Position Paper for Workshop "Enterprise Modelling and Ontology: Ingredients for interoperability" at PAKM 2004, Vienna, 2 December 2004

Defining Model Transformations for Business Process Models Graphically*

Marion Murzek[1], Gerhard Kramler[2]
[1]Women's Postgraduate College for Internet Technologies (WIT),
[2]Business Informatics Group (BIG),
Institute for Software Technology and Interactive Systems
Vienna University of Technology, Austria
E-Mail: murzek@wit.tuwien.ac.at, kramler@big.tuwien.ac.at

Abstract

Because of today's diversity of models and heterogeneous modelling languages in the area of business modelling there is a need to automate model transformations on the level of business process models. To define such transformation processes in a simple way, a graphical modelling approach is required. In previous works a textual model transformation tool, the BMT (BOC Model Transformer) has been introduced. It enables the transformation of business process models which are instances of different meta models. A rule file which contains XML-coded instructions controls the transformation process. To simplify the requirements for creating such a rule file this paper introduces a graphical approach for modelling transformation processes. For this purpose a meta model for the transformation language of the BMT is introduced. Furthermore transformation processes which represent model instances of this meta model are illustrated and discussed.

1. Introduction

Organisations require business process management tools that support model transformation between different business process meta-models, because of the following business requirements:

Model exchange: Multinational organisations use different modelling methods for modelling their business processes. They need to exchange their models in order to provide transparency of their organisational structures.

Method change: Due to organisational needs companies change or update their modelling method. After updating, they want to automatically transfer their whole stock of models from the old method into the new one.

Model deployment: A current request on model transformations in the area of business modelling is to derive basic IT infrastructure models from already existing business process models.

The BOC Model Transformer (BMT) has been designed to support these kinds of model transformations [5, 8]. It is a transformation tool which supports the idea of Enterprise Model Integration (EMI) [6] and of Model Driven Architecture (MDA) [9]. The BMT transforms business process models based on meta model X (for example event driven process chains) into business process models based on meta model Y (for example UML action diagrams).

* "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."

The transformation uses an XML-based rule file, which contains the description of the transformation process and the transformation rules.

The utilisation of the BMT in a number of projects has shown that for most users it is too complicated to define the whole transformation process textually. Basically, knowledge of XML and the structure of model representation in the meta modelling tool are needed. Additionally the usage of the rule file elements must be studied intensively to write efficient and executable rule files.

A graphical transformation language could reduce the requirements for constructing rule files for the BMT. Therefore this work proposes to graphically model the transformation process based on a meta model. For the implementation of the meta model we use the Business Process Management Tool ADONIS® [3].

The remainder of the paper focuses on the meta model and the graphical representation. Chapter 2 gives overview of the implementation process, the meta model and examples for graphical model transformations. Chapter 3 related work. The paper concludes with a summary and a position statement.

2. Graphically Modelling a Transformation Process

The goal of offering a possibility to *graphically* model transformation processes based on the BMT is realized in three steps, as shown in Fig. 1.

The first step is to create a meta model for transformation processes by abstracting away the XML-specifics out of the syntax of the rules file. The next step is to define the meta model in an appropriate way using the metamodel-enabled tool ADONIS®. Based on the meta model, the tool provides the user a graphical possibility to model the transformation process. The last challenge is to generate the rules file automatically out of ADONIS®' file format.

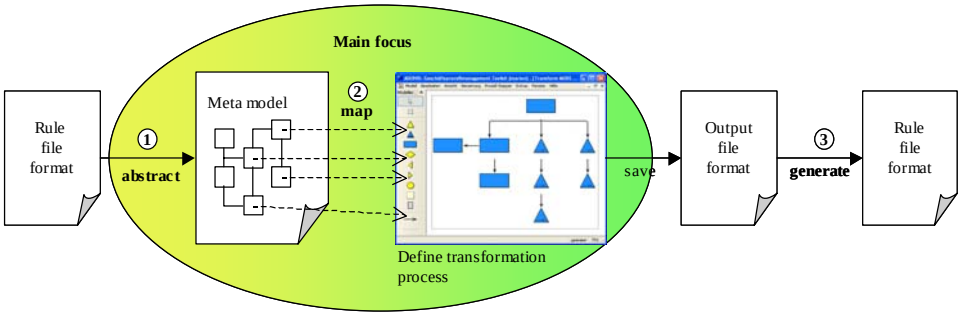

Fig. 1: Three steps to enable the graphical creation of rule files

2.1 Characteristics and Features of the BMT

To get familiar with the BMT and its mode of operation this section gives a short description of the architecture and main elements.

