BORM Model Transformation

Petr Splichal, Robert Pergl, Marek Pícka
Czech University of Life Sciences Prague
Faculty of Economics and Management
Department of Information Engineering
Kamýcká 129, Praha 6 Suchdol, 165 21,
{splichal, pergl, picka}@pef.czu.cz

Abstract. The paper presents the partial achievements of our research on transformations of the models between various phases of the analysis of information systems. Business Object Relationship Modelling was chosen as a starting point of the transformation due to business oriented approach to analysis. Its models are studied and along with its meta-model are presented here. Unified Modelling Language was selected as the output of the transformation. The output is represented by the class diagrams in UML. Simplified meta-model of the UML class diagram is presented as well. These meta-models represent the two sides of the transformation. The rules and recommendations for the transformation process between those two sides are presented in the paper. The transformation process brings new ways of knowledge exchange between the business and technology sectors.

Key words: BORM, UML, class diagram, business analysis, transformation

1. Introduction

This paper introduces the basic issues and work procedures of the analysis and transformation in the field of BORM models and UML class diagrams. The core of the paper is the conjunction of the concepts created to support the transformation of the models from one notation to the other.

2. Motivation

As M.B.Kuznetsov explains in [9], an effective application of the MDA software development calls for methods and techniques for specifying of software model transformations like the one we are describing in this paper.

Existing CASE tools support some ideas of concept transitions, e.g. the Craft.CASE (http://www.craftcase.com) modelling tool performs checks, whether the added element conforms to the method. To further improve the quality of an analyst’s work, it would be a great contribution to implement complex support for the concept transitions model. A CASE tool could thus better lead an analyst through the process of analysis, give him hints, check and record his steps [1].

3. Goal

The authors of [2] stated that models are becoming essential in development of complex IT systems and model transformation represents key activity. The goal of the research is to identify the group of such methods that could serve as a foundation for CASE tools supporting transformation of the analytic models’ parts created during the
business analysis into the computational model independent on the specific computer platform. Business processes models of the problem domain are the source part of the business analysis. A new set of design patterns, available for the IT professionals to guide them during the transformation of the analytic models, will be also identified.

4. Methodology

Business Object Relationship Modelling [5,6] was chosen as a starting point of the transformation and a representative of a business modelling methodology due to its expressive notation. Unified Modelling Language [10,11] was selected as the output of the transformation from the BORM business models. Main reason for the usage of UML as a destination modelling language is the fact that the UML is a widely used standard in information and communication technology and UML is well-known among the IT professionals. The class diagrams in UML are the output of the transformation process. Meta-model of the BORM business models and meta-model of the UML class models represent the two sides of the transformation.

The work introduced in this paper builds on the best-practices introduced below and authors’ experience and knowledge gathered during participation on projects in practice. Related approaches used in similar research projects derived from other methodologies and technologies are also taken into account.

4.1 Business Analysis

The International Institute of Business Analysis (IIBA™) defines business analysis as “the set of tasks and techniques used to work as a liaison among stakeholders in order to understand the structure, policies, and operations of an organisation, and recommend solutions that enable the organisation to achieve its goals” (www.theiiba.org).

Barbara A. Carkenord states in [3] that business analysis consists of elicitation of needs and constraints from stakeholders, analysis of stakeholder needs to define requirements for a solution, assessment and validation of potential and actual solutions and management of the “product” or requirements scope.

Organisations have introduced business analysis to make sure that business needs are paramount when new computer systems are being introduced. However, recognising the importance of this principle is easier than considering how this might be achieved [4]. The concept of Business analysis is mainly used for the first phases of the information system’s analysis. Performing the business analysis can have a different reasons and one of them is the formal description and storage of the initial phases of the information system’s development. Mainly a formal description of these phases needs to be recorded using user-friendly and simple techniques due to the needs of easy verification and validation. This fact precludes the use of the many tools for the software engineering because of their complex structure, interface or working techniques. Detailed description of these phases is vital for the later audits of the system’s business alignment. The main functions of a future information system (and sometimes also an existing information system) are described during the business analysis.

Tools for the business analysis have to support simple and general system of the notation of the system functions. The main reason for this need is the involvement of
the stakeholders who doesn’t know complex notations. Classic tools for the software engineering are well designed for the area of analysis, but their complexity and demands for understanding by far exceeds the capabilities of the stakeholders without a deep knowledge of this domain.

