*Journal of Universal Computer Science, vol. 17, no. 13 (2011), 1863-1883*
*submitted: 2/3/11, accepted: 29/8/11, appeared: 1/9/11* © *J.UCS*

# A Relational Approach to Model Transformation with QVT Relations Supporting Model Synchronization

**Kun Ma**
(Shandong Provincial Key Laboratory of Network Based Intelligent Computing
University of Jinan, Jinan, China
nic-makun@ujn.edu.cn)

**Bo Yang, Zhenxiang Chen**
(Shandong Provincial Key Laboratory of Network Based Intelligent Computing
University of Jinan, Jinan, China
{yangbo, czx }@ujn.edu.cn)

**Ajith Abraham**
(Machine Intelligence Research Labs, Scientific Network for Innovation and Research Excellence, Auburn, USA
ajith.abraham@ieee.org)

**Abstract:** With the help of model transformation, it is possible to generate target models from source models. A possible way to face iterative development process with frequent modifications is to use not only a single transformation but also frequent model synchronization. In this paper, we propose a relational approach to model transformation using Query/View/Transformations (QVT) Relations language that also provides model synchronization mechanism based on the version of the models. The proposed framework uses a Platform-Independent Business Model (*PIM-BM*) and a Platform-Specific Business Component Model (*PSM-BC*) via the extension of the UML metamodel and MOF at different levels of abstraction, which sufficiently describe both the structural and behavioral properties of generic Web applications. Also we present the typical model mapping rules between *PIM-BM*s and *PSM-BC*s using QVT Relations. Finally the model synchronization based on the version of models is provided for the above model transformation approach.

**Keywords:** Model Transformation, Modeling, Model Synchronization, Model Driven Software Development, Model-Driven Architecture
**Categories:** D.2.1, D.2.2, D.2.11, I.6.5

## 1 Introduction

Model-Driven Software Development (MDSD) is gaining increasing acceptance, mainly because it can raise the level of abstraction and automation in software construction as described in [Sánchez, Moreira, Fuentes, and Magno 10]. Model transformation is a focused area in the context of MDSD, object code or Platform Specific Model (PSM) can be converted through a series of abstract Platform Independent Model (PIM) as described in [Miller, Mukerji 2003]. In this paper, we focus on the key problem of MDSD: how to define PIM and PSM, and how to

transform PIM into PSM. That is the precondition of code generation from PSM to target codes in the context of MDSD.

In addition, the development of a software system is an iterative process with frequent modifications to the involved models as described in [Subramanyam, Weisstein and Krishnan 10]. As a consequence, not only a single transformation but also frequent model synchronization steps are required.

The remainder of the paper is organized as follows. Section 2 discusses the related work. In Section 3 and 4, the structure and metamodel of platform-independent business model (hereinafter *PIM-BM*) and platform-specific business component model (hereinafter *PSM-BC*) are introduced respectively. Section 5 discuses the relational model transformation approach using Query/View/Transformations (QVT) Relations language supporting model synchronization based on the version of the models. The paper closes with some final conclusions and an outlook on future work in Section 6.

## 2 Related Work

### 2.1 Modeling language

It was argued that Unified Modeling Language (UML) is a de facto standard for modeling vocabularies as described in [Bissell 03]. In the context of MDSD, there are at least three main ways to define a modeling language as described in [Frankel (03)]. UML can represent both the static structure and behavior of the Management Information System (MIS) [see Kim, Choi, Kang, Lee 10].

First, UML is extended via Profiles. The architects of UML made a fundamental decision not to try to make UML all things to all people. Instead they equipped it with built-in extension mechanisms. A set of extensions essentially constitutes a dialect of UML, which is officially called a *profile*. A UML profile is a definition of a set of *stereotypes* and *tagged values* that extend elements of the metamodel of UML as described in [Object Management Group 10]. The main advantage of the profile approach is that a modeler who wishes to use extensions defined by a profile can do so with generic UML tools. The main disadvantage of the profile approach is that it restricts the architect of the extension from using the full semantic power of object-oriented class modeling that MOF offers. An example of a secure mobile grid system through a UML extension is given in [Rosado, Fernández-Medina, López, Piattini 10].

Second, UML is extended via Meta-Object Facility (MOF) [see Object Management Group 06]. UML can be modeled via MOF since the metamodel of UML is defined via MOF. UML extensions that use the full power of MOF are sometimes called *heavy-weight extensions*. As we have seen, MOF offers metamodelers most of the familiar UML class-modeling constructs. Creators of heavyweight extensions are free to use rich set of modeling mechanisms of MOF. MOF tools can use the greater semantic depth to intelligently manage the new kind of metadata. But taking advantage of the greater semantic expression usually makes it impossible to use the extensions when modeling with generic UML tools.

Third, a new Modeling Language is created based on the syntax and semantics of MOF. You can use other languages, as long as you supply a MOF metamodel for each of the languages. When creating a MOF metamodel to define the abstract syntax of

such modeling constructs, it often does not make sense to try to extend the UML metamodel.

Combined with the extension of UML Profiles and MOF, a novel PIM-BM and PSM-BC at different levels of abstraction are proposed in [Section 3] and [Section 4] respectively.

### 2.2 Model transformation

The Model-Driven Architecture (MDA), initiated by Object Management Group (OMG), starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform [see Miller and Mukerji 03]. Specifying a system as a set of platform independent models and transforming them into various platform specific implementation models is one of the fundamental themes of MDA. Therefore, model transformations are touted to play a key role in MDA. A transformation may be considered from two different points of view as described in [Wahler 04]. From the viewpoint of function, a transformation is a function that maps a set of models from one or more domains onto another set of models in the same or different domains; from the viewpoint of operation, a transformation is a terminating algorithm that applies structural and/or semantic changes to a model or a set of models.

Among the various model transformation techniques, relational approaches seem to be promising for various reasons. Relations offer a declarative way of specifying transformations. As mentioned before, a relation that is used for transformation purposes is specified by using a set comprehension predicate $P$, e.g., in $R = \{(s, t) | P(s, t)\}$. First-order logic of predicate is usually used to describe the relation $R$ clearly. Several publications [see Akehurst and Kent 02] [see Czarnecki and Helsen 06] apply the concept of relations to model transformation. However, the above approaches do not involve the model synchronization.

