Supporting Model-Driven Development of Object-Relational Database Schemas: a Case Study

Juan M. Vara, Belén Vela, Verónica A. Bollati, Esperanza Marcos

Kybele Research Group
Rey Juan Carlos University
Madrid (Spain)
{juanmanuel.vara, belen.vela, veronica.bollati, esperanza.marcos}@urjc.es

Abstract. This paper completes our proposal for automatic development of Object-Relational (OR) DataBase (DB) schemas. By means of a case study, this work focuses on presenting the tooling developed to support the whole process. As usual, the proposal starts from a conceptual data model (Platform Independent Model) depicted in a UML class diagram. Then, the conceptual data model is mapped into an OR DB model (Platform Specific Model) that represents the OR DB schema. To that end, we have implemented a set of formalized mapping rules using the ATL language. Finally, the SQL code that implements the modeled schema in Oracle 10*g* is automatically generated from the OR model by means of a MOFScript model to text transformation. Moreover, since the OR model could be refined along the design process, we have developed a graphical editor for OR DB models.

Keywords: Model-Driven Engineering, Object-Relational Databases, Model Transformations, Code Generation.

1 Introduction

In spite of the impact of relational Databases (DBs) over the last decades, this kind of DB has some limitations for supporting data persistence required by present day applications. Due to hardware improvements, more sophisticated applications have emerged, which can be characterized as consisting of complex objects related by complex relationships. Representing such objects and relationships in the relational model implies that the objects must be decomposed into a large number of tuples. Thus, a considerable number of joins are necessary to retrieve an object. Thus, when tables are too deeply nested, performance is significantly reduced [3]. A new DB generation appeared to solve these problems: the Object-Oriented DB generation, which includes Object–Relational (OR) DB [22]. This technology is well suited for storing and retrieving complex data. It supports complex data types and relationships, multimedia data, inheritance, etc. Moreover, it is based on a standard [10]. It extends the basic relational model with user defined types, collection types, typed tables, generalizations, etc. Nowadays, OR DBs are widely used both in the industrial and academic areas and they have been incorporated into a lot of commercial products.

Nevertheless, good technology is not enough to support these extensions. Just like it happens with relational DBs [8], design methodologies are needed for OR DB

development. Those methodologies have to incorporate the OR model and to take into account old and new problems. Like choosing the right technology, solving platform migration and platform independence problems, maintenance, etc.

Following the Model-Driven Engineering (MDE) [5,21] proposal, this paper presents a model-driven approach for Object-Relational Databases development of MIDAS [13], a methodological framework for model-driven development of service-oriented applications. Since it is a model-driven proposal, models are considered as first class entities and mappings between the different models are defined, formalized and implemented. By means of a case study, this work presents the tooling developed to support the proposal.

The proposal starts from a Platform Independent Model (PIM), the conceptual data model, which is mapped to an OR DB model that represents the OR DB schema. To that end, we have defined metamodels for OR DB modeling. Then, the Platform Specific Model (PSM) is translated into SQL code by means of a model to text transformation. This work focuses on the mappings defined to transform the conceptual data model into the OR DB schema, i.e. to obtain the PSM from the PIM, as well as on the code generation from that schema.

The rest of the paper is structured as follows: first, section 2 summarizes the proposed OR DB development process, including the metamodel for OR DB modeling, as well as a brief overview of the PIM to PSM mappings. Next, section 3 uses a case study to illustrate the tooling developed to support the proposal. Section 4 examines related work and finally, section 5 outlines the main findings of the paper and raises a number of questions for future research.

2 Object-Relational DB Development in MIDAS

Our approach for OR DB development is framed in MIDAS, a model-driven methodology for service-oriented applications development. It falls on the proposal for the content aspect, which corresponds to the traditional concept of a DB. Here we focus on the PIM and PSM levels, whose models are depicted in Fig. 1. In our approach, the proposed data PIM is the conceptual data model represented with a UML class diagram while the data PSM is the OR model or the XML Schema model, depending on the selected technology to deploy the DB.

Regarding OR technology we consider two different platforms, thus two different PSMs. The first one is based on the SQL standard [10] and the second one on a specific product, Oracle10*g* [20]. Here, we focus on the OR DB development. We have defined two metamodels for OR DB modeling, one for the SQL standard and the other one for Oracle 10*g*. The later will be presented in the next section since the one for Oracle is still valid for SQL just by considering some minor differences. Moreover, using the one for Oracle we can check the output by loading the generated code into an Oracle DB.

In the proposed development process once the conceptual data model (PIM) is defined, an ATL model transformation generates the OR DB model (PSM). If needed, the designer could refine or modify this model using the graphical editor developed to that end. Finally, a MOFScript model to text transformation takes the previous OR

DB model as input and generates the SQL code for Oracle that implements the modeled DB schema. We will present all these mappings in the following sections.