BORM has proven to be effective and beneficial in the process of describing and subsequently understanding how real business systems evolve. This approach is especially beneficial for those employees who are responsible for business development [4].

4.2 BORM

Business Object Relationship Modelling [5, 6] is an object-oriented software engineering methodology, which has proven to be very effective in the development of business information and knowledge systems. Its effectiveness is achieved by a unified and simple method for presenting all aspects of the relevant model. The BORM methodology makes extensive use of business process modelling [5]. BORM was designed as a method covering all phases of the software development. BORM has the main focus on the first phases of the project also known as business analysis. BORM uses only a limited, easily comprehensible group of concepts for every lifecycle phase. This makes it easy to understand even for the first-time users with almost no knowledge of business analysis.

Another fact that makes the BORM methodology more expressive is that it doesn’t need the division to static and dynamic views of the model and therefore does not bring a need of creation of different diagrams with a different view points.

BORM introduces the following types of diagrams:

- Business architecture diagram
- Object relationship diagram
- Class diagram

BORM represents every concept with the same symbols in the data structure, the communication or other diagrams. BORM uses for visual presentation of the information simple diagrams that contain only a necessary number of concepts and symbols. These concepts and symbols cover most of the needs for the initial description of the model and its processes. The following symbols belong to the symbols used in initial description:

- **Participant** – an object representing the stakeholder involved in one of the modelled processes, which is recognised during the analysis.
- **State** – sequential changes of the participants in time are described by these states.
- **Association** – data-orientated relation between the participants.
- **Activity** – represents an atomic step of the behaviour of the object recognised during the analysis.
- **Communication** – represents the data flow and dependencies between the activities. Data may flow bidirectionally during the communication.
- **Transition** – connects state-activity-state and represents changes of the states through activities.
- **Condition** – expresses constraint that holds for the communication or activity [6].
4.3 MDA

Model-Driven Development [MDD] approach to software development suggests that one should first develop a model of the system under study, which is then transformed into the real thing (i.e., an executable software entity). The most important research initiative in this area is the Model-Driven Architecture [MDA], which is being developed under the umbrella of the Object Management Group [7].

MDA is not itself a technology specification but it represents an evolving plan to achieve cohesive model-driven technology specifications. MDA is built on OMG standards including the UML [8]. Models represent a major part in the whole concept of MDA as its name suggests. MDA uses models as a driver of the development lifecycle.

Another important part of MDD is Meta-Object Facility [MOF]. MOF is a specialised modelling language for defining meta-models. MOF provides a metadata management framework, and a set of metadata services to enable the development and interoperability of model-driven systems.

We use MDA and MOF in our research to accomplish the transformation of BORM model. Approach to model transformation used in MDA combined with MOF serves as a foundation of our transformation from BORM model to UML Class diagram.

4.4 UML Class Diagram

Unified Modelling Language (UML) Class Diagram is a static diagram. The Class diagram is probably the most widely used diagram of the UML. In fact, the Class diagram is the primary modelling tool for describing UML itself [10]. This diagram describes the data structure of the system and displays its classes along with their attributes and relations between them. Class diagram is a foundation of the database structure of the final application.

Sometimes it is used for modelling the ontology of the system [7]. This implies a decision that is based on which concepts or entities are part of the system and which concepts or entities are outside its boundaries [11].

This diagram is used for capturing of the static part of the system which is mainly represented by the relations between the objects. UML defines a few types of the relations between the objects in the class diagram besides the description of the classes:

- **Class** – represents an object identified during the analysis,
- **Attribute** – represents an attribute of the object identified during the analysis,
- **Method** – activity performed by the object identified during the analysis,
- **Association** – general relation between the objects,
- **Aggregation** – relation between the objects in which is one object consists of the other, but each of them may exist separately,
- **Composition** – relation between the objects in which is one object consists of the other and the existence of the contained object alone won’t make any sense,
- **Generalisation** – relation representing the inheritance between the classes.

Class diagram can be different for each of the phases of the system development and analysis and it becomes detailed during the development.
4.5 Method of gradual transformations

The main idea behind the method of gradual transformations deals with the way that an analyst follows during the modelling of an information system. An analyst adds a single new item to the model in a certain moment. The new item (class, method, generalization, etc.) is added there because of a decision based on items already present in the model [1, 12].