### 2.3 QVT Specification

| Name | Concept |
|---|---|
| *Query* | A query is an expression that is evaluated over a model. The result of a query is one or more instances of types defined in the source model, or defined by the query language. |
| *View* | A view is a model which is completely derived from another model. |
| *Transformation* | A model transformation is a process of automatic generation of a target model from a source model, according to a transformation definition. |

*Table 1: Basic concepts in QVT Specification*

Query/Views/Transformation (QVT) is the Object Management Group (OMG) standard language for specifying model transformations in the context of MDA in order to eliminate the heterogeneous of model transformation as described in [Object Management Group 09]. The three concepts *Query, View,* and *Transformation* have been given in the QVT Specification in Table1. In the QVT Relations language, a

transformation between models is specified as a set of relations. However, available implementations for the operational part of QVT do not support model transformations for synchronization purposes [see Nolte (09)].

It is regarded as one of the most important standards since model transformations are proposed as major operations for manipulating models. The languages *Relations* and *Core* are declarative languages at two different levels of abstraction. The specification document defines their concrete textual syntax and abstract syntax. The Relations language supports complex object pattern matching, and implicitly creates trace classes and their instances to record what occurred during a transformation execution as described in [Kurtev 08]. Relations can assert that other relations also hold between particular model elements matched by their patterns. Therefore, *Relations* language is a better choice to present the mapping rules of relational model transformation.

The relational model transformation approach from PIM-BM to PSM-BC is proposed in [Section 5]. The mapping rules of model transformation are described in the QVT Relations language.

### 2.4 Model Synchronization

In addition, a possible way to face iterative development process with frequent modifications is to use not only a single transformation where a source model is transformed into a target model by applying a set of transformation rules, but also by using frequent model synchronization. The mapping between models established by the transformation may be required to be preserved over time.

In fact, there are two different cases to clearly distinguish as described in [Hwan, Kim, Czarnecki 05]. On the one hand we can have a batch-oriented full model synchronization, which takes a source model as input and computes the resulting target model using a classical batch-oriented model transformation. On the other hand we can have model synchronization which synchronizes two models by propagating modifications. Some models that can be preserved are preserved. This basic feature updating existing target models based on changes in the source models is also referred to as *change propagation* in the Query/View/Transformation (QVT) final adopted specification [see Object Management Group 09]. A change impact analysis determines the total set of source models that need subsequent transformation based on the list of source models that were changed. This technique is about minimizing the amount of source models involved in model synchronization.

In addition, a practical approach should not replace a model by a new transformation result but rather reuse an already available model as much as possible and preserve extensions and refinements in the model wherever possible. For example, PSM is transformed into codes, and some additional codes would be added by developers. When the PSM is changed, the full model transformation would only produce the additional codes by PSM. So the additional codes are lost. A resolution for the model inconsistency problems in [Rumbaugh 04] is the mixture of models and source codes. [Efftinge, Friese, Köhnlein 08] also states some crucial basic recommendations to separate the generated and manual code from each other. However, the generic isolation is difficult to define. Combined with above resolutions, this paper proposes an effective way to resolve this problem is to embed manual codes in models rather than rewrite codes after model transformation.

There are some resolutions for model synchronization. [Hearnden, Lawley, Raymond 06a] extended a declarative rule-based live transformation engine in order to incrementally synchronize a target model with source model changes. In live update, changes to the source models or the transformation itself can then be directly mapped to their effects on transformation execution. This solution comes at the cost of a permanently maintained transformation execution context. For large transformations further optimizations of the extra needed space for the execution context have to be considered. [Giese and Wagner 09] presented an incremental model synchronization, which employs the visual, formal, and bidirectional transformation technique of triple graph. They focused on the efficient execution of the transformation rules and present their approach to achieve an incremental model transformation for synchronization purposes. But it is not very clear that it is suitable for larger numbers of changes in the case of multiple changes. [Madari, Lengyel 09] presented an approach that uses trace data structures and model transformations to facilitate incremental model synchronization. The idea of defining mappings between the elements comes from the theory of Triple Graph Grammars (TGG). The limitation of the approach is that the developers cannot modify the source and the target models simultaneously.

# 3 PIM-BM

*PIM-BM* is proposed as a platform-independent integrated business model focused on the business entity. Compared with the UML, *PIM-BM* has rich semantics, which is easy for the modelers to understand. *PIM-BM*, based on the extension of UML Profiles and MOF, removes UML elements that are not closely related to the modeling of information system.

## 3.1 Metamodel of PIM-BM

The metamodel of *PIM-BM* is shown in Figure 1. Systems are modeled as hierarchical collections of the metaclass *Entity*, *Attribute, PK, FK*, *EntityAction*, *EntityOperation* and so on.

### 3.1.1 Business Entity

Business entity is the core of business model. From the viewpoint of the applications, business entity is an integrated model with unique identifier and certain life cycle. From the viewpoint of the users, business entity is the integrated model representing static business data submitted and transferred by users and dynamic business operation. The static structure of business entity usually appears as a *master-slave* relationship. The master is called the *core* entity, while the slave is called the *detail* entity.

*Entity* is the derived class of metaclass *Class* shown in Figure 1. *Entity* is derived into *MasterEntity* and *DetailEntity*. Each *Entity* has a set of properties, a primary key *PK*, 0 or more foreign key *FK*. The property *table* of *Entity* indicates the table name used to store the business data, and the property *where* represents the value range of the stored business data. *Attribute* is the derived metaclass from *Property* shown in

Figure 1. The property *fieldname* of *Attribute* indicates the stakeholder's column of the table.

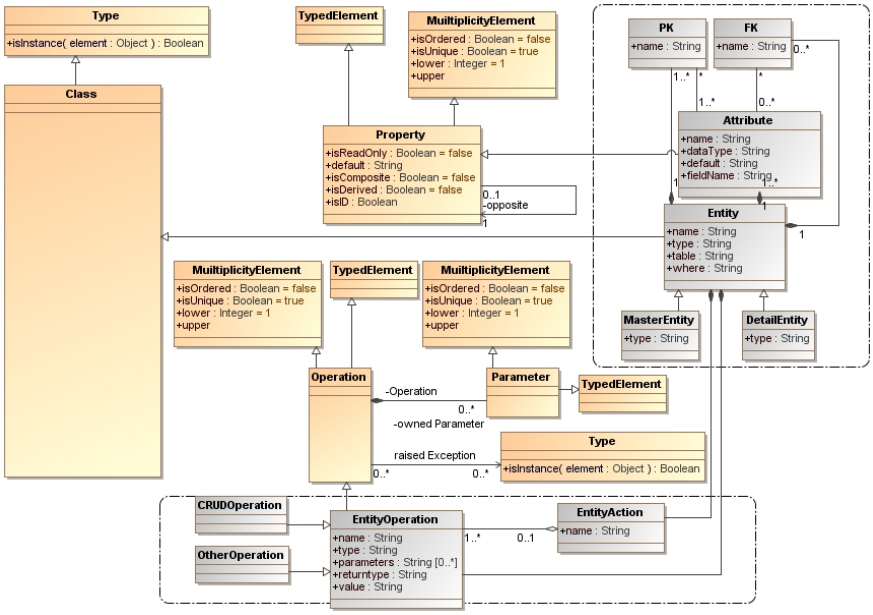


*Figure 1: Metamodel of PIM-BM*

First, a denotation ***Meta****(m, mm)=true* is given to present that model *m* is the instance of metamodel *mm*. In other words, metamodel *mm* is the abstract of model *m*.

