CASE Tool for Object-Relational Database Designs

Thiago Rais de Castro
Petrobras Company
Rio de Janeiro, RJ, Brazil
thiagoraiscastro@gmail.com

Solange N. Alves de Souza
Dept of Computing Eng. and Digital System
University of São Paulo, SP, USP
São Paulo, Brazil
ssouza@usp.br

Luiz Sergio de Souza
Faculty of Technology (FATEC)
Carapicuíba, SP, Brazil
desouza.luizsergio@gmail.com

Abstract—**In 1999, the SQL standard version presented new features to manipulate objects in relational database which has since been called Object-Relational Database (ORDB). Nowadays, many Object-Relational Database Management Systems (ORDBMS) offer resources to manipulate object in database. However, for these resources to become really utilized in corporate environment, it is necessary, among other things, to have CASE tools to aid in object-relational database design. An extension made in ArgoUML tool that permits developers to build graphics schemas to ORDBs is presented. These schemas can be generated using an ORDB profile presented here in. This profile is an extension of the UML class diagram and contains elements to represent the new resources to manipulate object in databases. The tool maps the graphic schema to SQL:2003 code and SQL code to Oracle dialect.**

Keywords-component; Object-Relational Database; SQL:2003; UML class diagram

I. INTRODUCTION

Nowadays, there are many CASE (Computer-Aided Software Engineering) tools to aid generation and maintenance of relational databases. Erwin, DBDesigner, DB-Main and others are some of the examples. Such tools offer resources to create graphic models and have an option for the automatic generation of SQL (Structured Query Language) code for some Database Management Systems (DBMS) (eg. Oracle, PostgreSQL, etc.). It is possible to divide tools into two categories: the first supports generation Entity-Relationship Model (ERM) [8] and the second supports generation of table models. In the first case, the developer creates a conceptual model using the tool which offers resource for mapping from that model to the table, in the other words, the tool generates a logical model through generation of the appropriated SQL code to the DBMS chosen. In the latter, the developer creates a model of table, that is, a graphic logical model, which is translated into the appropriated SQL code for the DBMS chosen. In both cases, the tool generates the SQL code automatically. This is important because the SQL code written by the developers can contain more errors and depends on the developers' knowledge. The scenario gets worse for cases in which code generation for more DBMS ones is necessary.

The use of a CASE tool for the generation of database schemas increases developers' productivity , since it frees them from writing an SQL code. In addition, database maintenance can be facilitated with the visualization of databases schemas at conceptual or logical levels, since it allows understanding database objects more easily and faster.

This article introduces an extension made in ArgoUML CASE tool to allow the generation of graphic logical models for Object-Relational Databases (ORBDs). The tool has elements that allow developers to create a graphic model using new available object resources in SQL:2003 specification [1,2,5,9]. Thus, the impedance mismatch problem that occurs when the object orientation (OO) paradigm and relational databases are used may be removed. The Graphic Logical Model built by the developer is an extension of the UML (*Unified Modeling language*) class diagram. The automatic translation from the graphic logical schema to the SQL code is made by the tool. The extension in the ArgoUML was made by adding two modules that permit the automatic generation of code in SQL:2003 specification and Oracle 11g.

This article is organized as follows. Section II introduces the architecture for ORDB design that shows all the technologies involved in our proposal. Section III shows an ORDB profile to support modeling graphic logical schemas. Section IV discusses some related works and highlights differences between previous works and the ones introduced here. Section V describes the CASE tool architecture. Section VI introduces the CASE tool result. Finally, Section VII concludes the paper with a proposal for future work.

II. TECHNOLOGIES AND MODELS

The use of the UML class diagram for conceptual modeling is an alternative to MER [6] which allows using a single model to represent persistent and transient objects. Hence, it not only eliminates the effort to make a specific model to represent persistent objects, but also to check the consistence between models generated. In addition, the UML class diagram has elements to represent objects, different kinds of relationships and intrinsic constraints of OO. As well as being an appropriate choice, the extension of UML also decreases the

effort made to obtain a model representing the exclusive elements for ORDB such as UDTs, typed tables and others. The extension was made by creating an UML profile called ORDB profile. The extension made in UML allows building Object-Relational Logical Model and with ORDB CASE tool developers can build graphic logical models defining elements supported by ORDB.