The method of gradual transformations arises from the following basic postulates:

- Every new item added to the information system’s destination model has to have a meaning in this model.
- Every new item that is not a basic domain item, is created on a basis of items already existing in destination model using a transformation.
- The transformation remodels the destination model into a new destination model by addition of exactly one new item already existing in the destination model.
- New basic domain item may be added to destination model from outside of the system by a “feed-in” transformation.

We will construct a new model layer which will hold all transformations used for creation of this model, if we stick to the above-mentioned postulates during the model construction. For every item, we can find the original items from which the destination item is created.

The transformations used during the construction of the model are given by the used software development method. Transformation itself could be described by a series of transformation rules.

Transformation rules consist of

- Header – defines the name of transformation, types and names of inputs,
- Input conditions – transformation can proceed only when it has all the needed inputs and all the input conditions are met.
- Transformation algorithm – describes the transformation of the inputs to output element.

The software development method may be then described as a set of all transformation rules allowed in this method.

Atomic transformations should be made very simple. Complex transformations should be created by composing atomic transformations.

Transformations defined by these transformation rules may be of several kinds: vertical (between diagrams of different types), horizontal (between diagrams of the same type), exogenous or endogenous.

5. Related work

The transformation of models presented in this paper brings new approaches to business analysis and modelling in UML. As List and Korherr explain in [13], current UML Profiles for Business Process Modelling realise a narrow focus of the process, and capture the process flow on a low level of detail. They do not provide a comprehensive coverage of domain ontological aspects. List and Korherr approached this problem by creation of a new profile which is described in [13].
Another attempt to make UML more reality-oriented is the OntoUML [14] extension. This represents a very powerful tool, but it is currently limited to structure class diagrams.

The method described in this paper enables to use the process-oriented analysis in BORM and it ensures a consistent methodological transformation to the industry standard UML.

6. Results and discussion

This paper summarises the achievements of our research work in this field so far. The majority of the current outcomes provide a theoretical foundation which will be utilised in the upcoming phases of the built methodology.

The basic idea behind the transformation is using the meta-model for consistent specification of the transformation rules. A meta-model is a model that defines the language for expressing a model, i.e. "a model of models". A meta-model is an explicit model of the constructs and rules needed to build specific models. It is a description of all the concepts that can be used in a model [7]. Meta-model contains all important information stored in the diagram by its author, but is free of any limitations of the diagram itself.

Transformation presented in this paper is based on the techniques and approaches presented in section 4. Meta-model of the BORM diagram and meta-model of the UML class diagram are both models describing constructs and rules needed for their creation. The conversion of the diagram created in BORM to meta-model takes us closer to the meta-model of the UML class diagram and also makes the search for the common attributes of both easier.

We use the meta-model of the BORM for the needs of this transformation, which is presented in Figure 1. The model is expressed using the UML class diagram. Diagram captures the basic parts (typically abstract) of the meta-model and other auxiliary elements like comments.

![Fig. 1. Meta-model of BORM basic elements [12]](image-url)
Meta-model of the UML class diagram is also needed to be able to specify the rules of the conversion between this meta-model and the BORM meta-model. We have created a simplified version of the UML class diagram for the needs of this paper. This version, presented in Figure 2, illustrates the basic idea and its utilisation.

![Fig. 2. Simplified meta-model of UML Class diagram](image)

The conversion rules are then specified. These rules are based on the knowledge of the syntax and semantics of the models. BORM model may contain both the static and dynamic information, so one BORM model may be transformed to more than one UML models. A part of the BORM ORD diagram is shown in Figure 3.

![Fig. 3. BORM – Business diagram](image)

We have created the first set of transformation rules in our project so far. This set is still incomplete and lacks some specific rules. We will present one composed transformation along with its atomic transformations as an example to illustrate the method in action.
The example presents the transformation of the communication between the Originator and Marketing assistant Participants from Figure 3. The communication will be transformed using the composed transformation. This transformation is presented in Figure 4.

![Fig. 4. Composed transformation of BORM's communication concept](image)

This composed transformation has to be disassembled to atomic transformations to allow more detailed view on the transformation process. The composed transformation “CommClass_MethodAssoc” breaks down to following atomic transformations “Class Association”, “Participant Class” and “Class Activity Method”. To be accurate the “Participant Class” transformation is not an atomic transformation included in “CommClass_MethodAssoc”, but it is a transformation which prepares necessary inputs and therefore is necessary predecessor for “CommClass_MethodAssoc”. The relations between these atomic transformations and their connection to the composed transformation are shown in Figure 5.