Business entity is defined as 2-tuple, which is denoted as *BusinessEntity:= (entity, attributes)*, where ***Meta****(entity, Entity)=true*, *attributes={attribute| **Meta**(attribute, Attribute)=true}*.

### 3.1.2 Business Action

Business entity defines the static structure of the business model, while the behavior model is defined with the metamodel *EntityAction* and *EntityOperation*.

The metamodel of the action of business entity is shown in Figure 1. *EntityOperation* defines the concrete operation of entity, which is the sub metaclass of *Operation*. There are two basic derived operation metamodels: *CRUDOperation* and *OtherOperation*. *CRUDOperation* is a database-related business model, and its business is to execute a Structured Query Language (SQL) statement and call the Stored Procedure (SPROC) of database; *OtherOperation* represents the other business process.

Business action denoted as *EntityAction :={operation|**Meta**(operation EntityOperation,)}*. The difference of the *EntityAction* and *EntityOperation* is that the business modeled by *EntityOperation* is considered as instantaneous, uninterrupted

and atomic operation, while the business modeled by *EntityAction* need to take up some time. *EntityAction* is related to several *EntityOperation*s.

## 4 PSM-BC

The feature of a generic enterprise Web application is abstracted as a MIS which supports user-interaction in the Web based interfaces. A generic Web interaction can be decomposed into request, process and answer. The user sends a request to the Web server, usually via a Web page already visualized in a Web browser. Requests can be sent to the server either as forms, links or buttons; the Web server receives the request and performs various actions; the browser renders the results of the request.

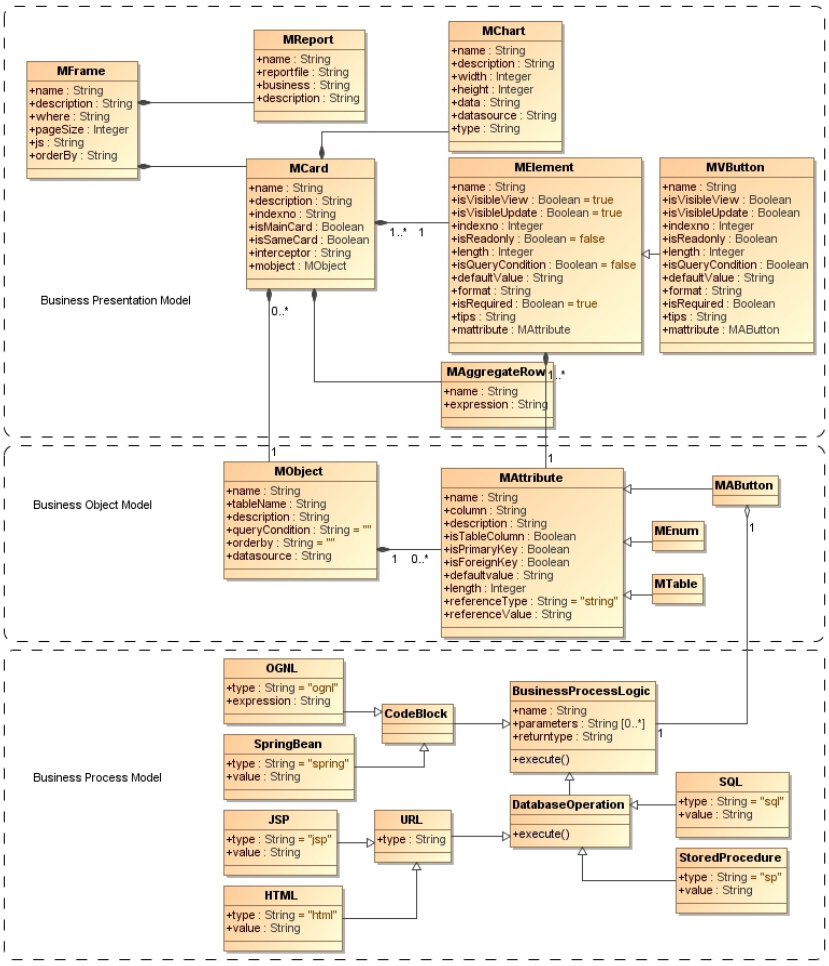


*Figure 2: Metamodel of PSM-BC*

In the UML specification, a component is a physical, replaceable part of a system that packages implementation and provides the realization of a set of interfaces. Business component is the soft implementation of business object, including the static and dynamic semantics. In the context of MDSD, the fine-grain components of the final system are generated from the instances of metamodel of PSM-BC in different layers. It indicate that the software conform to some features in the years of development experience. The features can also be extracted from the library of components. After the analysis of features, a web application is often made up of several forms. And each from is composed of the business data of a main table and some affiliated tables.

This paper proposes a novel platform-specified business component model named *PSM-BC* to describe the system business in the best possible way.

### 4.1 Metamodel of PSM-BC

PSM-BC defines business process, business object and business presentation object of the applications. A Web application is modeled from three points of view in order to reduce the complexity of models: business process model, business object model and business presentation model. Models in different layers are relatively independent with specific responsibilities and loosely coupled structure. A separation of design concerns into distinct model layers has several advantages such as ease of maintenance, oriented to the viewpoint, the ability to select specialized tools and techniques for specific concerns. The *PSM-BC*s conform to the metamodel shown in Figure 2. Systems are modeled as hierarchical collections of *MObject*, *MAttribute*, *MFrame*, *MCard*, *MElement* and so on.

