Model Transformation in Practice Using the BOC Model Transformer

Marion Murzek[1,*], Gerti Kappel[2], Gerhard Kramler[2]

[1] Women's Postgraduate College for Internet Technologies (WIT)
Vienna University of Technology, Austria
murzek@wit.tuwien.ac.at

[2] Business Informatics Group (BIG)
Institute for Software Technology and Interactive Systems
Vienna University of Technology, Austria
{gerti, kramler}@big.tuwien.ac.at

Abstract. Not least due to the widespread use of meta modelling concepts, model transformation techniques have reached a certain level of maturity [5]. Nevertheless, there is a lack of common community knowledge concerning different transformation approaches fitting different requirements and application areas. The aim of this paper is to present the BOC model transformer, an industry-proven transformation tool. It has been designed for transforming business models in the first place, but scales up nicely for other domains alike.

1 Introduction

At the moment, there is a lot of ongoing research concerning model transformations. The focus of most transformation approaches lies in the area of software development, i.e., automatically generating software out of modelling artefacts. This includes direct model-to-code transformations and transformations via several modelling levels as proposed by the OMG community with the Model Driven Architecture (MDA) [14] and the Meta Object Facility (MOF) [15].
Another area where applications and models have to be transformed and integrated is the area of business modelling. Enterprise Application Integration (EAI) approaches already provide technical solutions, for example, to integrate workflows and heterogeneous parts of enterprise applications [7]. They fall short, however, on integration techniques at the modelling level. To integrate and transform models on the business modelling level, the Enterprise Model Integration (EMI) approach has been proposed [9]. Beside others, this approach includes model transformation concepts for model exchange and model reuse. Based on these concepts and industrial customer needs the BOC Model Transformer (BMT) has been developed. This transformation tool aims

[*] This research has been partly funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.

at supporting transformation of models between different business modelling languages. The approach behind the BMT is very generic, thus, not only standardised modelling languages like BPML/BPMN [3, 4] and UEML [17] are supported, but also customer defined modelling languages. The BMT is based on an own meta-meta model developed within ADONIS® called meta^2 model [8]. The meta^2 model allows the representation of any kind of meta-model thus making it possible to use the BMT also for model transformations in other areas than business modelling. To demonstrate this claim is the main goal of this paper.
To compare existing model transformation approaches the organisers of the workshop "Model Transformation in Practice", co-located with MODELS'2005, have chosen as mandatory running example the "Class to RDBMS" transformation. Taking this as starting point, the *next section* describes the BMT in terms of its architecture, and rule file design. *Chapter 3* presents both the solution of the required aspects of the mandatory example, and the solution to an optional example. *Chapters 4* and *5* discuss lessons learned and future developments. The *appendix* comprises the EBNF grammar of the BMT transformation language, pseudo code for parts of the mandatory example, and the complete transformation code of the optional example.

2 A Short Tour on the BOC Model Transformer

The BMT focuses on the level of model-to-model transformations, in particular, on business model transformations at the CIM level [14]. A particular challenge at the CIM level is the transformation of the widely varying concepts present in different business modelling languages. In the following, the application environment of the BMT, its architecture, and its rule file are described [10, 13].

2.1 ADONIS® - The application environment of the BMT

The development of the BOC Model Transformer (BMT) started in 2003 and still continues. BMT is a product of the BOC Information Systems GmbH, which is a world-wide operating consulting and software house specialised in Strategy, Business Process, and IT Management [1]. The BMT is based on ADONIS®, a business process management toolkit, which is the main product of the company [2]. ADONIS® is designed as meta-modelling platform facilitating an easy creation of new business modelling languages based on the internal meta^2 model. The support of many different modelling languages within the same tool has called for the support of transformations between these modelling languages alike, the starting point of work on the BMT.

2.2 Architecture of the BMT

The core element of the BMT is the rule file. It contains all instructions to fulfil the requirements for transforming models. The source and target models are available in form of XML files conforming to the structure of the ADONIS® meta^2 model repre-

sented in the Data Type Definition (DTD) file. Figure 1 shows the interactions taking place during model transformation.

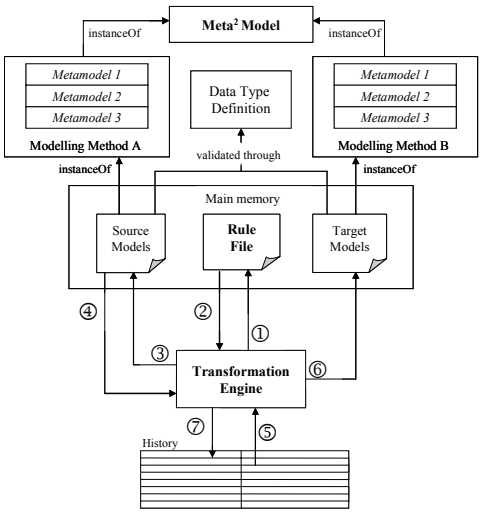

Fig. 1. Architecture of the BMT

1. The Transformation Engine (TE) reads a part of the rule file.
2. The information within the matched tag is being processed in the TE.
3. If it is a navigation tag, the TE searches through the source models until it finds one of the demanded elements. In all other case (namevaluemap and rulescontainer tag) the TE stores this information in program variables and containers, respectively.
4. The selected elements will now be transferred to the TE. If no elements match step 5 to 7 will be skipped and the process continues with step 1.
5. If elements have already been created, it is necessary to check for existing references in the history.
6. Now the TE navigates to the predetermined position and creates the elements as instructed in the rule.
7. In the last step the newly created target elements and their references to the source elements have to be registered in the history.

2.3 Design of the Rule File of the BMT

This section describes the transformer elements used in the rule file. There are six main elements, namely conditions, definitions, functions, navigations, rules, and post-processor actions.