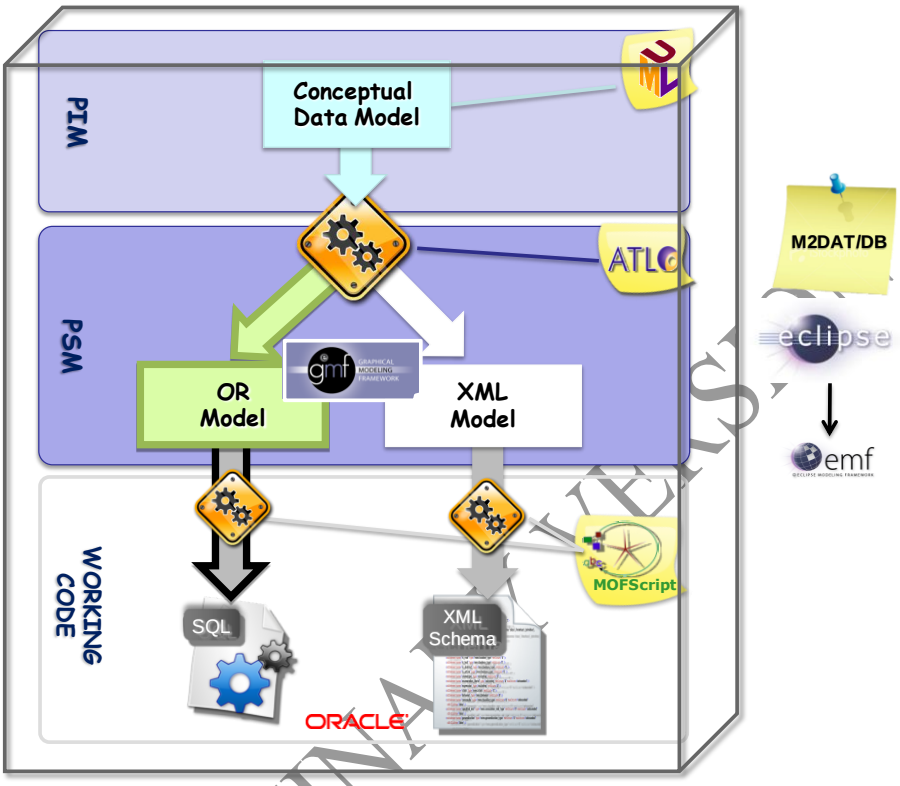


Fig. 1. MIDAS technical solution for ORDB Development: M2DAT/DB

All this work is been integrated in M2DAT (MIDAS MDA Tool) as a new module: M2DAT/DB. M2DAT is a case tool whose aim is to support all the techniques proposed in MIDAS for the semiautomatic generation of the information system. The modular nature of the MIDAS architecture facilitates the modular development of M2DAT. Therefore, we are able to address the development of separate modules that implement the different proposals of MIDAS and later integrate them by means of a model transformation bus. The tool is now under development in our research group and its early functionalities have been already presented [24]. Fig. 1 shows the architecture of M2DAT/DB.

2.1 Modeling OR DB

In previous works we described a pair of UML profiles for OR DB modeling. They were defined according to the earlier specifications of the SQL standard and Oracle OR capabilities [12].

However, when we first addressed the task of implementing the PIM to PSM mappings of our proposal, we decided to shift from UML profiles to Domain Specific

Languages (DSL) [14]. Technology is playing a key role in the distinction between UML based and non-UML based tools: the facilities provided in the context of the *Eclipse Modeling Project* (EMP) and other DSL frameworks, like the *Generic Modeling Environment* (GME) or the *DSL Tools*, have shifted the focus from UML-based approaches to MOF-based ones [5]. Therefore, regarding existing technology for (meta-) modeling and model transformations, it was more convenient to express the new concepts related with OR DB modeling by means of well defined metamodels. To that end we have defined a pair of MOF based metamodels. Here we will summarize only the one for Oracle10*g* since the one for the SQL standard is very similar.

Figure 2 shows the metamodel that represents the object characteristics of Oracle10*g*'s OR model. The main differences with regard to the standard metamodel are: Oracle neither supports the ROW type nor the inheritance of tables and Oracle does support the NESTED TABLE type instead of the MULTISET type (although they represent nearly the same concept).

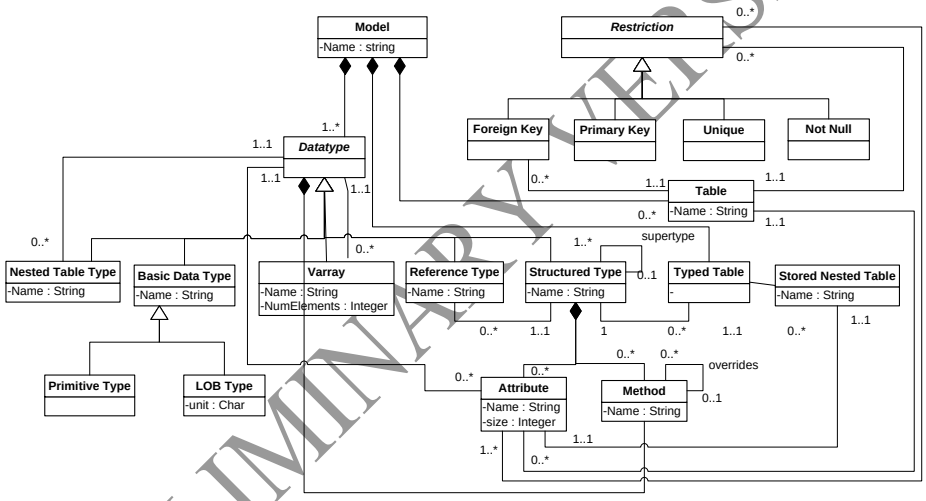


Fig. 2. Oracle 10g's OR metamodel

2.2 PIM to PSM Transformations for OR DB Development

We have to define a set of mapping rules to move from the conceptual data model (PIM) to the OR DB one (PSM). Regarding how they should be defined, in [19] it is stated that "the mapping description may be in natural language, an algorithm in an action language, or a model in a mapping language". This way, in previous works [7] we sketched a common approach to address the development of model to model transformations in MIDAS framework:

- First, the mappings between models are defined using natural language.
- Next, those mappings are structured by collecting them in a set of rules, expressed again in natural language.
- Then, these mapping rules are formalized using graph grammars.

- Finally, the resulting graph transformation rules are implemented using one of the existing model transformation approaches (we have chosen the ATL language; this decision will be explained later).

This proposal is oriented to give solution to some problems we have detected in the field of model transformations: there is a gap between the developers of the different model transformation engines and those who have to use them. We aim at reducing this gap by providing with a simple method for the definition of mappings. In this sense, the fact that models are well represented as graphs is particularly appealing to shorten the distance between modelers and model transformation developers. Moreover, formalizing the mappings before implementing them can be used to detect errors and inconsistencies in the early stages of development and can help to increase the quality of the built models as well as the subsequent code generated from them. These activities are especially important in proposals aligned with MDA (like the one of this paper), because MDA proposes the models to be used as a mechanism to carry out the whole software development process. Likewise the formalization of mappings has simplified significantly the development of tools supporting any model-driven approach.