The models and metamodels involved in the generation of Object-Relational Logical Model are shown in Figure 1. Abstraction levels connected with databases design are used to represent the models and specifications corresponding to the conceptual and internal levels of the ANSI/SPARC architecture. It is possible to highlight the parallel between the ANSI/SPARC architecture and the MDA (Model Driven Architecture). In addition, at the logical design level, the intersection between PIM (Platform-Independent Model) and PSM (Platform-Specific Model) may be noticed. The PIM is independent of specific technology, thus the developer only has to be concerned about aspects related to business domain. On the other hand, PSM considers some type of technology, thus the physical level is connected to PSM. The logical design level illustrated in Figure 1 is not related to any ORDBMS, although it is connected with the physical design. However, SQL:2003 is a specification of a specific technology: ORDB. Figure 1 illustrates the logical design level as being the intersection between PIM and PSM, since it is difficult to discern which is better.

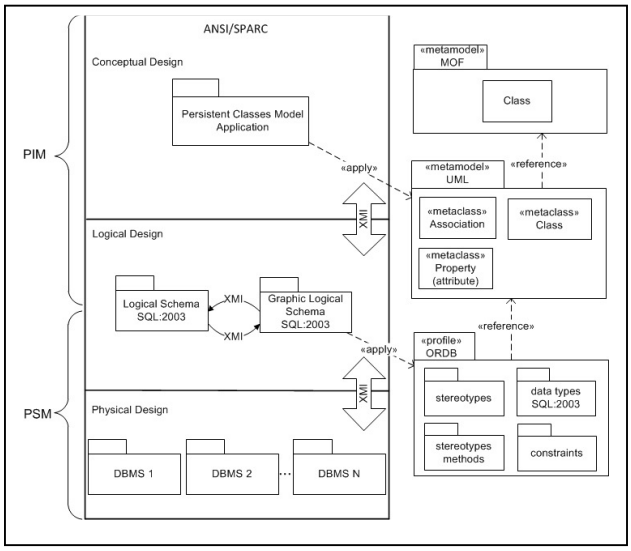

Figure 1. Metamodels and models involved in the generation of Object-Relational Graphic Logical Model

The mapping among all levels of abstraction is made by using XMI (XML Metadata Interchange) [7], Figure 1 shows the conceptual schema as well logical ones which may be easily transported to other CASE tools, in case it is necessary, since XMI facilitates interoperability.

A. *Graphic Logical Model*

The Graphic Logical Model utilized to develop graphic schemas of users is constituted by elements defined in ORDB profile and in the UML (right side of Figure 1). For logical separation of elements, the profile was organized into four subpackages: stereotypes, stereotype methods, constraints and data types SQL:2003. Table I introduces the elements belonging to stereotypes, stereotype methods and constraints subpackages. The first column in Table I indicates the name of stereotype, the second describes the UML element in which the stereotype is applied and the third describes the meaning of the stereotype.

TABLE I. STEREOTYPES OF GRAPHIC LOGICAL MODEL

Stereotype	*Basic Element*	*Description*
«table»	Class	Defines a table [6].
«udt»	Class	Defines a User Defined Type: can be composed by one or more attributes, to have methods and to generate supertype-subtype hierarchy
«typed table»	Class	Defines a typed table: a table defined as from an UDT.
«row type»	Class	Defines a row type: structure composed by one or more fields.
«define»	Association	Defines a typed table as of an UDT.
«udt»	Class	Defines a User Defined Type: can be composed of one or more attributes, to have methods and to generate supertype-subtype hierarchy
«static»	Method	Defines a static method of an UDT.
«constructor»	Method	Defines an UDT constructor method.
«instance»	Method	Defines an instance method, if any stereotype is specified the instance method is used.
«overriding»	Method	Utilized to overwrite the inherited method.
«pk», «fk», «unique», «check» and «not null»	attribute	Defines stereotypes corresponding to PRIMARY KEY, REFERENCES, UNIQUE, CHECK and NOT NULL constraints, respectively.

The SQL:2003 offers resources for method definition like program application. Hence, in the logical model proposed, the static, instance or constructor methods can be represented by appropriated stereotypes, according to Table I. In addition, the polymorphism is made with keyword overriding. The notations included in the profile proposed and shown in Table I allow an easy visualization, better understanding of the schema and automatic code generation for a specific schema.