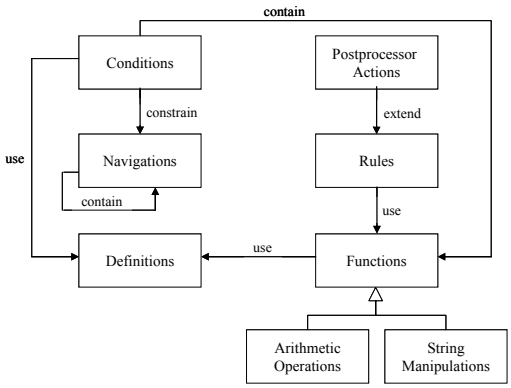

Fig. 2. Design of the BMT rule file

Conditions are used to constrain the selection of elements within navigations. They may contain functions and reuse definitions.
Definitions are used to save information about recurrent elements, using variables annotated with a unique name. In the course of transformation this information can be recalled with the unique name given to the variable. Definitions are used by conditions and functions.
Functions are used to manipulate strings or numbers. Moreover, there are functions converting time, shift or invert bendpoints (used for graphical alignment of relationships). Functions also reuse definitions and are used by rules.
Navigations offer the possibility to navigate within the source and target models. Generally, the position in the source models is influenced by the navigations in the rule file and the position in the target models is implicitly controlled by the creation of target objects. The navigations are constrained by conditions, and may contain further navigation elements.
Rules describe where and how elements should be created. There are two sorts of rules, one for copying and one for creating elements. Rules use functions, for example, to manipulate element values.
Postprocessor Actions are performed at the end of the transformation. These actions include sorting of attribute values, arranging model elements, and shifting model elements due to changes in size of other elements in the same model. Postprocessor actions extend rules.
The EBNF grammar of the rule file is described in Appendix A. Source code listing 1 illustrates the creation of a target table from a persistent source class extracted from the rule file of the mandatory running example.

Source code listing 1:

```
…
<CLASS name="CLASS">
 <!-- checks if class is persistent-->
 <IF>
  <COMPARE>
   <LEFT-VALUE>
```

```
   <ATTRIBUTE name="is_persistent"><select-value/></ATTRIBUTE>
  </LEFT-VALUE>
  <CONDITION>equals</CONDITION>
  <RIGHT-VALUE>true</RIGHT-VALUE>
 </COMPARE>
 …
 <THEN>
  <RULE type="create-instance">
   <PARAM name="Instanceclass">TABLE</PARAM>
  </RULE>
  ….
 </THEN>
</IF>
</CLASS>
```

2.4 Classifying the BMT

In [5] design features yielding a categorisation of model transformation approaches are identified. According to these features the BMT can be characterized as follows:

- *transformation rules* are implicitly structured in LHS/RHS (source/target). The parameter list of a rule describes both the LHS and the RHS whereas the RHS in detail is described via the parameter values.
- *untyped variables* hold source values and intermediate elements.
- *patterns* are used to match model fragments. They use the syntax of the source/target language.
- *parameterization* of navigation offers additional control.
- the *relationship between source and target* is unidirectional. Only new target models can be created.
- the creation of the output models is *target-driven*. This means that target objects are written sequentially, whereas source objects are read on demand.
- the *rule application strategy* is deterministic and partly interactive (user-queries).
- *rule scheduling* is explicit allowing the user to influence the order of rule execution.
- *rule iteration* is implicit by the use of navigation elements.
- the *organization* of the rule file is source-oriented.
- *traceability links* between the source and target elements are created and used during runtime.

The elements introduced in the previous subsection are used to describe the rule file in an XSLT-like language. One may ask why we didn't use XSLT [19], the standardised language to transform XML documents [18]. The main problem of using XSLT concerning our needs is that it does not support global variables and that the output must be written at once. This implies that XSLT does not provide possibilities to buffer information during the transformation process. Another disadvantage of XSLT is its poor scalability. All three shortcomings have been tackled by the BMT by using the Transformation Engine (TE) and a history mechanism (see also Fig. 1), as well as C++ as implementation language.

Given the classification in [5], the BMT doesn't unambiguously fit in one or the other category.

3 Case Study

In this section, the solutions for the mandatory example and for one optional example using the BMT are described. The optional example chosen is the conversion of Roman numbers to Arabic numbers (cf. subsection 3.7). Subsections 3.1 to 3.6 deal with the mandatory example.

3.1 Precursor work

The first step was to create the meta-models in ADONIS® as two new so-called model types named "Class diagram" and "RDBMS diagram". Then, the rule file with the transformation algorithm described in the following was implemented. The basic idea of the implemented algorithm is as follows:

"Navigate to each model in the source models and create a corresponding target model. Within the source model navigate to each persistent source class of the source model and create a corresponding target table within the target model. Then navigate to each source attribute within the source class and create a corresponding target column in the target table. Then all source attributes which must additionally be added to the target table columns must be acquired. This happens according to the specification of the given example: Get all primary attributes of persistent classes and all attributes of non-persistent classes which are connected via an association to the source class or are referenced by a non-primitive source attribute type. This must be repeated recursively for each found source class until all target columns for a distinct target table are calculated. Also source class hierarchies must be considered adequately."

This kind of execution leads to a processing path consisting of different source classes (persistent or non-persistent) connected via associations, generalisations or non-primitive types. This path could be branched, for example if a source class is source of more than one association etc.
As the BMT only supports the concept of global variables, the problems arising through the requirements and the necessary recursive function calls are solved as follows:

- To build the composed names of the target columns the variable *attr_name* of type String is used. It contains the actual name for recursively calculated attributes. Names (of associations or classes) must be appended and removed according to the branches in the processing path.
- To create foreign keys always the last persistent class in a processing path must be remembered. Therefore the variable *l_pers_c* of type String is used. It contains the ids of all persistent classes in the actual processing path. When creating a foreign key, always the last entry of the variable is taken. To consider the classes in dif-

ferent branches in the processing path, the ids must be appended and removed properly.
- For counting the persistent and non-persistent classes in a processing path the variables *p_c* of type Integer and *n_p_c* of type Integer are used. These two variables must also be increased or decreased according to different branches in the processing path.

Due to the differences in including primary attributes the variable *consider_pkey* of type Boolean indicates whether the value of is-primary should be considered or not.
Further variables used are *call_type* of type String used to differ between a call from the function "get-attributes-from-associations" and "get-attributes-following-non-primitive-class-types", and *primary_attr* of type boolean to indicate whether the source attribute in the actual class of the processing path is primary or not, in case of a non-primitive attribute type.