We have followed this method to carry out the PIM to PSM mappings of our proposal for OR DB development. As a first step, we collected the mappings in the set of rules summed up in Table 1.

It is worth mentioning that these rules are the result of a continuous refining process. As mentioned before, the very first version of these mappings was conceived for the SQL:1999 standard and the 8*i* version of Oracle [12].

Table 1. OR PIM to PSM Mappings

Data PIM		**Standard Data PSM (SQL:2003)**	**Product Data PSM (Oracle10*g*)**
Class		Structured Type + Typed Table	Object Type + Object Table
Class Extension		Typed Table	Table of Object Type
Attributes	Multivalued	Array/Multiset	Varray/Nested Table
	Composed	ROW/Structured Type (column)	Object Type (column)
	Calculated	Trigger/Method	Trigger/Method
Association	One-To-One	Ref/Ref	Ref/Ref
	One-To-Many	Ref/Multiset/Array	Ref/Nested Table/Varray
	Many-To-Many	Multiset/Multiset Array/Array	Nested Table/Nested Table Varray/Varray
	Aggregation	Multiset/Array	Nested Table/Varray of References
	Composition	Multiset/Array	Nested Table/Varray of Objects
	Generalization	Types/Typed Tables	Types/Typed Tables

The next step was the formalization of the mapping rules using a graph transformation approach [2]. Finally, those rules are implemented using the ATL Language [11], a model transformation language developed by ATLAS Group. It is mainly based on the OCL standard and it supports both the declarative and imperative approach, although the declarative one is the recommended.

We have chosen ATL because, nowadays, it is considered as a de-facto standard for model transformation since the OMG's Query/View/Transformations (QVT)

practical usage is called into question due to its complexity and the lack of a complete implementation. There do exist partial implementations, both of QVT-Relational, like ikv++'s medini QVT, and QVT Operational Mappings, like SmartQVT or Eclipse's QVTo. However, none of them combines both approaches (declarative and imperative), in theory, one of the strengths of QVT, and they are still to be adopted by the MDD community. On the other hand, ATL provides with a wide set of tools and its engine is being constantly improved. Next section presents some of the mapping rules next to their implementation using the ATL language.

3 Case Study

This section presents, step by step, our proposal for model-driven development of OR DB by means of a case study: the development of an OR DB for the management of the information related to the projects of an architect's office.

It should be noticed that, once the PIM has been defined, the whole process is carried out in an automatic way. Nevertheless, the designer has the option to refine and/or modify the OR generated model before the code generation step. To ease this task, we have developed a graphical editor for models conforming to the OR metamodel sketched in section 2.1.

3.1 Conceptual Data Model

The first step in the proposed development process is to define the conceptual data model. Fig. 3 shows the model for our case study: a project manager is related to one or more projects. Both, his *cod_manager* or his name serve to identify him. He has an *address* plus several phone numbers. In its turn, every project has a name and it groups a set of plans, each one containing a set of figures. We would like to know how many figures a plan contains. Thus, a *number figures* derived attribute is included in the Figure class. Those figures could be polygons, which are composed of lines. Every line has an identifier and contains a set of points. The length of the line is obtained by computing the distance between their points.

We have used UML2 to define this model. UML2 is an EMF-based implementation of the UML metamodel for the Eclipse platform. We use UML2 for the graphical definition of PIM models in M2DAT.

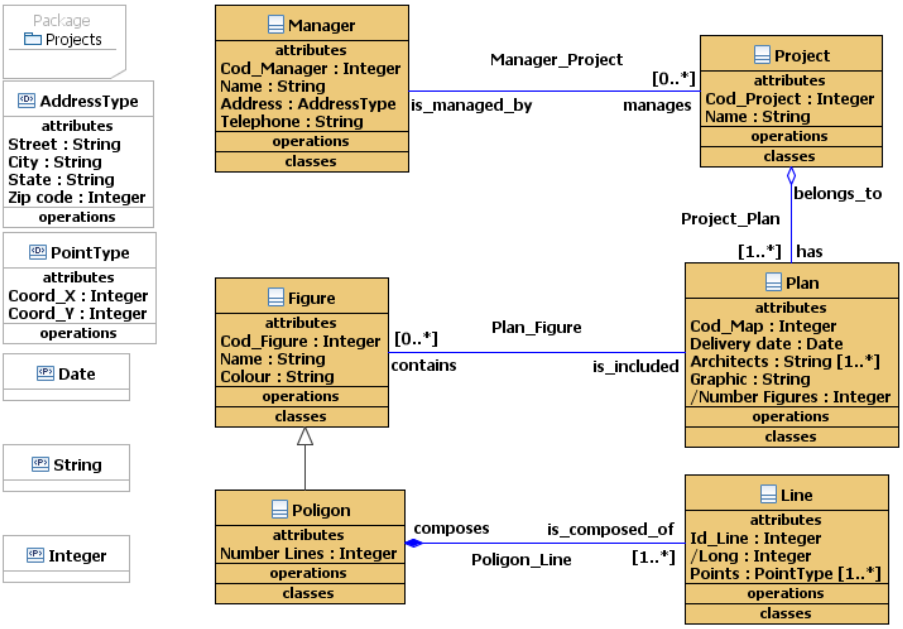
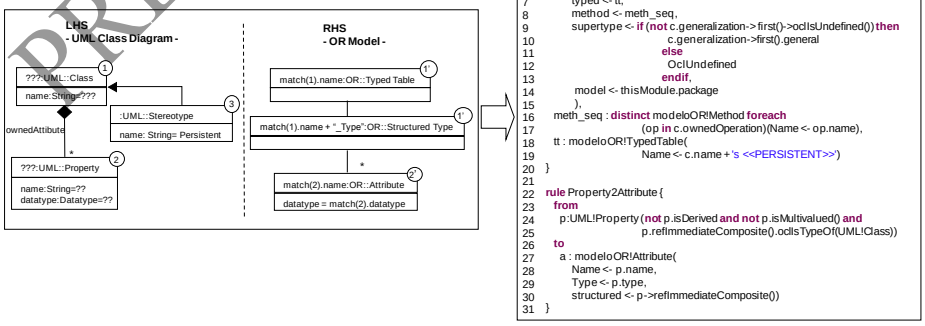


Fig. 3. Conceptual Data Model for the case study

3.2 PIM to PSM Mapping

Now we will show how some of the mapping rules are applied to obtain the OR PSM for Oracle 10*g*. We present the formalization of each rule with graph grammars next to its ATL implementation.