Tables II and III present a mapping proposal from an UML class model (conceptual model) to OR logical model.

TABLE II. MAPPING FROM CLASS MODEL TO OR LOGICAL MODEL - RELATIONSHIP

Association		Correspondent in a ORDB Logical Schema	
Bidirectional association	Composition aggregation association	1..1	A cross reference is defined, i.e., each class keeps a reference (REF) to another one
		1..* [a]	A cross reference is also defined, but the aggregator class will have an *Array* or a *Multiset* of references
Unidirectional Association		The same as in bi-directional association, but only one table will have the reference	
N-ary Association (three or more classes)		A table or an UDT is defined with the association name.	
Associative Class		A table or an UDT is defined with the name of associative class.	
Generalization, Specialization		An UDT is defined for each class in the inheritance.	

a. use Array if multiplicity is known, if not, use Multiset.

TABLE III. MAPPING FROM CLASS MODEL TO OR LOGICAL MODEL – CLASS, ATTRIBUTE AND METHOD

Class Diagram	ORDB Logical Schema	Justify
Class	Table	The logical model should be flexible. This way, the developer should be free to choose how to represent each class according to the business characteristics.
	UDT	
	Typed Table	
	Row Type	
Abstract class	UDT	An abstract class cannot be instantiated, although it can be used to define a concrete class (a class which can be instantiated). It is thus possible to represent these cases defining a UDT without a typed table associated (data only can be persisted in a typed table). Despite this, the UDT will still be used to define other UDTs.
Simple attribute	*Build-in*	It is possible to find types such as real, integer, character, etc in SQL.
Composite Attribute	Row Type	When defining methods connected to the structure it is not intended, composite attribute should be mapped in row type.
Multivalued attribute	Array or Multiset	Multidimensional structures are appropriated to keep the same type of attributes, i.e, which represent collections.
Methods	UDT's Methods	Methods are defined in UDTs. If a class method must be implemented by SGBD, an UDT must be defined with the appropriated method.

III. RELATED WORK

It is possible to classify previous works into three categories: 1. one that uses the UML class diagram to build a conceptual model which represents persistent objects [1,3,4,10]; 2. one that focuses on efforts in the definition of the mapping from conceptual objects to database objects using the new ORDB resources [3,4,1]; 3. one that extends the UML class diagram to represent new ORDB elements, making possible to design a graphic logical model [9]. Despite all these efforts, most of the application developers just map classes to relational tables, failing to exploit the strength of O-R model [2]. The lack of tools that support developers in the use of new ORDB resources contributes to this situation.

Models used and produced by a tool to represent the particularities of a specific business need to be flexible, easy to understand and to use. Thus, the model proposed and used by the tool presented here in has differences from the models proposed by [3,4,1,9]. From our point of view, these differences are important in practical cases. These differences are highlighted as follows.

a) «udt»: Differently from the proposals by [4,9], in which UDT is used only in field domains, here the UDT defined in the logic schema can be utilized for specifying field domains and typed tables (according to SQL:2003). In this way, the UDT can be reused to define the other typed table.

b) «Object Type»: this stereotype is not defined in the model proposed. Some authors [4,9] proposed «Object Type» stereotype to represent the UDT and the typed table which originated from this UDT. In this case, UDT and typed table being represented by a single notation, it is not possible to reuse the UDT. On the other hand, an UDT can be used to generate one or more typed tables (if necessary to create typed table having the same structure). Thus, in this work, an association having «define» stereotype must be used to link an existing UDT with a typed table.

c) «Knows»: this stereotype is not defined in the model proposed. According to [4,9], if there is an association between two classes (A and B) in a conceptual model, two associations «Knows» would be included in a graphic logical model, one from A to B and another from B to A. This makes the diagram heavy visually and difficult to understand. Differently and according to the UML class diagram, a single association line is utilized to represent the relationship in the graphic logic model. The CASE tool here proposed does the translation from the graphic logical model to the SQL code. In this process, the single association line is represented by appropriated attributes (according to rules defined in the tool) in associated classes.

d) «REF» «Array» «row»: in [4,9] if there is an association between two classes (A and B) in the conceptual model, two attributes must be included in each class in the graphic logical model, each one is associated with «REF», «Array» or «row» stereotype depending on the multiplicity of association. This approach adds more graphic complexity from our point of view. Furthermore, an association in the graphic model can be implicitly represented by the line of relationship and its multiplicity. Attributes would be used to represent line and multiplicity only in SQL code. Therefore, these stereotypes are not included in the model introduced here.