#### 4.1.1 Business Process Model

Business process model describes the basic business logic of system including *Create*, *Read*, *Update* and *Delete* (CRUD) business, compound CRUD business and some special business. The derived model of business process is related to database-related manipulation, Uniform Resource Locator (URL), JavaScript, code blocks and so on.

Database-related manipulation is a direct operation of database, such as SQL and SPROC of the specific database; JavaScript defines some scripts based on Web browser; URL means a navigation of a Web page, such as *HTML* page and *JSP*; we propose a novel derived business process model named *SpringBean* to implement embedding codes in models. It will be discussed in late [Section 4.2].

Business process logic is denoted as $BP=\{bp|\boldsymbol{Meta}(bp, BusinessProcessLogic)=true\}$.

#### 4.1.2 Business Object Model

Business object model describes the organization of the business concepts managed by the Web application, which includes *MObject*, *MAttribute*, *MAButton*, *Reference*, and so on. In order to refine the details of business objects, it is divided into business object model *MObject* and the attribute model of business object *MAttribute*. In the context of Web modeling, *MObject* defines the name, description of a business object, table mappings (i.e. corresponding to the table of the relational database), and query condition (i.e. values range of business data represented by the instance of *MObject*).

*MAttribute* describes the property of the business object, including the name, description, column (i.e. corresponding to the key of the table of the relational database), and so on.

The most important property of *MAttribute* is the reference. *Reference* is made up of reference type and reference value. Reference type can be further broken into primitive data types and special reference types. Primitive data type is the data type identified by the system, such as *string*, *integer*, Universally Unique Identifiers (*uuid*) and *stringdate*. While special reference type includes *button* (user-defined button) and *enum* (enumerated data type). These two references require reference value that is additional information for the reference type. The reference value is a series of concrete enumerated values or a list of data for the data type *enum*; the reference value is the name of the business process model for the data type *button*. The derived model *MAButton* is the bridge between business process model and business object model.

Business object is defined as 2-tuple, denoted as *BO :=(mobject, mattributes)*, where ***Meta****(mobject, MObject)=true, mattributes={mattribute*| ***Meta****(mattribute, MAttributes)=true}*.

### 4.1.3 Business Presentation Model

Business presentation models contain the details of the graphic appearance of Web applications. It is composed of the instances of *MFrame*, *MCard*, *MElement* and *MVButton* and so on.

*MCard* is for the sake of the maintenance of a business object. The instance of *MCard* is related to several *MElement*s. *MElement* defines the smallest element of business presentation models, which may be the presentation of the business data. The important property of *MElement* is *isVisibleUpdate*, *isVisibleView* and *isQueryCondition*. When the property *isVisibleUpdate* is true, the *MElement* is a storage element. The business data represented by *MElement* can be modified in the maintenance user interface (UI); when the property *isVisibleView* is true, the *MElement* is a presentation element. The business data represented by *MElement* can be only displayed in the UI; when the *isQueryCondition* is true, it is as a query condition in the query area. Those are called *storage MElement*, *presentation MElement* and *query MElement* respectively. User-defined button *MVButton* is also a kind of *MElement*, and its specific business is defined in the property *referenceValue* of related *MAButton*. *MFrame* is the entrance to present business data for users.

Business presentation object is defined as 2-tuple, denoted as *VO :=(mcard, melements)*, where ***Meta****(mcard, MCard)=true*, *melements={melement*| ***Meta****(melement, MElement)=true*}. The Web UI object is denoted as *UI :=(mframe, vos)*, where ***Meta****(mframe, MFrame)=true*, *vos={vo*| ***Meta****(vo, VO)=true*}.

## 4.2 Extension mechanism of PSM-BC

In this Section, we provide the details of other extension mechanisms such as *expression*, *model interceptor* and *embedding source codes in models*, which might be useful in the implementation of the *PSM-BC*.

| Syntax | Semantics |
|---|---|
| *$C{constantName}* | A constant |
| *$S{parameterName}* | A session variable of HttpSession |
| *$R{parameterName}* | The value of specific parameter of HttpServletRequest |
| *$OGNL{expression}* | Access the member of a class or invoke the static method of class |
| *$SpEL{expression}* | SpEL is a powerful expression language that supports querying and manipulating an object graph at run time |
| *$SpringBean {variableName}* | SpringBean variable returns the execute method defined in the specific Spring configuration file |

*Table 2: Expressions*

*Expression* is a dynamic value, which is substituted at run time. Common types of expressions are constants, the requested variables, session variables, SpringBean variables, Object Graph Navigation Language (OGNL) expressions and Spring Expression Language (SpEL). The details are shown in Table 2.

In the basic case, *interceptor*s are inserted between a caller and a callee for method execution, which is defined in the configuration file of Spring Framework. For instance, the interceptor of *MCard* can define some extra work with *Spring Bean* before or after the maintenance of business data represented by this model.

Some business process is easy to describe by the source codes since not all the business behavior can be represented in models. Therefore, we propose a novel derived business process metamodel named *SpringBean*, which uses dependency injection [Tanter, Toledoa, Pothier and Noyé 08] and method interception [Mak, Rubio, Long (10)] techniques. *Aspect-Oriented Programming* (AOP) [Fuentes, Jimenez, Pothier and Pinto 06] currently supports method execution join points in the forms of the execution of methods on Spring Beans, which is implemented in pure Java. When the business process models use dependency injection, it becomes much cleaner and easier to follow. Some codes of the complicated business process can be embedded in models. Embedding source codes in models rather than rewriting the generate codes also can solve the problems of inconsistencies with the mixture of models and codes. This is a new way of the separation of manual codes and models. Finally, the embedded source codes are merged with the generated codes after model synchronization.

### 4.3 Schedule Model of PSM-BC

In the context of MDSD, schedule model may be a solution to the job that needs to occur at given moments in time. Some samples of scheduling are: system maintenance, reminder services and system monitor. Therefore, the *PSM-BC* is designed to support schedule models.