Mapping Classes and Properties. Left-hand side of Fig. 4 shows the graph transformation rule to map persistent classes (PIM) to DB schema objects (PSM).




```
1   rule Class2UDT {
2   from
3     c : UML!Class
4   to
5     udt : modeloOR!StructuredType(
6         Name <- c.name + ' <<UDT>>',
7         typed <- tt,
8         method <- meth_seq,
9         supertype <- if (not c.generalization->first()->oclIsUndefined()) then
10                        c.generalization->first().general
11                      else
12                        OclUndefined
13                      endif,
14        model <- thisModule.package
15        ),
16    meth_seq : distinct modeloOR!Method foreach
17                      (op in c.ownedOperation)(Name <- op.name),
18    tt : modeloOR!TypedTable(
19                      Name <- c.name + 's <<PERSISTENT>>')
20  }
21
22  rule Property2Attribute {
23   from
24     p:UML!Property (not p.isDerived and not p.isMultivalued() and
25                      p.refImmediateComposite().oclIsTypeOf(UML!Class))
26   to
27     a : modeloOR!Attribute(
28         Name <- p.name,
29         Type <- p.type,
30         structured <- p->refImmediateComposite())
31  }
```

Fig. 4. Persistent Classes mapping rule

Whenever an UML class stereotyped as persistent is found on the PIM (①−③), a structured type (also known as user-defined type - UDT) and a typed table are added on the PSM (①'). The type of the new table will be the UDT. Each property associated with the persistent class will be mapped by adding an attribute to the UDT (②⇨②').

Right-hand side of Fig. 4 shows the *Class2UDT* and *Property2Attribute* ATL rules that implement the prior graph transformation rule. The *Class2UDT* rule states that for every Class found in the source model (`UML!Class`), an UDT and a typed table are created in the target model (`modeloOR!StructuredType` and `modeloOR!TypedTable`). A set of direct bindings initialize the simple attributes of the structured type, such as the *name*. While more complex bindings serve to initialize the rest. For instance, the expression used to initialize the *supertype* attribute first checks if the source class was the descendant of any other class (lines 9-13).

On the other hand, the *Property2Attribute* ATL rule maps every UML property of each class in the source model to an attribute of the corresponding UDT. To that end, the *structured* property of each new attribute is bound to the source class that contains the property that is being mapped (line 30). When the ATL engine evaluates this expression, the reference to the source class is replaced by a reference to the structured type that maps the source class. This way, we do not need to navigate the target model when we need to establish references between its elements. It is the ATL engine that deals with this task using its resolve mechanism [11]: whenever a reference to an element in the source model is found, it is replaced by a reference to the element that maps it in the target model.

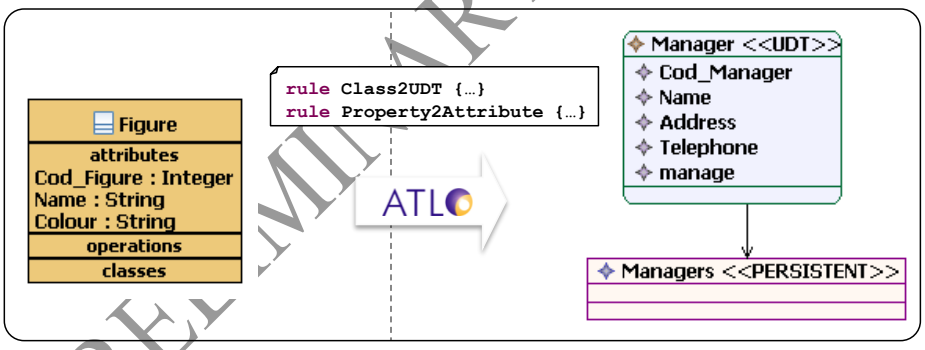


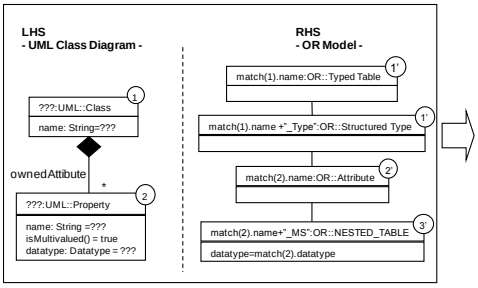
Fig. 5. Mapping of class *Manager*

Fig. 5 shows the instantiation of this rule. On the left-hand side there is an extract of the conceptual data model: the *Manager* UML class owning a set of properties. The right-hand side is a partial screen capture from our graphical editor. It shows the elements that map the source class into the target model: an object type named *Manager* owning a set of attributes, next to the corresponding *Managers* typed table.

Notice that we have opted for giving a UML-alike appearance to the graphical editor. This way, we try to take the best from UML: it is well-known by software engineers and developers. However, we try to take out the worst from UML: its complexity. When you are developing model transformations for UML stereotyped models you have to deal with the complex and big metamodel of UML − profiles

always add something, they never remove anything. We prefer to avoid this complexity by using a DSL with "UML graphical syntax".