e) «row type»: [4,9] propose «row» stereotype applied to attributes. Different «row type» stereotype is applied to class. This allows reusing it, so the same row type can be used to define the domain of attributes in different classes.

f) «overriding»: it is proposed to represent polymorphism, similarly to SQL:2003. Differently, [4,9] proposed «redef»

stereotype.«def»: was proposed by [4,9] to represente an abstract method. However, the SQL:2003 specification does not include the concept of abstract method, which is hence not included in our model.

Besides the stereotypes shown above, others are introduced in our model and they are presented as follows.

g) «typed table»: It is utilized to represent a typed table.

h) «define»: it is utilized to associate a typed table with an UDT.

i) «static», «constructor», «instance»: in SQL:2003 different types of method are defined, the UML notation is thus proposed.

IV. EXTENSION MADE TO ARGOUML ARCHITECTURE

The ArgoUML architecture is illustrated, in general lines, in the upper part of Figure 2. An ArgoUML module is a collection of classes and resources of files (such as SQL:2003 profile which is a XMI file) that can be enabled or not. The tool offers resources to extend it from the addition of new modules (that is, Java classes must be extended). These modules are independent among them and have well defined scopes (for example, a specific module to Java, another to .Net, another to SQL, etc).

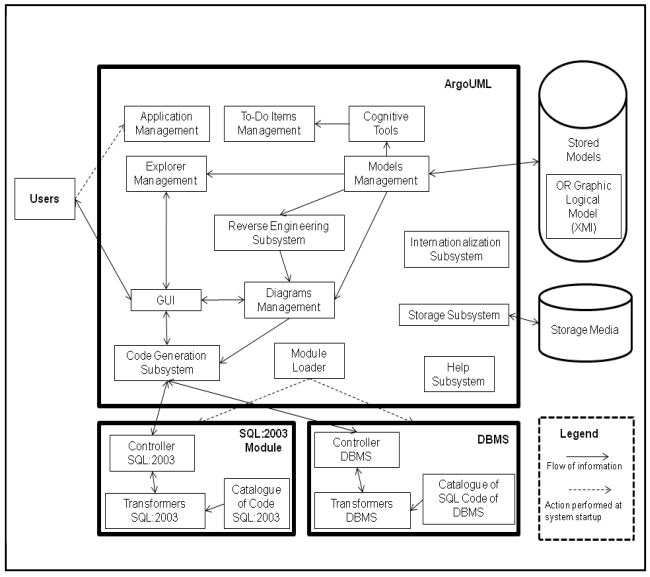

Figure 2. Architecture of the CASE tool proposed

The tool may be extended using these resources in a relatively simple way and without changing the code of the tool. As soon as the new installed module is recognized by the ArgoUML modules, the load system will receive calls which will be answered through specific actions such as generating code, exhibiting ORDB profile, etc.

According to previous arguments, the choice was to extend the tool using its own resources because, in the other option, it would be necessary to spend extra effort in tests, verification and validation which were not the main focus of this work. Figure 2 shows the architecture of ArgoUML CASE tool with the extension made to generate the logic model for ORDB.

In Figure 2, the modules built to generate the logic model for the SQL:2003 specification and for an DBMS (example: PostGreSQL, Oracle, BD2, SQL Server, etc.) are introduced. The graphic model created will be translated by the tool to the SQL dialect connected with the DBMS chosen. As new DBMSs have to be supported by the tool, new modules will be added. Modules built (Figure 2) and ArgoUML components linked to them are detailed as follows.

- Controllers: are Java classes available to ArgoUML; they are used to run module specific actions such as generating SQL code for the selected element on GUI, generating an SQL code to all the schema elements, etc. The controllers work as an interface of the SQL:2003 module or the specific DBMS (e.g. Oracle 11g, DB2, Postgre SQL, etc).
- Transformers: are components of the SQL:2003 module (or DBMS) that implement the IStrategyCodeGenerator interface for each element of the ORDB profile. These components operate the translation from graphic logical model to SQL:2003 code or to SQL code for the DBMS chosen. Transformers use OR Graphic Logical Model and the catalogue of SQL code (from the SQL:2003 or the DBMS chosen) to do translation users' schemas.
- SQL Code Catalogue: contains the code of the SQL:2003 or the DBMS chosen.
- OR Graphic Logical Model: the model made with elements defined in ORDB profile and stored in XMI. This profile is accessed via modules developed through a Models Management component.
- Models Management: manages models maintained by the system.
- Code Generation Subsystem: supports an interface for the SQL modules.
- Module Loader: offers resources to load (and to unload) auxiliary modules.