The rule file consists of the following parts:

1. Main program
2. Function "Get attributes from sub-classes"
3. Function "Get attributes from outgoing associations"
4. Function "Get attributes following non-primitive attribute types"
5. Sub-function "Calculate attributes"

In the following, each of the five parts will be described in detail. First, the algorithm is described, and secondly the implementation is listed in pseudo code. Representation in form of pseudo code was chosen because it is not as long as and easier readable than the rule file itself.

3.2 Main program

The main program creates a target model of type "RDBMS diagram" for each source model of type "Class diagram". Then it creates one target table for each source class depending if the class is persistent. From the transformed source classes all primitive attributes are transformed to target columns and for all non-primitive attributes the function "Get-attributes-following-non-primitive-attribute-types" is called. After that the functions "Get-attributes-from-outgoing-associations" and "Get-attributes-from-sub-classes" are called to get all source attributes for the transformed target table recursively.

```
(1) For each source model SM with model type "Class dia-
    gram"
      a. Create target model TM with model type "RDBMS
         diagram"
      b. For each source class SC in SM
            i.  If SC is persistent
               1. Create target table TT with
                  TT.name = SC.name
```

2. For each source attribute SA of SC
   a. If SA.kind_of_type is primitive
      Create target column TC in TT with:
      TC.name = SA.name,
      TC.type = SA.type,
      TC.pkey = SA.is_primary,
      TC.fkey = “false”
   b. Else (type is non-primitive)
      i. Set *attr_name* = “ ”, *p_c* = 0, *n_p_c* = 0, *l_pers_c* = “ ”
      ii. CALL “Get-all-attributes-following-non-primitive attribute types”
3. Set *attr_name* = “ ”, *l_pers_c* = “ ”, *p_c* = 0, *n_p_c* = 0
4. **CALL “Get-attributes-from-outgoing-associations”**
5. Set *attr_name* = “ ”, *l_pers_c* = “ ”, *p_c* = 0, *n_p_c* = 0
6. **CALL “Get-attributes-from-sub-classes”**

(2) END

3.3 Algorithm “Get attributes from sub-classes”

The algorithm “Get attributes from sub-classes” collects recursively the attributes from classes which are connected via incoming generalisations. Additionally the attributes from classes connected via outgoing associations and classes which are referenced by non-primitive types are collected.

Assumption: Sub-classes are considered to be non-persistent and do not add new primary keys to the super-class, also classes which are connected to subclasses via outgoing associations do not add primary keys – this is indicated by the variable *consider_pkey.*

Algorithm “Get-attributes-from-sub-classes”

Used variables:

l_pers_c, attr_name, p_c, n_p_c

Position in source model: class SC

Position in target model: table TT

(1) Follow each incoming generalisation from SC to source class SC_1 //is subclass = is non-persistent

i.*attr_name*.add_entry(SC_1.name)

ii. For each source attribute SA of SC_1

1. If SA.type is non-primitive

a. Set *n_p_c*++, *consider_pkey* = "false",

b. **Call "Get attributes following non-primitive attribute types"**

c. Set *n_p_c*--, *consider_pkey* = true

d. delete_last_entry(*attr_name*)

2. Else // SA.type is primitive

a. Create TC in TT with: TC.name = *attr_name* + "_" + SA.name, TC.type = SA.type, TC.pkey = "false", TC.fkey = "false"

// subclasses, assoc of subclasses and non-primitive class types of subclasses or assoc of subclasses do not add further pkeys.

iii. If SC_1 has outgoing associations

1. Set *n_p_c*++, *consider_pkey* = "false"

2. **CALL "Get attributes from outgoing assocs"**

3. Set *n_p_c*--, *consider_pkey* = "true"

4. delete_last_entry(*attr_name*)

iv. If SC_1 has incoming generals

1. **CALL "Get attributes from subclasses"**

2. delete_last_entry(*attr_name*)

v. delete_last_entry(*attr_name*)

(2) END

3.4 Algorithm "Get attributes from outgoing associations"

The algorithm "Get attributes from outgoing assoc" collects recursively the attributes from classes which are connected via outgoing associations to the current class. If outgoing associations are found, the function "calculate-attributes" is called.

Algorithm "Get attributes from outgoing assocs"

Position in source model: class SC

Position in target model: table TT

(1) Follow each outgoing association assoc from SC to SC_1

a. *attr_name*.add_entry(assoc.name)

b. Set *call-type* = "assoc"

c. CALL "calculate-attributes"

d. Set *call-type* = " "

(2) END

3.5 Algorithm "Get attributes following non-primitive attribute types"

The algorithm "Get attributes following non-primitive attribute types" is similar to the previous function "Get attributes from outgoing assoc". Instead of following outgoing associations the algorithm follows the non-primitive type of attributes to the referenced class. For each found class the function "calculate-attributes" is called.

Algorithm "Get attributes following non-primitive attribute types"

Position in source model: attribute SA

Position in target model: column TC

(1) Set *primary_attr* = value of SA.is_primary

(2) *attr_name*.add_entry(SA.class.name)

(3) Navigate to class SC_1 referenced by SA.type

a. *call-type* = "nonp"

b. CALL "calculate-attributes"

c. *call-type* = " "

(4) END

3.6 Algorithm "Calculate attributes"

The algorithm "Calculate attributes" creates target columns in a target table depending on the combination of the following facts:

- whether the actual source class is persistent or non-persistent, and
- which kind of source classes (persistent or non-persistent) occurred in the processing path so far, and
- if the actual source class is a sub-class,
- if the actual source class is a top-most class, or

- if the actual source class is not involved in a hierarchy.

In the algorithm “Calculate attributes” we primarily distinguish between “actual class is persistent” and “actual class is non-persistent”. Depending on the persistence status and the occurrence of the actual class in the sequence of classes, the attributes are transformed differently.

If the found class is *persistent*, there are three different cases how the attributes get transformed and if further attributes must be collected or not:

1. it is the first class of this sequence.
2. one or more previous non-persistent classes have been found in this sequence.
3. one previous persistent class has been found in this sequence.

These three cases are checked using the variables *p_c* and *n_p_c*.

If the found class is *non-persistent*, two different cases must be considered:

1. no previous persistent class was found in this sequence.
2. one previous persistent class has been found in this sequence.

Within these two different cases further distinctions must be made. First, because non-persistent classes may be subclasses, and secondly, because non-persistent classes may require that further foreign key attributes have to be obtained from following persistent classes. This leads to three different cases. The found class is:

1. a sub-class.
2. a top-most class.
3. not involved in a hierarchy.