Mapping Multivalued Properties. The mapping rules we have just presented work fine for mapping classes and their properties in the generic case. However, as showed on Table 1, specific mapping rules are needed to handle the special nature of certain properties. This way, we have defined rules for mapping *multivalued*, *composed* and *calculated* (i.e. derived) properties. For the sake of space we present here just the first one (see Fig. 6).




```
1   rule PropertyMultivalued2Attribute {
2   from
3     p :UML!Property (not p.isDerived and p.isMultivalued() and
4                      p.refImmediateComposite().oclIsTypeOf(UML!Class))
5   to
6    a : modeloOR!Attribute(
7          Name <- p.name,
8          Type <- thisModule.generateNestedTableMultivalued(p),
9                      structured <-(p.refImmediateComposite())
10         ),
11    st : modeloOR!StoredNestedTable(
12         Name<- p.name + 's_ List',
13         attribute <- a,
14         typed <-thisModule.resolveTemp(
15                     p->refImmediateComposite(),'tt')
16       )
17  }
18  unique lazy rule generateNestedTableMultivalued {
19   from
20     p: UML!Property
21   to
22     nt : modeloOR!NestedTableType (
23          Name <-'NT_' + p.name + 's <<NT>>',
24          Type <- p.type,
25          model <- thisModule.package)
26  }
```

Fig. 6. Multivalued properties mapping rule

Multivalued properties on the conceptual data model correspond to columns of a collection type on OR DB schemas. Oracle 10*g* provides with two pre-defined constructors for collection types: VARRAY and NESTED TABLE. Fig. 6 shows the graph transformation rule when we choose the NESTED TABLE constructor.

The UML class (①) has a multivalued property (②). It is stated by the *true* value of its *isMultivalued* attribute. Therefore, the structured type that maps the persistent class (①') owns an attribute of a NESTED TABLE type (②'–②') (the same is valid for a VARRAY type).

When we coded this rule we did not know of any way to decide whether we want to create a VARRAY or a NESTED TABLE each time the rule is executed. Therefore, we chose NESTED TABLE as the default collection type since it is the less restrictive. Right now we have solved this problem using annotation models to parameterize the transformation. Right-hand side of Fig. 6 shows that we need two different rules to code the mapping of multivalued properties. The guard of the *Property2MultiValuedAttribute* rule ensures that only multivalued properties of UML classes will match this rule (line 3-4). Its target pattern specifies that two elements will be added in the target model to map the property found on the source model: an OR attribute and an OR Nested Table (lines 11-25). Moreover, by the time the new attribute is created, a Nested Table type is also defined. To that end, the *type* property of the new attribute is initialized by calling the *generateNestedTableMultivalued* lazy rule (line 8-9). A lazy rule is also a declarative rule but it has to be explicitly called. This way, the type of the attribute is the newly created Nested Table type.

Fig. 7 shows the result of mapping the multivalued property *Architects* of the class *Plan*.

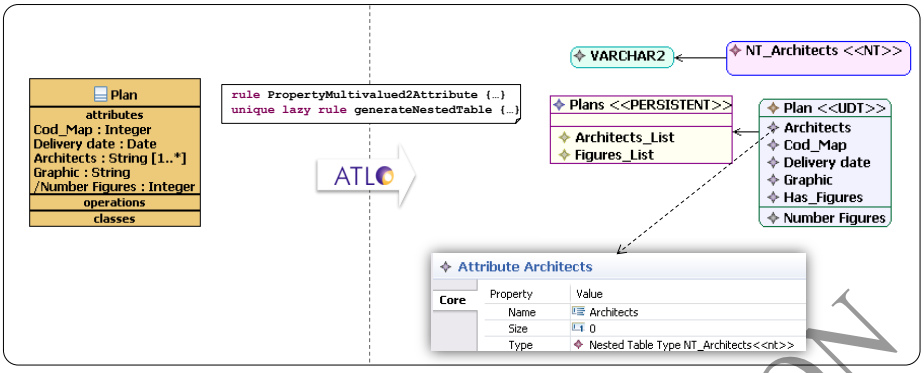


Fig. 7. Mapping of *Architects* multivalued property

Mapping Associations. We propose different transformation rules depending on the maximum multiplicity of the classes involved in the association (see Table 1). For the sake of simplicity, we have considered only uni-directional relationships. The same rules can be applied for bi-directional relationships with two minor adjustments: duplicating the construction in both sides of the rule and adding some mechanism to maintain referential integrity (a trigger for instance). Here we introduce the rule for mapping many-to-many associations.

As left-hand side of Fig. 8 shows, *Maximum Many Multiplicity* associations are identified in the PIM by the value of the *upper* attribute in the UML property (②) of the source class. The corresponding uni-directional relationship in the PSM is built upon an attribute in the Structured Type that maps the source class of the association (①'–②'). This attribute is a collection of references to the Structured Type that maps the target class (②'–②'). Thus, it is a Nested Table object containing a set of references (REF type objects).

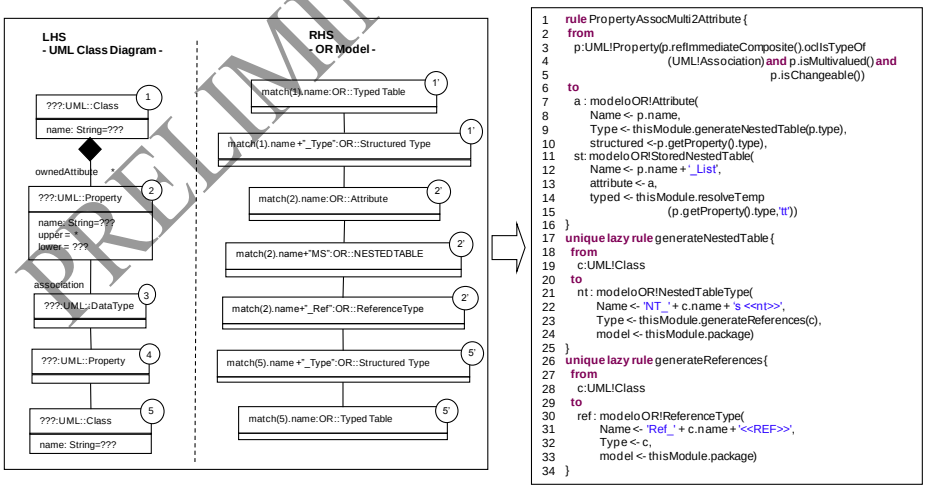


Fig. 8. Associations mapping rule

Right-hand side of the picture shows the corresponding ATL implementation. The guard ensures that only UML multivalued properties of an UML association will be

mapped by this rule (line 3-5). To map the property, an OR attribute is created (lines 6-16). The type of the attribute will be a Nested Table type. It is created by the *generateNestedTable* lazy rule (lines 17-25). Also, we need to create a new REF type by invoking the *generateReferences* lazy rule (lines 26-33). It serves to define the type of the elements of the Nested Table type. Notice that both lazy rules are *unique*. This way, we ensure that neither the REF, nor the Nested Table types will be duplicated if those rules are called again with the same parameters. Instead, they will return a reference to the already created types.