A. *Mapping from the Graphic Logic Model to SQL Code*

In general, there are differences among the SQL code generated by different ORDBMS (e.g. PostgreSQL, SQLServer, etc) and these codes can differ from the SQL specification, too; moreover, an element defined in the ORDB profile may not be supported by ORDBMS. On the other hand, the SQL specification supports all the elements defined in the ORDB profile, so, to translate from the Graphic Logical Model into the SQL code, SQL:2003 was chosen for building the model. Oracle 11g was chosen due to its support of almost all the elements defined by the SQL pattern. Furthermore, it has been employed as an academic and as a corporate project having both small and large sizes.

From ORDB stereotypes, it was possible to map profiles to their respective codes in SQL:2003 and Oracle 11g. In some cases, there is no straight mapping in Oracle 11g, for these

cases, the CASE tool will show messages that will suggest the action will be made. It is important to highlight the decision must be taken by the developer.

V. MAIN ORDB MODULE FUNCTIONALTIES

The ArgoUML, through the GUI component (Figure 2), offers menus, flaps and panels to other subsystems, external models and users. The modules developed are connected with the tool using GUI.

Figure 3 illustrates the main window of ArgoUML. In this figure, there are a menu bar (number 1), a tool bar (number 2) and four main panels: Explorer (number 3), Edition (number 4), Task (number 5) and Detail (number 6).

ArgoUML offers resources to build UML class diagram. The modules developed, introduced in section 4, extend these resources with new elements through ORDB profile (section II).

The graphic logical schemas are made in the Edition panel (number 4 of Figure 3). For this, the ORDB profile may be accessed using flaps from the Detail panel or by the Edition panel itself. The developer can take short cuts of the tool bar (number 2 of Figure 3) to build new diagrams, to save the current project and others.

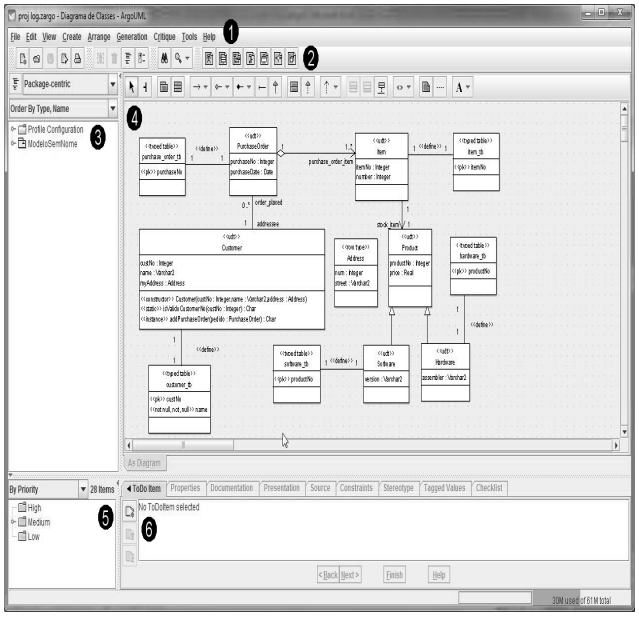

Figure 3. Main Window of ArgoUML.

The Explorer panel (number 3, Figure 3), enlarged in Figure 4, details the elements of the ORDB profile. It is important to notice that these elements are organized into packages, as discussed in section III.