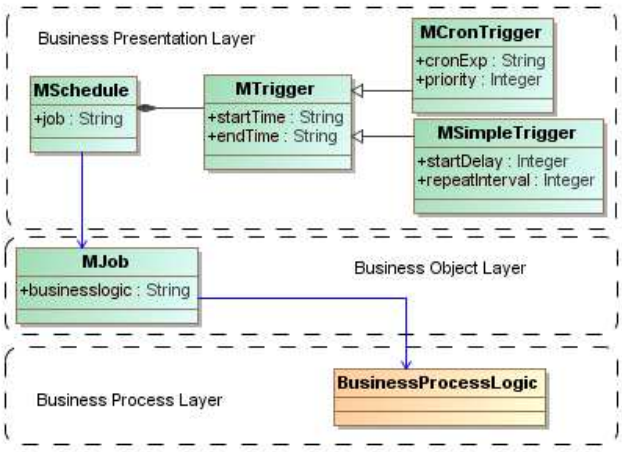


*Figure 3: Metamodel of Schedule Model*

We provide a new schedule metamodel to execute or trigger tens, hundreds, or thousands of jobs. Jobs whose tasks are defined by *MJob* metamodel in the server that may execute virtually anything represented in the *BusinessProcessLogic* model. Its metamodel is shown in Figure 3. A schedule model is denoted as *Schedule=(job, triggers)*, where ***Meta****(job, MJob)=true*, and *triggers={trigger|****Meta****(trigger, MTrigger) =true}*.

*MJob* defines a concrete task. Jobs are scheduled to run when a given trigger occurs. The property *businesslogic* of *MJob* is the business process model. Some complicated jobs can be defined with *SpringBean* model with a code block in the business process model.

*MTrigger* lies in the business presentation layer for triggering the jobs, which is derived into *MCronTrigger* and *MSimpleTrigger*. In the metamodel *MTrigger*, the property *startTime* is the start time of the job, while the property *endTime* is the end time of the job.

A *MSimpleTrigger* that is used to fire a job at a given moment in time, and optionally repeated at a specified interval. With this description, you may not find it surprising to find that the properties of a *MSimpleTrigger* include: start time, end time, start delay, and repeat interval. The property *repeatInterval* must be zero or a positive long value, representing a number of milliseconds.

If you need a job-firing schedule that recurs based on calendar-like notions rather than on the exactly specified intervals, *MCronTrigger* is often more useful than *MSimpleTrigger*. A *MCronTrigger* uses *cron expressions*[1] to create firing schedules. The property *cronExp* is a cron expression; the property *priority* indicates the importance of the job. Cron expressions are strings that are actually made up of seven sub-expressions that describe individual details of the schedule. The fields are shown

[1] Cron is a time-based job scheduler in Unix-like computer operating systems. Cron enables users to schedule jobs according the cron expression.

in Table 3. Wild-cards (the "*" character) can be used to say every possible value within a field. For example, "*" in the minute field means "every minute"; the "?" character is used to specify "no specific value", which is allowed for the day-of-month and day-of-week fields; the "/" character can be used to specify increments to values. For example, "0 15 00 * * ? 2011" means firing at 00:15am every day during the year 2011.

| Field Name | Mandatory | Allowed Values | Allowed Special Characters |
|---|---|---|---|
| Seconds | YES | 0-59 | , - * / |
| Minutes | YES | 0-59 | , - * / |
| Hours | YES | 0-23 | , - * / |
| Day of month | YES | 1-31 | , - * ? / L W |
| Month | YES | 1-12 | , - * / |
| Day of week | YES | 1-7 | , - * ? / L # |
| Year | NO | 1970-2099 | , - * / |

*Table 3: The format of cron expression*

# 5 Model Transformation Approach using QVT Relations

## 5.1 Formal definition of model transformation

Model transformation is the process of converting one model to another model of the same system [Miller and Mukerji 03]. QVT Specification has given some model transformation languages instead of the definition of model transformation. There is no accepted formalizing definition of model transformation [Didonet, Fabro, Bézivin, Jouault, Valduriez 05]. Thus, we develop our own full definitions.

Model transformation is the process of converting source models to target models of the same system. Its semantics of model transformation is defined by a group of mapping rules. We use $m(s)/f$ to denote a model $m$ of the system $s$ in the formalism $f$. A model mapping is a transformation $m_1(s)/f_1 \rightarrow m_2(s)/f_2$, shortened $m_1/f_1 \rightarrow m_2/f_2$. Given $S$ as the source model and $T$ as the target model, model transformation is defined as 3-tuple, denoted as $MT:=(F, S, T)$, where $F$ is a set of mapping rules, denoted as $F=\sum r$. The semantics of model transformation may be also expressed as a model. A transformation is considered as a special model $MT(S \rightarrow T)/F$, where $S$ and $T$ are the source models and target models respectively. We use *MediniQVT* [ikv++ technologies 11] as transformation engine of QVT-Relations. mediniQVT is implemented on the EMF framework, and uses the EMF generated Java classes to manipulate models. It is a complete QVT implementation, supporting the expressive power defined by QVT language, and satisfying the properties we require as assumptions.

## 5.2 Model transformation from PIM-BM to PSM-BC

In order to eliminate the heterogeneous of the model transformation, a relational model transformation approach conformed to the OMG QVT Specification is

proposed. We present the QVT relations language and discuss how it addresses the mapping rules.

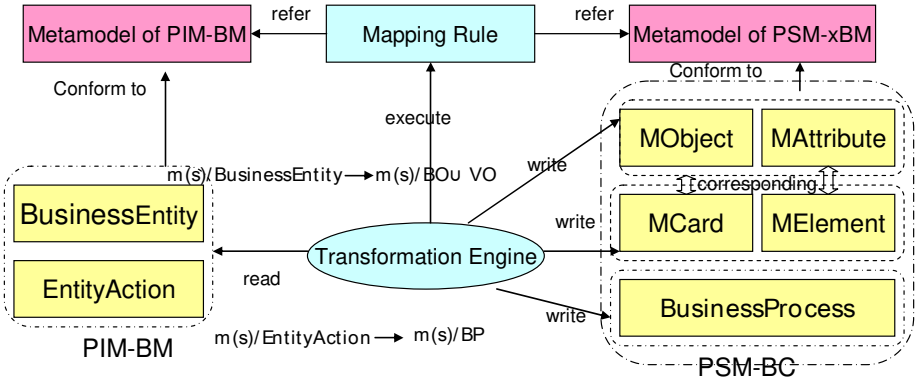


*Figure 4: The process of model transformation*

The process of model transformation is shown in Figure 4. *BusinessEntity* shown in Figure 1 is transformed into *MObject*, *MAttribute*, *MCard* and *MElement* shown in Figure 2, and *EntityAction* shown in Figure 1 is transformed into *BusinessProcessLogic* shown in Figure 2.

### 5.2.1 Transformation from Business Entity to Business Object Model

Business Entity is transformed into business object model, which is denoted as $m_1(s)/BusinessEntity \rightarrow m_2(s)/BO$ shown in Figure 5, where $BusinessEntity \subset PIM\text{-}BM$ , $BO \subset PSM\text{-}BC$.