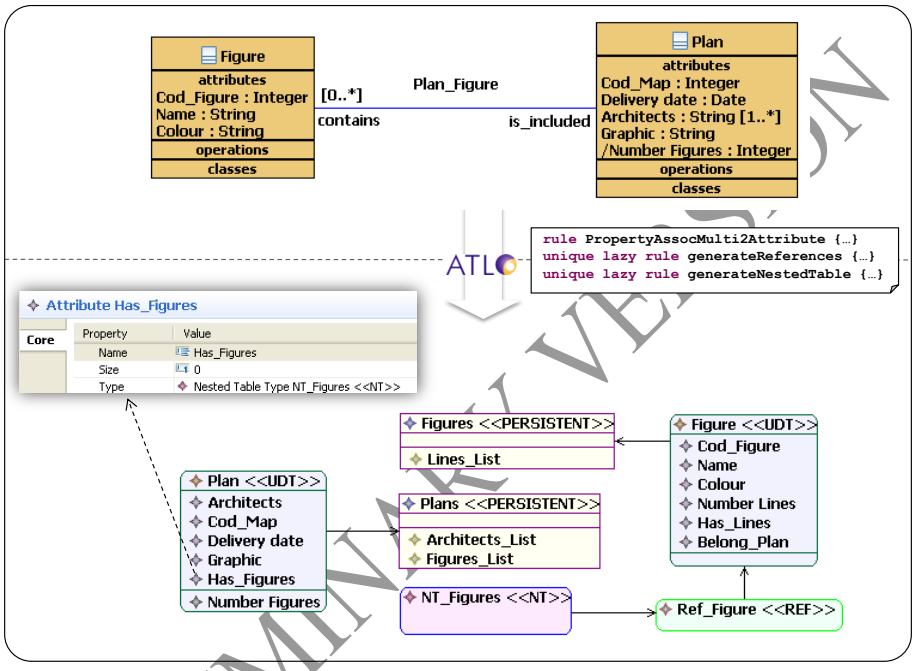


Fig. 9. Mapping of *Has_Figures* association

Fig. 9 shows the result of mapping the association between a *Plan* and its *Figures*: apart from the structured types and the typed tables that map the *Plan* and the *Figure* classes, a REF type (*Ref_Figure*) and a Nested Table type are created (*NT_Figures*). The later contains a collection of objects of the former.

3.3 Code Generation

In order to implement the generation of the corresponding SQL code from the OR model, we have chosen the MOFScript language [18]. MOFScript is a prototype implementation based on concepts submitted to the OMG MOF Model to Text RFP process.

In front of the declarative approach of ATL (and the vast majority of existing model to model proposals), model to text transformation engines take the form of imperative programming languages. In fact, a MOFScript script is a parser for models conforming to a given metamodel. While it parses the model structure, it generates a

text model based on transformation rules. On a second phase this text model is serialized into the desired code. This way, the script uses the metamodel to drive the navigation through the source model, just as an XML Schema drives the validation of an XML file. As a matter of fact, every model is persisted in XMI format, an XML syntax for representing UML-like (or MOF) models.

The program that implements the model to text transformation is basically a model parser. It navigates the structure of the OR model, generating a formatted output stream: the SQL script that implements the modeled DB schema in Oracle 10*g*. In the following we introduce this script showing some code excerpts. The reader is referred to [18] for more information on how to configure MOFScript execution.

As showed below, a *main* function (line 13) is the entry point for the script. It includes a set of rules for processing each possible type of element that can be found in the source model (so-called context types in MOFScript). Besides, we include the *eco* parameter in the script header to specify which the input metamodel is (line 11). To that end we use the URI that identifies the OR metamodel we have presented in section 2.1.

```
11 texttransformation codigo (in eco:"http:///modeloOR.ecore") {
12
13   eco.Model::main(){
```

```
18     //Structured Type code generation
19     self.datatype->forEach(s:eco.StructuredType)
20     {
21           s.generateStructured()
22     }
```

Next, a transformation rule is defined for each context type. For simple rules, we code the rule inside the *main* body whilst the complex ones are coded by means of auxiliary functions. Those functions are invoked from the *main* body.

For instance, the rule for Structured Types creation is probably the most complex one since it encapsulates a lot of semantics. Thus, it is coded in the *generateStructured* auxiliary function. The *main* body invokes it for every Structure Type object found in the source model (line 21 below).

The next code extract presents the *generateStructured* function (for the sake of space, we have skipped the code for attributes mapping, lines 100-127).

```
88   // Auxiliary Function for Structured Type code generation
89  eco.StructuredType::generateStructured() {
90     var texto:String=""
91     if (self.supertype.Name.size()=0)
92        texto="CREATE OR REPLACE TYPE " + self.Name.replace(" <<UDT>>", "") + " AS OBJECT \r\n("
93     else
94        texto="CREATE OR REPLACE TYPE " + self.Name.replace(" <<UDT>>", "") + " UNDER " +
95              self.supertype.Name.replace(" <<UDT>>", "") + "\r\n("
96     println(texto)
97
98     // Attributes
99      self.attribute->forEach(a:eco.Attribute) {
      .....
128     // Typed Tables
129     self.typed->forEach(t:eco.TypedTable){
130       println("CREATE TABLE " + t.Name.replace(" <<PERSISTENT>>","") +
                              " OF " + self.Name.replace(" <<UDT>>","") + "\r\n(")
131       t.generateTypedTable()
132        println("")
133     }
134   }
```

First, the auxiliary variable that will store the SQL code is initialized (line 90). Next, we add the SQL code to start the creation of the structured type, distinguishing those types that inherit from any other type (lines 91-96) from those that do not. Then, another rule is coded to transform attribute objects (lines 100-127). This rule is executed for every attribute object that belongs to the structured type (line 99). Finally, we iterate over the *typed* property of the structured type to identify the typed tables, whose type is the one being mapped (line 129). Then, the SQL code to start the creation of each table is added and the rule to create them, the *generateTypedTable* function, is invoked (lines 130-131).