In each case, attributes are transformed differently, and further attributes from following classes are calculated differently as well.

The transformation of attributes referenced by non-primitive attribute types is slightly different to that of outgoing associations. This is controlled by the variable *primary_attr* and *call_type*.

Within the algorithm “Calculate attributes”, the creation of foreign keys is a good example to show the mechanism for keeping relations “clean” and how the history of the BMT is used.

In the rule file the *creation of a foreign key* (fkey) works as follows. The position in the source model must be at a source class. The rule “create-interref” with the parameter “kind of the interref” (possible values are: object- or modelreference) is called. If the target table corresponding to the actual source class has already been created, the reference (interref) points to this target table. Otherwise the reference (interref) points to the actual source class and an entry in the history is created. At the end of the transformation, when the history is processed, the corresponding target table will be determined and the reference (interref) will be updated.

Due to the length of the pseudo code for the calculation of attributes, it is listed in Appendix B.

3.7 Convert Roman numbers to Arabic numbers

Since business model transformations require experience with string and arithmetic expressions, the optional example “Converting Roman numbers to Arabic numbers” was chosen.

The algorithm converts each character of the roman number into the according value and adds or subtracts the value from the running result – depending on the next character. If the next character is lower or equal the value gets added, in the other case it gets subtracted. In the following, the algorithm is described in pseudo code. The original rule file containing this algorithm is included in Appendix C.

Used variables:
rn … roman number, an … arabic number,
c_c … character-counter, rn_l … length of roman number
c … character of roman number, n … number
f_c … first character, s_c … second character,
f_n … first number, s_n … second number

Main program:

```
(1) Get rn from command-line
(2) Set c_c = 0, an = 0
(3) Set rn_l = length_of(rn)
(4) CALL "Calculate-arabic-number"
```

Function "Convert character to number"

```
(1) Case c is
    a)  I then Set n = 1
    b)  V then Set n = 5
    c)  X then Set n = 10
    d)  L then Set n = 50
    e)  C then Set n = 100
    f)  D then Set n = 500
    g)  M then Set n = 1000
    h)  default Set n = 0 // used for special case, if
    the roman number consists of only one character
(2) END
```

Function "Calculate-arabic-number"

```
(1)  Set f_c = get_character(rn, c_c)
(2)  Set s_c = get_character(rn, c_c+1)
(3)  Set c = f_c
(4)  Call "Convert character to number"
```

```
(5)  Set f_n = n
(6)  Set c = s_c
(7)  Call “Convert character to number”
(8)  Set s_n = n
(9)  If f_n >= s_n
   a)  Set an = an + f_n
(10) Else
   a)  Set an = an – f_n
(11) Set c_c++
(12) If c_c = = rn_length
   a)  Put an on command-line
   b)  End
(13) Else
   a)  CALL “Calculate-arabic-number”
(14) END
```

4 Lessons Learned

Due to the fact, that the BMT is optimised for transforming models in the area of business modelling, where mainly dynamic (process flow) models are transformed, it has been very interesting to solve an example of another domain, where the models are mainly of static nature. This section deals with positive, negative and general observations, requirements specific for transformations in the area of business modelling, and experiences from industrial projects.

4.1 Positive observations

The *expressive power* of the BMT is satisfying, since the rule file language offers nearly all concepts of a common programming language. It has been possible to solve the mandatory example although it belongs to a complete different application domain. The *creation of foreign keys* is supported very well using the history and the post-processing concepts of the BMT. Furthermore, we have observed that the *recursive calculation* used for collecting attributes has been no problem.

4.2 Negative observations

As the BMT language is based on XML, the rule file for the "class2RDBMS" transformation consists of *many lines of code*, approximately 30-35 pages (variation due to comments). Due to the fact that the target models are created in one step and that it is not possible to modify already created target elements, it has been quite *difficult to develop the algorithm*. It has also been observed, that the *attributes* for classes are *calculated more than once*. An optimised solution would intermediately store calculated output information for reuse, e.g., collected columns for a table, which is not possible in the current version of the BMT. Checking the *validity of* the XML-based *input models* is currently also not supported. In addition, the *missing concept of local variables* makes recursive calculations difficult in the BMT.

4.3 General observations

The *debugging mechanisms* of the BMT are sufficient but not substantial. Debugging is supported by the XSLT-processor Xalan regarding the well-formedness of the rule file. Additionally, the developer of the rule file is supported by a rule named "debug-output", which may be used to check the position in the source and target models at a distinct point in the rule file. Moreover the text tag may be used for debugging.
The *logfile* provided by the BMT reports all nodes of the source document which have not been taken into consideration during the transformation. Objects which have not been transformed directly but have been involved in a transformation are also considered.
Bi-directionality is not supported by the BMT. If the models have to be transformed in either direction, a second rule file has to be written.
The models to be transformed with the BMT must be in the ADONIS® XML-file format. The BMT itself is not able to deal with other *file formats.* All formats, however, which may be read by ADONIS® including XMI [16], may be also converted into the ADONIS® XML-file format.

4.4 Requirements for business model transformations

Due to the fact that the BMT has been designed for transforming business models in the first place, many of its features have not been used for solving the running example. This leads to the assumption, that transformations of models in the area of business modelling must partly satisfy other requirements than in the area of software modelling and code generation. Based on our experience (see also next subsection), the following concepts are important when transforming business models:

1. String and arithmetic operations, e.g., character conversions like ä to ae, rtf2text and text2rtf conversions, and rounding numbers.
2. Operations for conversions of time and date formats.
3. Concepts for renaming, splitting and merging model elements (models, classes, attributes, etc.), and the resulting impact on references and relationships.
4. Version management.

5. Creation of new objects and connecting them to existing (transformed) objects.
6. Handling diagrammatic information of both existing and newly created objects.
7. Copying objects more than once (MODE concept).
8. Deleting objects without information loss – hiding information.