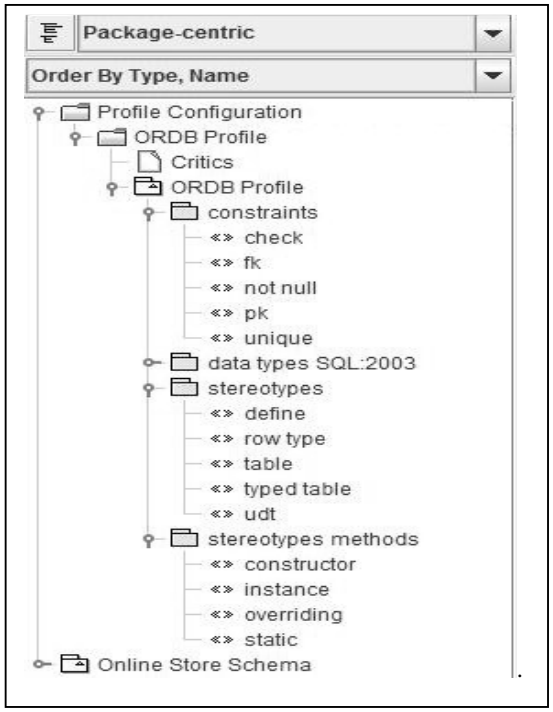

Figure 4. Example of a figure caption. (figure caption)

The Properties flap (number 6, Figure 3) is defined to all the elements of the logic schema and it permits to create/update/delete attributes, operations, relationships and others.

The language, that is, the SQL dialect (SQL:2003 or Oracle 11g), can be selected in the Source flap (number 6, Figure 3). Script generation to export schema is made in this option, too. Finally, the selection of objects for to SQL code generation is made in the menu bar (number 1, Figure 3).

The extension made in ArgoUML allows the developer to create the graphic logical schema for an application (Figure 3, number 4). Then, the SQL code for specific dialect (SQL:2003 or Oracle 11g) can be generated by the tool according to the options made by the developer.

VI. CONCLUSIONS AND FUTURE WORK

An extension to ArgoUML CASE tool to generate logical models to ORDBMS is proposed. The objective is to increase the use of ORDB. Experience has shown that the existence of tools that support technologies broadens their use.

The new modules added to ArgoUML use the Graphic Logical Model to generate, respectively, SQL code in Oracle 11g dialect and SQL:2003 dialect from graphic logical schema.

The tool introduced shows that our proposal is viable. However, to use it in real projects, it is necessary to make improvements such as new modules connecting to other SGBDs that offer support to objects besides changes in the tool interface to better represent the new resources and the

automatic generation of logical schemas to ORDB as from a conceptual model (UML classes diagram) (Figure 1).

The mapping of class diagrams to generate logical schemas has been investigated; it is necessary to consider that one class diagram element may be represented by different elements in the logical model. For example, the mapping of a class can originate a UDT, or a row type, or a table, or a typed table (see table III).

References

[1] G. Feuerlicht, J. Pokorný, K. Richta, "Object-Relational Database Design: Can Your Application Benefit from SQL:2003?", Galway, Ireland: Springer, p. 1-13. 2009.

[2] M. Fotache, C. Strîmbei, "Object-Relational Databases: An Area with Some Theoretical Promises and Few Practical Achievements", Communications of the IBIMA, v. 9, ISSN 1943-7765, 2009.

[3] Golobisky, M. F.; Vecchietti, A. Mapping UML Class Diagrams into Object-Relational Schemas. Proceedings of Argentine Symposium on Software Engineering, Rosario, Argentina, p. 65-79, 2005.

[4] Marcos, E. and Vela, B. and Cavero, J. M. A Methodological Approach for Object-Relational Database Design using UML. Heidelberg/Springer Berlin: Software and Systems Modeling, v. 2, n. 1, p. 59-72, 2003.

[5] J. Melton, "Database languages SQL: Part 1 Framework (SQL/Framework)". ISO-ANSI WD 9075, ISO, Working Group WG3, 2003G.

[6] B. S. Navathe, R. Elmasri, Sistemas de Banco de Dados. São Paulo. Pearson Education.

[7] OMG. MOF 2.0/XMI Mapping - Version 2.1.1. Disponível em: < http://www.omg.org/cgi-bin/doc?formal/07-12-02> 2007.

[8] A. Silberschatz, H. Korth, S. Sudarshan, Sistemas de Banco de Dados; Campus, 1a edição. 2006.

[9] J. M. Vara, B. Vela, J. M. Cavero, E. Marcos, "Model Transformation for Object-Relational Database Development". ACM symposium on Applied computing, New York, NY, p. 1012-1019, 2007.

[10] M. Wang, "Using UML for Object-Relational Database Systems Development: A Framework", Issues in Information Systems, vol. IX, no. 2, pp.538-543, 2008.