and named RDB2MODEL: After choosing a relational data base, RDB2MODEL retro design this one and extract the relational schema which will be transformed to generate a conceptual model by instantiating our conceptual meta model.

RDB2MODEL is a part of a global approach that consist to generate a conceptual model from any kind of classical data sources (Object Oriented, Semi Structured or Relational):

The generation of the conceptual model is enriched semantically be the constraints extracted from the data base. In this paper we are limited to data description level constraints and data level constraints, so, we plan to extend the study to include the data manipulation level (stored procedures, triggers, joins ...) to further enrich the generated model by the semantics

References

[1] Torrance, R., & James, D. (2011, June). The state-of-the-art in semiconductor reverse engineering. *Proceedings of the 48th Design Automation Conference*.

[2] Canfora, H. G., & Di, P. M. (2007, May). New frontiers of reverse engineering. *2007 Future of Software Engineering*.

[3] Fonkam, M. M., & Gray, W. A. (1992, May). An approach to eliciting the semantics of relational databases. *Proceedings of the International Conference on Advanced Information Systems Engineering* (pp. 463-480).

[4] Jeusfeld, M. A., & Johnen, U. A. (1994, December). An executable meta model for re-engineering of database schemas. *Proceedings of the International Conference on Conceptual Modeling* (pp. 533-547). *Springer Berlin Heidelberg*.

[5] Hainaut, J. L., Tonneau, C., Joris, M., & Chandelon, M. (1993, December). Transformation-based database reverse engineering. *Proceedings of the International Conference on Conceptual Modeling* (pp. 364-375).

[6] Andersson, M. (1994, December). Extracting an entity relationship schema from a relational database through reverse engineering. *Proceedings of the International Conference on Conceptual Modeling* (pp. 403-419).

[7] Signore, O., Loffredo, M., Gregori, M., & Cima, M. (1994, December). Reconstruction of er schema from database applications: A cognitive approach. *Proceedings of the International Conference on Conceptual Modeling* (pp. 387-402).

[8] Vermeer, M. W., & Apers, P. M. (1995, December). Reverse engineering of relational database applications. *Proceedings of the International Conference on Conceptual Modeling* (pp. 89-100).

[9] Petit, J. M., Kouloumdjian, J., Boulicaut, J. F., & Toumani, F. (1994, December). Using queries to improve database reverse engineering. *Proceedings of the International Conference on Conceptual Modeling* (pp. 369-386).

[10] Premerlani, W. J., & Blaha, M. R. (1993, May). An approach for reverse engineering of relational databases. In Reverse Engineering.

[11] Chiang, R. H., Barron, T. M., & Storey, V. C. (1994). Reverse engineering of relational databases: Extraction of an EER model from a relational database. *Data & Knowledge Engineering, 12(2), 107-142*.

[12] Demuth, B. H. (1999). Using UML/OCL constraints for relational database design. *Proceedings of the*

International Conference on the Unified Modeling Language. Springer Berlin Heidelberg

[13] Godfrey, P., Grant, J., Gryz, J. *et al*. Integrity constraints: Semantics and applications. *Logics for Databases and Information Systems.*

[14] Codd E. F. A. (1970). Relational model of data for large shared data banks. *Communications of the ACM.*

[15] OMG. Retrieved from http://www.omg.org/spec/OCL/2.4/"omg

[16] Warmer, J. B., & Kleppe, A. G. (2003). The object constraint language: Getting your models ready for MDA.

Jakjoud Widad was born in Marrakech, Morocco in 1979. She got her diploma of computer science engineer from Hassania School of Public Works (EHTP) after completing her license in computer sciences from University Cadi Ayyad of Marrakech; she is PhD student in the Department of Mathematics and computer sciences, Faculty of Sciences & Technology of Settat, University Hassan I, Settat, Morocco. In her research she is interested in the ontology, semantic web and model driven engineering.

Mohamed Bahaj was born in 1964, in Ouezzane, Morocco. He got his PhD in applied mathematics from University of Pau, France, in 1993. He is now working as a professor at the Department of Mathematics & Computer Sciences, University of Hassan 1er, Faculty of Sciences & Technology Settat, Morocco. His research interests include pattern recognition, semantic web & ontology in MAS, controls of mobiles agents.