4.5 Industrial experiences with the BMT

In the following, we report on two industrial projects using the BMT, and the experiences gained so far. In both cases, proprietary modelling languages have been used.
The aim of the *first project* has been *model interoperability* [10]. The customer demanded a repeated exchange of several models between two different modelling languages, one in German and one in English. It was necessary to write two rule files, one for each direction. The main requirements were string operations to convert the elements of the modelling languages described in German to English and back again. The content didn't get translated. After the rule file had been written, the users got trained on handling the tool (not concerning the rule file) for exchanging the models on demand.
The aim of the *second project* has been *model integration* [11]. A company had three different modelling languages in three different departments in use. Due to a merge of these three departments they wanted to have one new modelling method. The entire stock of models written in the three source modelling languages exceeded the number of thousand. So they wanted to transform the models automatically. In this case, three different rule files were needed.
All in all we observed that in industrial practice one of the *main problems* of model transformation is the specification of *syntactic and semantic correspondences* between proprietary modelling languages. Experiences show that the customers and the consultants need more than one workshop to agree on these correspondences[1]. Therefore our future research focus concerning model transformation lies on the semi-automatic creation of semantic correspondences to support the process of agreeing upon correspondences between modelling languages.
Technical problems have been observed concerning the size of input files, because big companies often have thousands of models. And these files cannot be split easily because of inter-model cross referencing.

5 Outlook

The experience gained by solving the mandatory example has given an additional impulse on the direction of possible further developments of the BMT.
A fundamental decision concerning the future of the BMT is whether to restrict it to the area of business modelling or to generalise its use to other domains as well. This decision depends mainly on the future practical application requirements in the realm of the company which is owner of the BMT.

[1] This may easily be related to the mandatory running example of this workshop. Although the example is very simple, an own FAQ-page has been necessary for clarifications.

In our opinion further developments shall comprise the following:

- Rule file creation should be supported by a *graphical, textual or mixed user interface* to ease the writing of a rule file regarding user skills and complexity. The first step in this direction has been the design of a graphical user interface [12].
- Without doubt, the *concept of local variables* has to be added. This will both ease the use of recursive function calls and make the rule file shorter.
- *Error handling* concerning syntactic and semantic inconsistencies should be added.
- The *semantic correspondences* of the source and target objects should be pointed out (comparable to the ATL Model Weaver [6]). At the moment, they tend to get lost within the rule file code. Further research goes in the direction of supporting a semi-automatic generation of such semantic correspondences, which may be adapted by the user.
- On the long run, the partial derivation of rules for back-transformation taken from the rule file should be supported leading to *bi-directionality*. This will implicitly promote a mechanism for verification of the transformation.

Acknowledgement

We would like to thank BOC Information Technologies GmbH for providing the BMT and the Business Process Modelling Tool ADONIS®, in which the meta models have been configured.

References

1. ADONIS Homepage. www.boc-eu.com, access 10 August 2005.
2. BOC: ADONIS 3.7 - User Manual III: ADONIS Standard Modelling Method. BOC GmbH, May 2003.
3. BPMI.org: Business Process Modeling Language - Specification Version 1.0. http://www.bpmi.org/BPML.htm, access 12 August 2005.
4. BPMI.org: Business Process Modeling Notation - Specification Version 1.0. http://www.bpmi.org/BPMN.htm, access 12 August 2005.
5. K. Czarnecki; S. Helsen: Classification of Model Transformation Approaches. OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture.
6. M. Didonet Del Fabro; J. Bézivin; F. Jouault; E. Breton; G. Gueltas: AMW: a generic model weaver. Proceedings of the 1ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM05).
7. P. Johannesson; B. Wangler; P. Jayaweera: Application and Process Integration –Concepts, Issues, and Research Directions. In: Brinkkemper, S.; Lindencrona, E.; Solvberg, A. (Eds.): Information Systems Engineering Symposium CAiSE 2000, Springer-Verlag, 2000.
8. S. Junginger; H. Kühn; R. Strobl; D. Karagiannis: Ein Geschäftsprozessmanagement-Werkzeug der nächsten Generation – ADONIS: Konzeption und Anwendungen. WIRTSCHAFTSINFORMATIK 42 (2000) 5, pp. 392-401 (in German).
9. H. Kühn; F. Bayer; S. Junginger; D. Karagiannis: Enterprise Model Integration. Bauknecht, K.; Tjoa, A M.; Quirchmayr, G. (eds.): Proceedings of the 4th International Conference

EC-Web 2003, Prague, Czech Republic, September 2003, Springer LNCS 2738, pp. 379-392.
10. H. Kühn; M. Murzek; F. Bayer: Horizontal Business Process Model Interoperability using Model Transformation. INTEREST'2004 Workshop at ECOOP 2004, Oslo, Norway, June 2004.
11. H. Kühn; M. Murzek; F. Bayer: Business Model Interoperability using Enterprise Model Integration. Proceedings of the 14th International Conference on eChallenges, 27-29 October 2004, Vienna, Austria, pp. 233-240.
12. M. Murzek; G. Kramler: Defining Model Transformations for Business Process Models Graphically. Proceedings of the Workshop on "Enterprise Modelling and Ontology: Ingredients for interoperability" at PAKM2004, December 2004, Vienna, Austria.
13. M. Murzek: Methodenübergreifende Modelltransformationen am Beispiel von ADONIS. Diploma Thesis, University of Vienna, April 2004 (in German).
14. Object Management Group: *MDA Guide, Version 1.0.1*, 12. June 2003. http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf, access 30 July 2005.
15. Object Management Group: Meta Object Facility (MOF) 2.0 Core Specification, October 2004. http://www.omg.org/docs/ptc/04-10-15.pdf, access 30 July 2005.
16. Object Management Group: OMG XML Metadata Interchange (XMI) Specification, Version 1.2, January 2002. http://www.omg.org/cgi-bin/doc?formal/02-01-01.pdf, access 10 August 2005.
17. UEML – Unified Enterprise Modelling Language, http://www.ueml.org, access 30 July 2005.
18. World Wide Web Consortium (W3C): Extensible Markup Language (XML) 1.1, Recommendation, November 2003, http://www.w3.org/TR/2003/PR-xml11-20031105/, access 12 August 2005.
19. World Wide Web Consortium (W3C): XSL Transformations (XSLT) Version 1.0, Recommendation, November 1999, http://www.w3.org/TR/xslt, access 12 August 2005.