![Fig. 5. Detailed model of composed transformation](image)
The first step of this transformation is the conversion of the participants which incorporate activities involved in this communication. “Participant_Class” transformation converts BORM’s participant and by doing so prepares the input class for the “CommClass_MethodAssoc”. The formal specification of transformation:

**Name:** Participant_Class

**Inputs & outputs:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>Participant</td>
</tr>
<tr>
<td>Output:</td>
<td>Class</td>
</tr>
</tbody>
</table>

**Condition:**

appropriate(participant)

**Algorithm:**

class = new Class();
class.addPrecedessor(participant)
Model.add(class);

Once both participants are converted to input classes of the “CommClass_MethodAssoc” the transformation process can be executed. The communication, besides the participants, also identifies activities involved in it – from activity and to activity. The formal specification of atomic transformation:

**Name:** ClassActivity_Method

**Inputs & outputs:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>Activity</td>
</tr>
<tr>
<td></td>
<td>class</td>
</tr>
<tr>
<td>Output:</td>
<td>Method</td>
</tr>
</tbody>
</table>

**Condition:**

activity belongs to participant, from which is class created
activity.participant == class.getPrecedessor()

**Algorithm:**

method = new Method();
class.addNewMethod(method);
method.addPrecedessor(activity,class)
Model.add(method);

The next atomic transformation of this composed transformation creates association between the prepared classes. Creation of this association is the final step of the
composed transformation. The formal specification of the “Class_Association” transformation:

**Name:** Class_Association

**Inputs & outputs:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>Class</td>
</tr>
<tr>
<td>class2</td>
<td>Class</td>
</tr>
<tr>
<td>Output:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Association</td>
</tr>
</tbody>
</table>

**Condition:**
appropriate means that (typically) we are not able automatically test this condition and user must choose right element.

\[ \text{appropriate(} class1 \text{)} \land \text{appropriate(} class2 \text{)} \]

**Algorithm:**

\[
\text{assoc} = \text{new} \text{ Association}(); \\
\text{assoc}.\text{begin} = \text{class1} ; \text{assoc}.\text{end} = \text{class2} \\
\text{assoc}.\text{addPrecedesor}(\text{class1, class2}) \\
\text{Model}.\text{add}(\text{assoc});
\]

All of the necessary atomic transformations used in the composed transformation of the communication are defined at this point. The specification of composed transformation is:

**Name:** CommClass_MethodAssoc

**Inputs & outputs:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Concept</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>class1</td>
<td>Class</td>
<td></td>
</tr>
<tr>
<td>class2</td>
<td>Class</td>
<td></td>
</tr>
<tr>
<td>Comm.</td>
<td>Communication</td>
<td></td>
</tr>
<tr>
<td>Output:</td>
<td>Method</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Assoc</td>
<td></td>
</tr>
</tbody>
</table>

**Condition:**

activity belongs to patricipant, from which is class created

\[ \text{comm}.\text{from.participant} == \text{class1}.\text{getPrecedessor()} \land \text{comm}.\text{to.participant} == \text{class2}.\text{getPrecedessor()} \]

**Algorithm:**

\[
\text{ClassActivity}.\text{Method}(\text{comm}.\text{from} , \text{class1} ) \\
\text{ClassActivity}.\text{Method}(\text{comm}.\text{to} , \text{class2} ) \\
\text{Class_Association}(\text{class1},\text{class2});
\]
7. Conclusions

The paper presents fundamental achievements of our research focused on transformations between various models. Specifically, we presented the transformation from the BORM model into the UML class model along with the method of gradual transformations. Models created according to this method bring some advantages as improved audit trail of the model creation, identifying the entry requirements for every item, localisation of change impacts to the model or support for automatic generation of requirements traceability matrix.

The transformation enables analysts and developers to utilise specific methods and notations that fit their problem domain, while maintaining a defined relation between the model elements. This approach addresses one of today's most painful software engineering gaps: the gap between the business analysis and technical implementation.

Our research shows that the transformation is quite a complex task and it will be probably not possible to specify simple and straightforward rules for all the possible situations. However, some kind of heuristics and methodological rules along with the semi-automated algorithm for the transformation represents a contribution to this research area that is currently missing.

Acknowledgement

The authors would like to acknowledge the support of the research project No. 201011130035 of Internal Grant Agency, Faculty of Economics and Management, Czech University of Life Sciences Prague.

References