This transformation *EntityToMObject* specifies a mapping between (any) two models that are instances of metamodels *Entity* and *MObject*, and the transformation *AttributeToMAttribute* specifies a mapping between (any) two models that are instances of metamodels *Attribute* and *MAttribute*. The query *sizeof* returns the length of the data type, and the query *typemapping* is a mapping rule of data type.

```
top relation EntityToMObject {//R1
   n:String; t:String; w:String;
   enforce domain source entity:PIM-BM::Entity{
     name=n, table=t, where=w };
   enforce domain target mobject:PSM-BC::MObject{
     name=' O_'.concat(n), tableName=t, queryCondition=w};
}
top relation AttributeToMAttribute {//R2
   n:String; de:String; dt:String; f:String;
   enforce domain source attribute:PIM-BM::Attribute{
     name=n, default=de, dataType=dt, fieldname=f, entity=e:PIM-
BM::Entity{} };
   enforce domain target mattribute:PSM-BC::MAttribute{
     name='A_'.concat(n), column=f, isTableColumn=true,
     defaultvalue=de, length=sizeof(dt), referenceType=typemapping(dt),
     mobject=m:PSM-BC::MObject{} }
   when{
     EntityToMObject(e,m); }
query typemapping(typename : String) : String {
     if typename = 'Integer' then 'Integer'
     else if typename = 'String' then 'String'
     else if typename = 'Boolean' then 'Boolean'
     else if typename = 'Real' then 'Double'
     else if typename = 'EnumerationLiteral' then 'Enum'
     else if typename = 'CollectionType' then 'Table'
     if typename = 'Button' then 'Button' endif
     endif endif endif endif endif endif}
query sizeof (typename : String) : Integer{
     if typename = 'Integer' then 11
     else if typename = 'String' then 50
     else if typename = 'Boolean' then 1
     else if typename = 'Real' then 11
     endif endif endif endif}
}
```

*Figure 5: The mapping rule from BusinessEntity to BO using QVT Relations*

### 5.2.2 Transformation from Business Entity to Business Presentation Model

Business entity is transformed into business presentation model, which is denoted as $m_1(s)/BusinessEntity \rightarrow m_2(s)/VO$ shown in Figure 6, where $BusinessEntity \subset PIM\text{-}BM$ , $VO \subset PSM\text{-}BC$.

```
top relation EntityToMCard {//R3
   n:String; t:String;
   enforce domain source entity:PIM-BM::Entity{
     name=n, type=t };
   enforce domain target mcard:PSM-BC::MCard{
     name='C_'.concat(n), isMainCard=ifMainCard(t) };
   query ifMainCard(typename : String) : Boolean{
     if typename = 'core' then true else false endif }
}
top relation AttributeToMElement{//R4
   n:String; de:String; dt:String;
   enforce domain source attribute:PIM-BM::Attribute{
     name=n, default=de, dataType=dt, entity=e:PIM-BM::Entity{}};
   enforce domain target melement:PSM-BC::MElement{
     name='E_'.concat(n), defaultvalue=de, length=sizeof(dt),
     mcard=c:PSM-BC::MCard{} };
   when{
     EntityToMCard(e,c);
   }
   where{
        if dt=Integer then
          defaultValue=0
          format='^-?\d+$'
        else if dt=Button then
           isQueryCondition=false
           defaultValue=''
           format=''
           endif
           else if dt=Real then
             defaultValue=0
             format='^\d+\.?\d+$|^\d+$'
           endif
           else if dt=String then
                defaultValue=''
              endif
              else if dt=stringdate then
                   defaultValue=''
                   format='yyyyMMdd '
                 endif
        endif
        ;
   }
}
```

*Figure 6: The mapping rule from BusinessEntity to VO using QVT Relations*

This transformation *EntityToMCard* specifies a mapping between (any) two models that are instances of metamodels *Entity* and *MCard*, and the transformation *AttributeToMElement* specifies a mapping between (any) two models that are instances of metamodels *Attribute* and *MElement*.

### 5.2.3 Transformation from Business Operation to Business Process Model

Entity operation is transformed into business process model, which is denoted as $m_1(s)/EntityOperation \rightarrow m_2(s)/BP$ shown in Figure 7, where $EntityOperation \subset PIM\text{-}BM$ , $BP \subset PSM\text{-}BC$.

```
top relation EntityOperationToBusinessProcessLogic{//R5
  n:String; params:String; ret:String; v:String; t:String;
    enforce domain source entityOperation:PIM-BM::EntityOperation{
      name=n,parameters=params,returntype=ret,value=v,type=t };
    enforce domain target bp:PSM-BC::BusinessProcessLogic{
      name='P_'.concat(n),parameters=params,returntype=ret,
      value=valuemapping(entityOperation),type= typemapping(t)};
    query typemapping(typename : String) : String{
      if typename = 'crud' then 'SQL' else 'SpringBean' endif}
    query valuemapping (entityOperation: EntityOperation) : Object{
      if entityOperation.type = 'crud' then entityOperation.value
      else entityOperation.execute(entityOperation.parameters) endif}
  }
```

*Figure 7: The mapping rule from BusinessEntity to BP using QVT Relations*

This transformation *EntityOperationToBusinessProcessLogic* specifies a mapping between (any) two models that are instances of metamodels *EntityOperation* and *BusinessProcessLogic.*

## 5.3 Model synchronization mechanism

### 5.3.1 Version control of models

In MDSD, models are the primary artifacts of the software development process. Like other software artefacts, models undergo a complex evolution during their life cycles. As a consequence, there is a growing need for techniques and tools to support model evolution activities such as version control. Present-day MDSD tools offer limited support for the version control of models. Traditional version control systems are based on the *copy-modify-merge* approach [Collins-Sussman, Fitzpatrick, Pilato (08)], which is not fully exploited in MDSD since current implementations lack model-orientation.