Appendix

A Grammar of the rule file in EBNF

```
ADOXMLMIGRATE ::= (Definition | Navigation | Rule)*

Definition ::= (NAMEVALUEMAP | LOGMESSAGE |
RULESCONTAINER )*

RULESCONTAINER ::= (Navigation | Rule | NAMEVALUEMAP)*

Navigation ::= NAVIGATE, Navigation-parameter

Navigation-parameter ::= (Parameter-list | Rule |
Definition)*

Rule ::= RULE, Parameter-list

Parameter-list ::= (PARAM, Value)*

Value ::= Literal | Function* | Condition*

Literal ::= Text
```

Function ::= **select-value** | **get-map-value** | **get-attribute-value** | **query-user** | **text** | **newline** | **str-len** | **str-find** | **str-copy** | **str-replace** | **str-concat** | **add** | **subtract** | **multiply** | **divide** | **minus** | **round** | **convert-time** | **count-references**

Condition ::= IF | Navigation

Definition ::= (**NAMEVALUEMAP** | **LOGMESSAGE** | **RULESCONTAINER**)*

RULESCONTAINER ::= (Navigation | Rule | **NAMEVALUEMAP**)*

Navigation ::= **NAVIGATE**, Navigation-parameter

B Pseudo code for the algorithm "Calculate attributes"

```
Position in source model: class SC
Position in target model: table TT
  (1) If source class SC is persistent
      a. If p_c = 0 AND n_p_c > 0
         i. For each primary source attribute pSA of SC
            1. If pSA.type is non-primitive
               a. Set p_c++, l_pers_c.add_entry(SC.id)
               b. Call "Get attributes following non-
                  primitive attribute types"
               c. Set p_c--,
               d. delete_last_entry(attr_name)
               e. delete_last_entry(l_pers_c)
            2. Else (pSA.type is primitive)
               a. Create table column TC in TT:
                  TC.name = attr_name + "_" + pSA.name,
                  TC.type = pSA.type,
                  TC.pkey =
                  if call-type = "nonp" and primary_attr =
                  "true" and consider_pkey != "false"
                    value of pSA.is_primary
                  else
                    "false",
                  TC.fkey = reference to TT created from
                  SC
      b. If p_c = 0 AND n_p_c = 0
         i. For each pSA of SC
            1. If pSA.type is non-primitive
               a. Set p_c++,
```

b. *l_pers_c*.add_entry(SC.id)

c. Call “Get attributes following non-primitive attribute types”

d. Set *p_c*--,

e. delete_last_entry(*attr_name*)

f. delete_last_entry(*l_pers_c*)

2. Else (pSA.type is primitive)

a. Create TC in TT:
TC.name = *attr_name* + ”_” + pSA.name,
TC.type = SA.type,
TC.pkey =
if *call-type* = “nonp” and *primary_attr* = “true”
value of pSA.is_primary
else
“false”,
TC.fkey = reference to TT created from SC

ii. If SC has outgoing associations

1. Set *p_c*++

2. *l_pers_c*.add_entry(SC.id)

3. **CALL “Get attributes from outgoing assocs”**
// no further calls, because SC could not be subclass (persistent) and if SC is top-most, then subclasses would not add pkeys.

4. Set *p_c*--,

5. delete_last_entry(*attr_name*)

6. delete_last_entry(*l_pers_c*)

c. Else (*p_c* != 0)
// not further pkeys or fkeys will be added through outgoing assocs, but through non-primitive types

i. If *call-type* = “nonp”

1. For each pSA of SC

a. If pSA.type is non-primitive

i. Set *p_c*++

ii. *l_pers_c*.add_entry(SC.id)

iii. Call “Get attributes following non-primitive attribute types”

iv. Set *p_c*--

v. delete_last_entry(*attr_name*)

vi. delete_last_entry(*l_pers_c*)

b. Else (pSA.type is primitive)

i. Create TC in TT:
TC.name = *attr_name* + ”_” + pSA.name,
TC.type = pSA.type,
TC.pkey = “false”,
TC.fkey = reference
get_last_entry(*l_pers_c*)

(3) Else (if SC is non-persistent)

a. If *p_c* = 0

i. If SC has outgoing generals
(SC = subclass)

1. Get top-most class SC_1

2. If SC_1 is persistent

a. For each pSA of SC_1

- If pSA.type is non-primitive

1. Set *p_c*++

2. *l_pers_c*.add_entry(SC_1.name)

3. **Call “Get attributes following non-primitive attribute types”**

4. Set *p_c*--

5. delete_last_entry(*attr_name*)

6. delete_last_entry(*l_pers_c*)

- Else (pSA.type is primitive)

1. Create TC in TT:
TC.name = *attr_name* +”_”+ pSA.name,
TC.type = pSA.type,
TC.pkey =
if *call-type* = “nonp” and *primary_attr* = “true” and *consider_pkey* != “false”
value of pSA.is_primary
else
“false”,
TC.fkey = reference TT created from SC_1

b. If SC_1 has outgoing assocs

- Set *p_c*++

- *l_pers_c*.add_entry(SC_1.name)
// further sub-classes don’t add pkeys

- **CALL “Get attributes from outgoing associations”** // looking for further non-persistent classes
- Set *p_c*--,
- delete_last_entry(*attr_name*)
- delete_last_entry(*l_pers_c*)

3. If SC_1 is non-persistent
   a. For each source attribute SA of SC_1
      - If SA.type is non-primitive
        1. Set *n_p_c*++
        2. **Call “Get attributes following non-primitive attribute types”**
        3. Set *n_p_c*–
        4. delete_last_entry(*attr_name*)
      - Else (SA.type is primitive)
        1. Create TC in TT:
           TC.name = *attr_name* + ”_” + SA.name,
           TC.type = SA.type,
           TC.pkey =
           if *consider_pkey* = “false”
             “false”
           else
             value of SA.is_primary
           TC.fkey = “false”
           // p_c = 0 → no previous p-class
   b. If SC_1 has outgoing associations
      - Set *n_p_c*++
      - **CALL “Get attributes from outgoing associations”** // looking for further non-persistent classes
      - Set *n_p_c*--,
      - delete_last_entry(*attr_name*)
   c. If SC_1 has incoming generals
      - Set *n_p_c*++
      - **CALL “Get attributes from sub-classes”**
      - Set *n_p_c*–
      - delete_last_entry(*attr_name*)

ii. Else If SC has incoming generals
(SC = top-most class)