To conclude this section, Fig. 10 shows a piece of the SQL code generated for the case study.

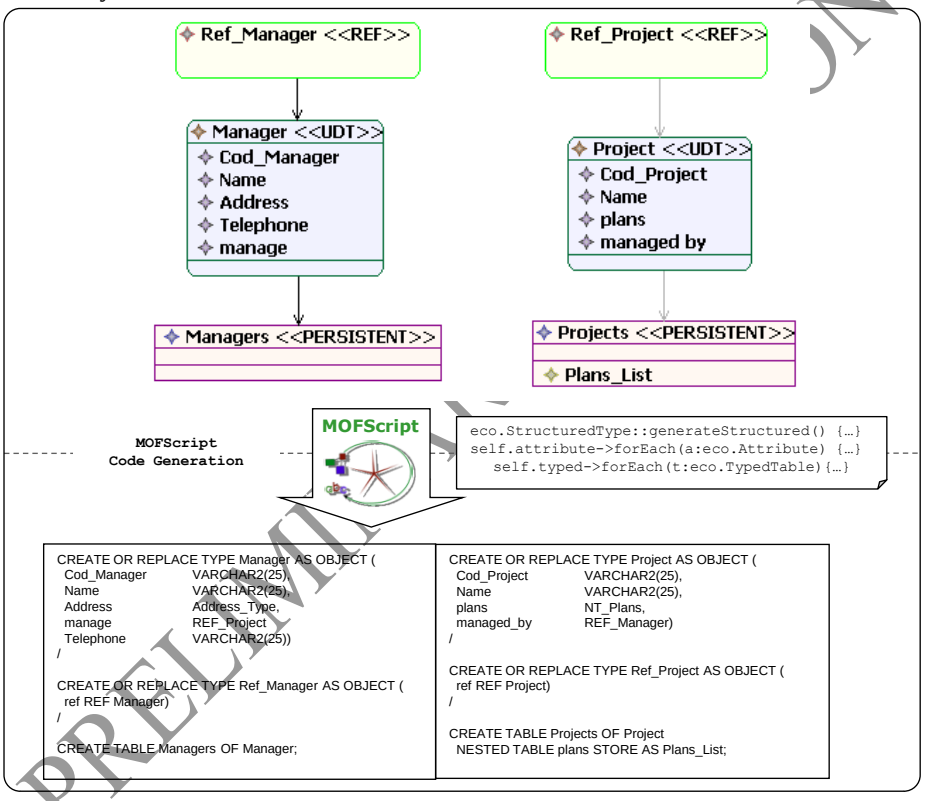


Fig. 10. Code Generation excerpt

The upper side is a screen capture of the developed graphical editor. It shows an extract from the OR model of the case study. Specifically the *Manager* and *Project* types and the corresponding typed tables, next to the REF types created from them. This code is generated by the execution of the mapping rules listed in the annotation beside.

4 Related Work

The mapping of object models to relational DB schemas and vice versa has been widely used as a case study to present new model transformations proposals. For instance, see the works from [6].

Besides there are some works on the application of model-driven techniques to Data Warehouses (DW) development though they are still too immature, limiting themselves to the proposal of new UML profiles for DW modeling. Probably the most mature are the ones from Mazón et al. [9, 15] and Tonkunaite et al. [23] where a modeling process is proposed and the corresponding model transformations are already defined. Nevertheless, they are just formally specified with the QVT standard but not implemented using any technical solution for model transformation.

Likewise, we should consider the works from Atzeni, Bernstein et al. on model management [1]. They propose a framework focused on schema mappings. The proposal is based on a solid formal basis (relational DBs) but some objections can be made from the point of view of MDE. First, the use of a new metametamodel (known-as Supermodel) different from MOF is not justified. It makes it hard to develop bridges towards the universe of MOF-compliant proposals. Moreover, such Supermodel is extended when needed, raising other questions. Where is the limit to extend the Supermodel? Why is it not extended for each new metamodel considered? Using an evolving metametamodel seems to be a bad practice. One can argue that, each time the Supermodel is extended, you might consider updating your existing metamodels to consider the modifications or just to improve them. But there is no doubt this is a bad practice. Evolving metamodels are a hard problem in MDE. Whenever you modify your metamodel, you will have to propagate those changes to your model transformations, code generator, diagrammers, etc. i.e. to all your tooling.

However, to the best of our knowledge there are no previous works applying model-driven techniques for OR DB development. As a matter of fact, this is the main contribution of this work with regard to other existing approaches for OR DB development, like the ones from [16] and [17]. On the one hand, the creation of the models that take into account technological aspects are postponed as much as possible in the development process, so that technology changes imply low costs. On the other hand, the definition of model to model and model to text transformations results in the automatic generation of the most part of the working-code. In this sense, our proposal is original and complete. We have defined a modeling process and we have developed the technical support to carry out the proposed process.

5 Conclusions and Future Work

In this paper we have used a case study to present the tool support of our proposal for model-driven development of OR DB schemas. To that end, we have implemented an ATL model transformation that generates an OR DB model from a conceptual data model and a MOFScript model to text transformation that generates the SQL code for

the modeled DB schema. As part of the proposal we have defined a MOF-based DSL for OR DB modeling as well as a graphical editor for such DSL.

The implementation of our proposal is one of the modules of M2DAT (MIDAS MDA Tool). M2DAT is a case tool, which integrates all the techniques proposed in MIDAS for the semi-automatic generation of service-oriented Information Systems. We are still working on the development of the technical support for the rest of the methodological proposals comprised in MIDAS framework. In this sense, this work has served also as test bench. We have obtained a set of lessons learned and good practices to address such challenges. Once we complete the modules that rest, we will address the integration task.