In contrast, we use Java Content Repository (JCR) [Nuescheler 10] as the storage of models. A content repository consists of one or more workspaces, each of which contains a tree of items. An item is either a *node* or a *property*. The structure of content repository is shown in Figure 8. Each node may have zero or more child

nodes and zero or more child properties. There is a single root node per workspace, which has no parent. All other nodes have one parent. This structure is similar to the model. The model may be considered as a *node*, and the property of the model may be considered as a *property*. In JCR 2.1 (JSR-333) [Nuescheler 10], it provides simple versioning or full versioning of *node* in the repository. A versioning repository has, in addition to one or more workspaces, a special version storage area. The version storage consists of version histories. A version history is a collection of versions connected to one another by the successor relationship. A new version is added to the version history of a versionable *node* when one of its workspace instances is checked-in. The model stored in the repository can be restored to a previous version according to the version number, which is useful when developers have made some fatal mistake in modeling the system. This model evolution approach conforms to JSR specification, which is independent of the metamodel of models and has a strong commonality and extensible ability. Some open source tools have implemented the JCR 2.1 specification, such as Jackrabbit and ModeShape[2].

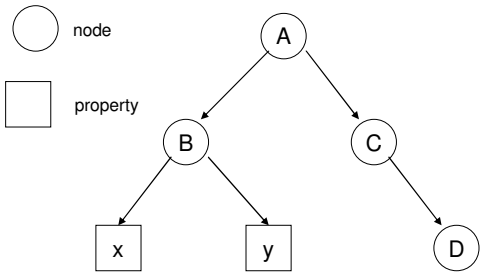


*Figure 8: The structure of Java Content Repository*

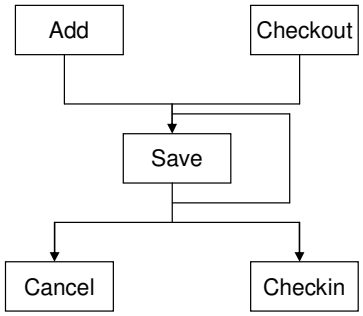


*Figure 9: Version control of the models in JCR*

The process of version control of models in JCR is shown in Figure 9. The states of a node are divided into draft, approved and revised. The format of version number of nodes is *MajorNum.MinorNum.RevisionNum*. The rule of increase of version is:

[2] Apache Jackrabbit and ModeShape are both a fully conforming implementations of JSR-283 specification, which can be downloaded from http://jackrabbit.apache.org and http://www.jboss.org/modeshape respectively.

the *MinorNum* increases by 1 for the draft node; the *RevisionNum* increases by 1 for the revised node. When a new versionable node is created, a new version history is created for it. On save of a node, if the state of node is draft, the draft version is saved and the minor number increases by 1; if the state of node is revised, only a new versionable node is created and the revision number increases by 1. To create a new version of a versionable node, the application calls *checkin*. If the state of node is draft, change the state to approved and change the revision number to 1.0.0; if the state of node is revised, change the state of the last draft node to approved and increase the minor number by 1, set revision number to 0.

### 5.3.2 Model synchronization based on the version of models

The development of system is an iterative process with frequent modifications to the involved models according to user requirements.

Source models involved in model synchronization may face with the modifications shown in Table 4. The modifications in the three circumstances are identified based on the version number. All the version numbers of models involved in model synchronization are recorded. In the subsequent model synchronization, the version of involved models is needed to compare with the last recorded version. If the model is not in the last recorded models, it is *addition*; if the new version number is greater than the past and meantime it is not a new model, it indicates that the model is *updated* after the last model synchronization; if one of the last recorded models is not involved in the subsequent model synchronization, it indicates that the source model has been *deleted*.

| **Name** | **Definition** |
|---|---|
| *add* | Source model is added. |
| *delete* | Source model is deleted. |
| *update* | The property of source model is changed. |

*Table 4: Classification of modifications of involved models in model synchronization*

Before model synchronization, *PIM-BM*s are detected whether they need subsequent model transformation or not. A transformation creates target codes if it is missing on the first execution. A subsequent execution with the same model as in the previous execution has to detect that the needed code already exists. This detection can be achieved by using version number of involved models. Only *PIM-BM*s with changed version number are synchronized to regenerated codes, while the source models with no modification keeps constantly.

Compared to these current approaches in [Section 2.4], our approach of model synchronization only takes the repository space instead of the extra needed space for the execution context. This detection whether the source model need subsequent execution can be easily achieved based on the revision number of models. Therefore, this approach is appropriate to the model transformation with large source models. When any of the source models are modified, the necessary changes of models are determined from the model repository. At the same time, the target elements that can

be preserved are preserved. The implementation of this approach is based on the JCR specification, which is simple and easily integrated with the current MDA tools.

Finally *PSM-BC*s are transformed into codes of applications. The implementation of code generation is based on the *textual template evolution* in our previous work [Chen, Ma, Abraham, Yang, Sun 10]. The final Web applications are made up of models, model execution engine, and the generated codes instead of traditional Web distribution.

## 6 Conclusions and Future Work

In this paper, our method provides a resolution to the model transformation from PIM and PSM. A novel *PIM-BM* and *PSM-BC* is proposed to describe both the structural and behavioral properties of generic Web applications at different levels of abstraction. The metamodel and extension mechanism are discussed in detail. In addition, a relational model transformation approach from *PIM-BM* to *PSM-BC* is proposed. In order to eliminate the heterogeneous of the model transformation, this approach uses *Relations* language to present the mapping rules for conformance to QVT Specification.

Compared with other model transformation approaches, we provide model synchronization mechanism based on the versions of model. All the *PIM-BM*s and *PSM-BC*s are stored in the repository. Before model synchronization, *PIM-BM*s are detected whether they need subsequent model transformation or not. A transformation creates target codes if it is missing on the first execution. This approach of model synchronization will only take the storage space of model repositories rather than some extra space. Only the models with changed version number need a subsequent model transformation. This way is named *source incrementality*, which is simple and useful for working with large scale source models. In this way, model synchronization is a special and partial model transformation. That is a good way to minimize the amount of source that needs to be re-examined by a transformation when the source is changed.

Our model transformation approach supported model synchronization is the prerequisite to the code generation in the process of MDSD. We have tested all the mapping rules from PSM-BM to PSM-BC using *mediniQVT*. The generated PSM-BCs from PIM-BMs are sufficiently to describe the business. It has generated some real enterprise Web applications. The distribution and uninterrupted running of the generated applications proves that our approach is feasible in practice. This model-driven development method can speed up the software development, which is particularly appropriate for the applications with frequent changes of business.

Future work is targeted in two directions to complete and improve the current proposal. The first target is to provide a formalization of the version control approach and a Web UI for the management of the model repository. We plan to further support visual UI for a full-text search engine of *PSM-BC*s saved in the content repository. Second, we will provide a visual model-driven rapid development platform, which is easier to model, execute model transformation and synchronization, and generate the practical enterprise Web applications.