1. For each source attribute SA of SC

a. If SA.type is non-primitive

- Set *n_p_c*++
- **Call "Get attributes following non-primitive attribute types"**
- Set *n_p_c*--
- delete_last_entry(*attr_name*)

b. Else (SA.type is primitive)

- Create TC in TT:
  TC.name = *attr_name* + "_" + SA.name,
  TC.type = SA.type,
  TC.pkey =
  if *consider_pkey* = "false"
    "false"
  else
    value of SA.is_primary
  TC.fkey = "false"
  // p_c = 0 → no previous p-class

2. If SC_1 has outgoing assoc

a. Set *n_p_c*++

b. **CALL "Get attributes from outgoing associations"** // looking for further non-persistent classes

c. Set *n_p_c*--,

d. delete_last_entry(*attr_name*)

3. If SC1 has incoming generals

a. Set *n_p_c*++

b. CALL "Get attributes from sub-classes"

c. Set *n_p_c*--

d. delete_last_entry(*attr_name*)

iii. Else (SC is not involved in a hierarchy)

1. For each source attributes SA of SC

a. If SA.type is non-primitive

- Set *n_p_c*++
- **Call "Get attributes following non-primitive attribute types"**
- Set *n_p_c*--

- delete_last_entry(*attr_name*)

b. Else (SA.type is primitive)

- Create TC in TT:
  TC.name = *attr_name* + "_" + SA.name,
  TC.type = SA.type,
  TC.pkey =
  if *consider_pkey* = "false"
    "false"
  else
    value of SA.is_primary
  TC.fkey = "false"
  // p_c = 0 → no previous p-class

2. If SC_1 has outgoing assoc

a. Set *n_p_c*++

b. **CALL "Get attributes from outgoing associations"** // looking for further non-persistent classes

c. Set *n_p_c*--,

d. delete_last_entry(*attr_name*)

b. Else (*p_c* != 0) // previous p-class exists

i. If SC has outgoing generals
(SC = subclass)

1. Get top-most class SC_1

a. If SC_1 is persistent
// not further pkeys or fkeys will be added through outgoing assocs, but through non-primitive types

- If *call-type* = "nonp"

1. For each primary source attribute pSA of SC_1

a. If pSA.type is non-primitive

i. Set *p_c*++

ii. l_pers_c.add_entry(SC_1)

iii. **Call "Get attributes following non-primitive attribute types"**

iv. Set *p_c*--

v. delete_last_entry(*attr_name*)

vi. delete_last_entry(*l_pers_c*)

b. Else (pSA.type is primitive)

i. Create TC in TT:
TC.name = *attr_name* + "_" + pSA.name,
TC.type = pSA.type,
TC.pkey = "false",
TC.fkey = reference get_last_entry(*l_pers_c*)

b. If SC_1 is non-persistent

- For each primary source attribute pSA of SC_1

1. If pSA.type is non-primitive

a. Set *n_p_c*++

b. Call "Get attributes following non-primitive attribute types"

c. Set *n_p_c*--

d. delete_last_entry(*attr_name*)

2. Else (pSA.type is primitive)

a. Create TC in TT:
TC.name = *attr_name* + "_" + pSA.name,
TC.type = pSA.type,
TC.pkey = "false"
TC.fkey = reference get_last_entry(*l_pers_c*)

- If SC_1 has outgoing associations

1. Set *n_p_c*++

// subclasses do not add primary keys

2. CALL "Get attributes from outgoing assocs"

3. Set *n_p_c*--

4. delete_last_entry(*attr_name*)

ii. Else //SC = top-most class or not involved in a hierarchy

1. For each primary source attribute pSA of SC

a. If pSA.type is non-primitive

- Set *n_p_c*++

- **Call "Get attributes following non-primitive attribute types"**

- Set *n_p_c*--

```
      - delete_last_entry(attr_name)
    b. Else (pSA.type is primitive)
      - Create TC in TT:
        TC.name = attr_name + "_" +
        pSA.name,
        TC.type = pSA.type,
        TC.pkey = "false"
        TC.fkey = reference
        get_last_entry(l_pers_c)
  2. If SC1 has outgoing assoc
    a. Set n_p_c++
       // subclasses do not add primary keys
    b. CALL "Get attributes from outgoing
       assocs"
    c. Set n_p_c--
    d. delete_last_entry(attr_name)
(2) End
```

C Rule file of the algorithm "Convert Roman number to Arabic number"

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<ADOXMLMIGRATE>

 <!-- Algorithm for converting a character to a value -->
 <RULESCONTAINER name="char2value">
  <IF>
   <COMPARE>
    <LEFT-VALUE>
     <get-map-value>character</get-map-value>
    </LEFT-VALUE>
    <CONDITION>equals</CONDITION>
    <RIGHT-VALUE>I</RIGHT-VALUE>
   </COMPARE>
  <THEN>
   <NAMEVALUEMAP>
    <ELEM name="number">1</ELEM>
   </NAMEVALUEMAP>
  </THEN>
  </IF>
  <IF>
   <COMPARE>
    <LEFT-VALUE>
```

```
      <get-map-value>character</get-map-value>
    </LEFT-VALUE>
    <CONDITION>equals</CONDITION>
    <RIGHT-VALUE>V</RIGHT-VALUE>
  </COMPARE>
<THEN>
  <NAMEVALUEMAP>
    <ELEM name="number">5</ELEM>
  </NAMEVALUEMAP>
</THEN>
</IF>
<IF>
  <COMPARE>
    <LEFT-VALUE>
      <get-map-value>character</get-map-value>
    </LEFT-VALUE>
    <CONDITION>equals</CONDITION>
    <RIGHT-VALUE>X</RIGHT-VALUE>
  </COMPARE>
<THEN>
  <NAMEVALUEMAP>
    <ELEM name="number">10</ELEM>
  </NAMEVALUEMAP>
</THEN>
</IF>
<IF>
  <COMPARE>
    <LEFT-VALUE>
      <get-map-value>character</get-map-value>
    </LEFT-VALUE>
    <CONDITION>equals</CONDITION>
    <RIGHT-VALUE>L</RIGHT-VALUE>
  </COMPARE>
<THEN>
  <NAMEVALUEMAP>
    <ELEM name="number">50</ELEM>
  </NAMEVALUEMAP>
</THEN>
</IF>
<IF>
  <COMPARE>
    <LEFT-VALUE>
      <get-map-value>character</get-map-value>
    </LEFT-VALUE>
    <CONDITION>equals</CONDITION>
    <RIGHT-VALUE>C</RIGHT-VALUE>
  </COMPARE>
```