Besides, this paper allows confirming one of the hypotheses around model transformation sketched in [7]: PIM to PSM transformations are easier to automate than PIM to PIM transformations. The former implies decreasing the abstraction level, thus handling more specific artifacts that result easier to model. The later implies a higher abstraction level and more similarity between the elements of the source and target models, in short, more ambiguities. Thus, it requires a higher level of decision making from the designer.

Regarding this issue there is still some place for improvement. At present time we are working in the use of weaving models [3] to annotate the OR model (i.e. the source model). This way, we are able to *parameterize* the model transformation, so that the mapping process for OR DB development can be customized. For each specific case some design decisions can be made. This way we keep the generic nature of the model transformation without polluting the source model.

Acknowledgments

This research has been carried out in the framework of the MODEL-CAOS (TIN2008-03582/TIN) and CONSOLIDER (CSD2007-0022, INGENIO 2010) projects financed by the Spanish Ministry of Education and Science (TIN2005-00010/) and the M-DOS project (URJC-CM-2007-CET-1607) cofinanced by the Rey Juan Carlos University and the Regional Government of Madrid.

References

1. Atzeni, P., Cappellari, P., and Gianforme, G. 2007. MIDST: model independent schema and data translation. In *Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data* (Beijing, China, June 11 - 14, 2007). SIGMOD '07. ACM, New York, NY, 1134-1136.
2. Baresi, L. and Heckel, R. 2002. Tutorial Introduction to Graph Transformation: A Software Engineering Perspective. In Proceedings of the First international Conference on Graph Transformation). LNCS 2505. Springer-Verlag, London, pp. 402-429, 2002.
3. Bernstein, P A., *Applying Model Management to Classical Meta Data Problems*. First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA. 2003.
4. Bertino, E. and Marcos, E. Object Oriented Database Systems. In: Advanced Databases: Technology and Design, O. Díaz and M. Piattini (Eds.). Artech House, 2000
5. Bézivin, J. Some Lessons Learnt in the Building of a Model Engineering Platform. 4th Workshop in Software Model Engineering (WISME), Montego Bay, Jamaica (2005)
6. Bézivin, J. et al. Model Transformations in Practice Workshop. Proceedings of the Satellite events of the 8th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2005. Montego Bay, Jamaica, LNCS 3844, Springer-Verlag, 2005.

7. Cáceres, P., De Castro, V., Vara, J.M. and Marcos, E. Model Transformations for Hypertext Modeling on Web Information Systems. In: SAC '06. Proc. of the 2006 ACM Symposium on Applied Computing. ACM Press, New York, pp. 1232-1239, 2006.
8. Chen, P.P. The Entity-Relationship Model – Toward a Unified View of Data. ACM Transactions on Database Systems, Vol. 1, No. 1. March 1976, pp. 9-36, 1976.
9. E. Fernández-Medina, J. Trujillo, R. Villarroel, M. Piattini, Developing secure data warehouses with a UML extension, Information Systems, 32 (2007) pp.826–856.
10. ISO / IEC 9075-14:2008 *Standard, Information Technology – Database Languages – SQL:2008*, International Organization for Standardization, 2008.
11. Jouault, F. and Kurtev, I. Transforming Models with ATL. In: Proc. of the Model Transformations in Practice Workshop. MoDELS Conference, Jamaica. 2005.
12. Marcos, E., Vela, B. and Cavero, J.M. Extending UML for Object-Relational Database Design. Fourth Int. Conference on the Unified Modeling Language, UML 2001. Toronto (Canadá), LNCS 2185, Springer-Verlag, pp. 225-239. October, 2001.
13. Marcos, E. Vela, B., Cáceres, P. and Cavero, J.M. MIDAS/DB: a Methodological Framework for Web Database Design. DASWIS 2001. Yokohama (Japan), November, 2001. LNCS 2465, Springer-Verlag, pp. 227-238, September, 2002.
14. Marjan, M., Jan, H., Anthony, M.S.: When and how to develop domain-specific languages. ACM Comput. Surv. 37 (2005) 316-344.
15. Mazon, J.-N., Trujillo, J., Serrano, M., Piattini, M.: Applying MDA to the development of data warehouses. Proceedings of the 8th ACM international workshop on Datawarehousing and OLAP. ACM, Bremen, Germany, 2005.
16. Muller, R. Database Design for Smarties. Morgan Kaufmann, 1999.
17. E.J. Naiburg and R.A. Maksimchuk, UML for Database Design. Addison-Wesley, 2001.
18. Oldevik, J., Neple, T., Grønmo, R., Aagedal, J., Berre, A.-J.: Toward Standardised Model to Text Transformations. Model-driven Architecture – Foundations and Applications, pp. 239-253, 2005.
19. OMG. MDA Guide Version 1.0. Document number omg/2003-05-01. Ed.: Miller, J. and Mukerji, J. Retrieved from: http://www.omg.com/mda, 2003.
20. Oracle Corporation. Oracle Database 10g. Release 2 (10.2). In: www.oracle.com.
21. Selic, B. The pragmatics of Model-Driven development, IEEE Software, Vol. 20, 5, Sept.-Oct. 2003, pp.19-25, 2003.
22. Stonebraker, M. and Brown, P.,Object-Relational DBMSs. Tracking the Next Great Wave. Morgan Kauffman, 1999.
23. Tonkunaite, J., Nemuraite, L., Paradauskas, B.: Model driven development of data warehouses. Databases and Information Systems, 2006. 7th International Baltic Conference on. 106-113 (3-6 July 2006).
24. Vara, J.M., De Castro, V. and Marcos, E. WSDL automatic generation from UML models in a MDA framework. In International Journal of Web Services Practices. Volume 1 – Issue 1 & 2. November 2005, pp.1 – 12. 2005.
25. Voelter, M. *MD* Best Practices*. Published in: http://voelter.de, 12/2008. Retrieved January, 10, 2009.