**Acknowledgements**

This work was supported by the Provincial Natural Science Foundation for Outstanding Young Scholars of Shandong under Contract Number JQ200820, Technology development Program of Shandong Province under Contract Number 2011GGX10116, the Program for New Century Excellent Talents in University under Contract Number NCET-10-0863 and Youthful Science and Technology Star Plan of Jinan under Contract Number 20110112.

## References

[Akehurst and Kent 02] Akehurst, D.H., Kent, S.: "A Relational Approach to Defining Transformations in a Metamodel"; Proc. UML'02, Springer-Verlag, Germany (2002), 1-15.

[Bissell 03] Bissell, A.: "UML 2.0 - A major revision of the industries de facto software modeling language"; Aircra Eng Aerosp Tec (Aircraft Engineering and Aerospace Technology), 75, 2 (2003), 178-181.

[Chen, Ma, Abraham, Yang, Sun 10] Chen, Z., Ma, K., Abraham, A., Yang, B., Sun, R.: "An Executable Business Model for Generic Web Applications"; Proc. CISIM'10, IEEE Computer Society, Kraków (2010), 573-577.

[Collins-Sussman, Fitzpatrick, Pilato (08)] Collins-Sussman, B., Fitzpatrick, B. W., Pilato, C. M.: " Version Control with Subversion "; O'Reilly Media, Sebastopol, Calif. (2008)

[Czarnecki and Helsen 06] Czarnecki, K., Helsen, S.: "Feature-based survey of model transformation approaches"; IBM Systems Journal (IBM Systems Journal), 45 (2006), 621-645.

[Didonet, Fabro, Bézivin, Jouault, Valduriez 05] Didonet, M., Fabro, F., Bézivin, J., Jouault, F., Valduriez, P.: "Applying Generic Model Management to Data Mapping"; Proc. BDA '05, Actes Publishing, Saint Malo (2005), 1-13.

[Efftinge, Friese, Köhnlein 08] Efftinge, S., Friese, P., Köhnlein, J., Best Practices for Model-Driven Software Development, 2008, http://www.infoq.com/articles/model-driven-dev-best-practices.

[Frankel (03)] Frankel, D.S.: "Model Driven Architecture: Applying MDA to Enterprise Computing"; Wiley, Hoboken (2003).

[Fuentes, Jimenez, Pothier and Pinto 06] Fuentes, L., Jimenez, D., Pinto, M., J. Noyé: "Development of Ambient Intelligence applications using components and aspects"; J. UCS (Journal of Universal Computer Science), 12, 3 (2006), 236-251.

[Giese and Wagner 09] Giese, H., Wagner, R.: "From model transformation to incre-mental bidirectional model synchronization"; SoSyM (Software and Systems Modeling), 82, 1 (2009), 21-43.

[Hearnden, Lawley, Raymond 06] Hearnden, D., Lawley, M., Raymond, K.: "Incremental Model Transformation for the Evolution of Model-Driven Systems"; Proc. MoDELS'06, Springer, Copenhagen (2006), 321-335.

[Hwan, Kim, Czarnecki 05] Hwan, C., Kim, P., Czarnecki, K.: "Synchronizing Cardinality-Based Feature Models and Their Specializations"; Proc. ECMDA-FA'05, Springer, Copenhagen (2005), 331-348.

[ikv++ technologies 11] ikv++ technologies, mediniQVT 1.7.0, 2011, http://projects.ikv.de/qvt.

[Kim, Choi, Kang, Lee 10] Kim, H., Choi J., Kang, I., Lee I.: "UML Behavior Models of Real-Time Embedded Software for Model-Driven Architecture"; J. UCS (Journal of Universal Computer Science), 16, 17 (2010), 2415-2434.

[Kurtev 08] Kurtev, I.: " State of the Art of QVT: A Model Transformation Language Standard "; Lect Notes Comput Sci (Lecture Notes in Computer Science), 5088 (2008), 377-393.

[Madari, Lengyel 09] Madari, I., Lengyel, L.: "Synchronizing user interfaces of different mobile platforms"; Proc. EUROCON'09, Springer, Copenhagen (2009).

[Mak, Rubio, Long (10)] Mak, G., Rubio, D., Long, J.: "Spring Recipes: A Problem-Solution Approach"; Apress, New York (2010).

[Miller and Mukerji 03] Miller, J., Mukerji, J., MDA Guide Version 1.0.1, 2003, http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf.

[Nuescheler 10] Nuescheler, D., JSR 333: Content Repository for Java Technology API Version 2.1, 2010, http://jcp.org/en/jsr/summary?id=333.

[Nolte (09)] Nolte, S.: "QVT - Relations Language"; Springer Xpert Press, Heidelberg (2009) [Object Management Group 10] Object Management Group, Unified Modeling Language 2.3, 2010, http://www.omg.org/spec/UML/2.3/.

[Object Management Group 06] Object Management Group, Meta Object Facility 2.0, 2006, http://www.omg.org/spec/MOF/2.0.

[Object Management Group 09] Object Management Group, Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Version 1.1, 2009, http://www.omg.org/spec/QVT/1.1.

[Rosado, Fernández-Medina, López, Piattini 10b] Rosado, D. G., Fernández-Medina, E., López, J., Piattini, M.: " Developing a Secure Mobile Grid System through a UML Extension"; J. UCS (Journal of Universal Computer Science), 16, 17 (2010), 2333-2352.

[Rumbaugh 04] Rumbaugh, J.: " Raising the Level of Development: Models, Architectures, Programs"; IBM developerWorks Live! Report, Beijing, China (2004).

[Sánchez, Moreira, Fuentes, and Magno 10] Sánchez, P., Moreira, A., Fuentes, L., Magno, J.: "Model-driven development for early aspects"; Inform Software Tech (Information and Software Technology), 52, 3 (2010), 249-273.

[Subramanyam, Weisstein, and Krishnan 10] Subramanyam, R., Weisstein, F.L., Krishnan, M.S.: "User participation in software development projects"; Commun Acm (Communications of the ACM), 53, 3 (2010), 137-141.

[Tanter, Toledoa, Pothier and Noyé 08b] Tanter, É., Toledoa, R., Pothier, G., J. Noyé: "Flexible metaprogramming and AOP in Java"; Sci Comput Program (Science of Computer Programming), 72, 1-2 (2008), 22-30.

[Wahler 04] Wahler, M.: "Formalizing Relational Model Transformation Approaches"; Swiss Federal Institute of Technology Zurich Report 998, Zurich, Switzerland (2004).