```
 <THEN>
  <NAMEVALUEMAP>
   <ELEM name="number">100</ELEM>
  </NAMEVALUEMAP>
 </THEN>
 </IF>
 <IF>
  <COMPARE>
   <LEFT-VALUE>
    <get-map-value>character</get-map-value>
   </LEFT-VALUE>
   <CONDITION>equals</CONDITION>
   <RIGHT-VALUE>D</RIGHT-VALUE>
  </COMPARE>
 <THEN>
  <NAMEVALUEMAP>
   <ELEM name="number">500</ELEM>
  </NAMEVALUEMAP>
 </THEN>
 </IF>
 <IF>
  <COMPARE>
   <LEFT-VALUE>
    <get-map-value>character</get-map-value>
   </LEFT-VALUE>
   <CONDITION>equals</CONDITION>
   <RIGHT-VALUE>L</RIGHT-VALUE>
  </COMPARE>
 <THEN>
  <NAMEVALUEMAP>
   <ELEM name="number">1000</ELEM>
  </NAMEVALUEMAP>
 </THEN>
 </IF>
</RULESCONTAINER>

<!-- Algorithm for calculating the arabic number out of the roman number -->
<RULESCONTAINER name="calculate-arabic-number">
 <NAMEVALUEMAP>
  <ELEM name="first-character">
   <str-copy>
    <param name="string">
     <get-map-value>get-roman-number</get-map-value>
    </param>
    <param name="from">
     <get-map-value>char-counter</get-map-value>
    </param>
```

```
      <param name="count">1</param>
    </str-copy>
  </ELEM>
  <ELEM name="second-character">
    <str-copy>
      <param name="string">
        <get-map-value>get-roman-number</get-map-value>
      </param>
      <param name="from">
        <add>
          <param name="value">
            <get-map-value>char-counter</get-map-value>
          </param>
          <param name="value">1</param>
        </add>
      </param>
      <param name="count">1</param>
    </str-copy>
  </ELEM>
</NAMEVALUEMAP>

<!-- set input-parameter to first-character and call function for converting the char-
acters into number-values -->
<NAMEVALUEMAP>
  <ELEM name="character">
    <get-map-value>first-character</get-map-value>
  </ELEM>
</NAMEVALUEMAP>

<CALL name="char2value"/>

<!-- save "return-value" -->
<NAMEVALUEMAP>
  <ELEM name="first-number">
    <get-map-value>number</get-map-value>
  </ELEM>
</NAMEVALUEMAP>

<!-- set input-parameter to second-character and call function for converting the
characters into number-values -->
<NAMEVALUEMAP>
  <ELEM name="character">
    <get-map-value>second-character</get-map-value>
  </ELEM>
</NAMEVALUEMAP>
<CALL name="char2value"/>
```

```
<!-- save "return-value" -->
<NAMEVALUEMAP>
  <ELEM name="second-number">
    <get-map-value>number</get-map-value>
  </ELEM>
</NAMEVALUEMAP>

<!-- compare first with second and add/subtract to/from result-->
<IF>
  <COMPARE>
    <LEFT-VALUE>
      <get-map-value>first-number</get-map-value>
    </LEFT-VALUE>
    <CONDITION>greater-equals</CONDITION>
    <RIGHT-VALUE>
      <get-map-value>second-number</get-map-value>
    </RIGHT-VALUE>
  </COMPARE>
<THEN>
  <NAMEVALUEMAP>
    <ELEM name="result">
      <add>
        <param name="value">
          <get-map-value>result</get-map-value>
        </param>
        <param name="value">
          <get-map-value>first-number</get-map-value>
        </param>
      </add>
    </ELEM>
  </NAMEVALUEMAP>
</THEN>
<ELSE>
  <NAMEVALUEMAP>
    <ELEM name="result">
      <subtract>
        <param name="value">
          <get-map-value>result</get-map-value>
        </param>
        <param name="value">
          <get-map-value>first-number</get-map-value>
        </param>
      </subtract>
    </ELEM>
  </NAMEVALUEMAP>
</ELSE>
</IF>
```

```
  <!-- increase counter to get the next two characters -->
  <NAMEVALUEMAP>
   <ELEM name="char-counter">
    <add>
     <param name="value">
      <get-map-value>char-counter</get-map-value>
     </param>
     <param name="value">1</param>
    </add>
   </ELEM>
  </NAMEVALUEMAP>

  <!-- did we reach the last character? - end or process -->
  <IF>
   <COMPARE>
    <LEFT-VALUE>
     <get-map-value>char-counter</get-map-value>
    </LEFT-VALUE>
    <CONDITION>equals</CONDITION>
    <RIGHT-VALUE>
     <get-map-value>roman-number-len</get-map-value>
    </RIGHT-VALUE>
   </COMPARE>
  <THEN>

   <LOGMESSAGE>
    <newline/>
    <text>Arabic number: </text>
    <get-map-value>result</get-map-value>
   </LOGMESSAGE>

  </THEN>
  <ELSE>
   <CALL name="calculate-arabic-number"/>
  </ELSE>
  </IF>
 </RULESCONTAINER>

 <!-- Main program - gets Roman number from command line and calls function
calculate Arabic number -->
 <MODELS>
  <MODELTYPE>
   <NAMEVALUEMAP>
    <ELEM name="get-roman-number">
     <query-user>
      <param name="message">
```

```
          Insert a valid roman number:
        </param>
      </query-user>
    </ELEM>
    <ELEM name="char-counter">0</ELEM>
    <ELEM name="arabic-number">0</ELEM>
    <ELEM name="roman-number-len">
      <str-len>
        <get-map-value>get-roman-number</get-map-value>
      </str-len>
    </ELEM>
    <ELEM name="result">0</ELEM>
  </NAMEVALUEMAP>

  <CALL name="calculate-arabic-number"/>

 </MODELTYPE>
</MODELS>
</ADOXMLMIGRATE>
```