The BMT consists of a transformation engine and a transformation language. The language is used for describing the transformation process in a rules file. During the transformation process the transformation engine reads the rules file and executes the instructions step by step. The main instructions which are available to construct rule files are: navigations, definitions, functions, rules and conditions.

Navigations are used to match model fragments in the source context. *Definitions* are comparable with variables. The values assigned to definitions can either be fixed values or source values selected by functions. Beside selecting values, functions are also used to manipulate values. Therefore functions can be subdivided in two groups, *manipulation functions* and *selection functions*.

To generate the target models and model fragments *rules* are used. Rules are able to create model fragments in the target context, e.g. models, classes, relations, attributes etc. Depending on how a model fragment is transformed we distinguish *copy-rules* and *create-rules*. *Conditions* are used to decide if rules should be executed or not, e.g. depending on a comparison of two attribute values.

For further details regarding to the BMT and its rules file refer to [5] and [8].

2.2 Meta Model for Transformation Processes of BMT

The meta model shown in figure 2 describes the main classes of a transformation process and their relationships. The meta model itself is an instance of the meta^2-model of ADONIS® [3]. Consequently all three participating meta models – source-, target- and transformation meta model – are instances from the same meta^2-model.

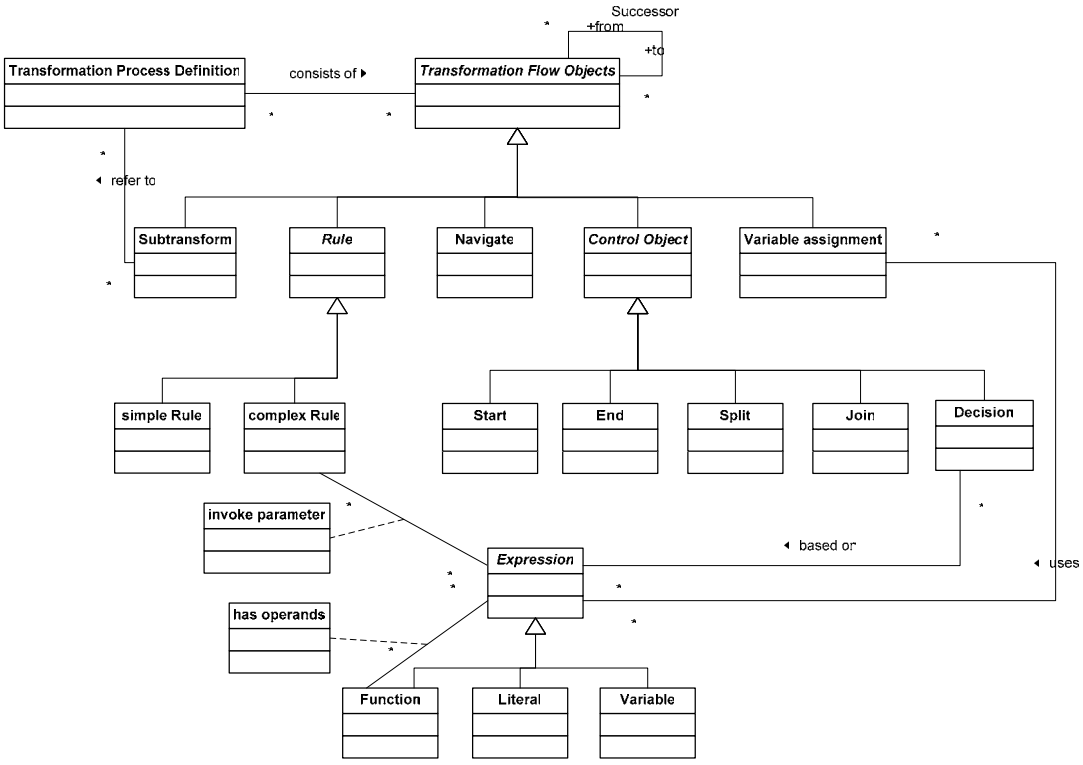

Fig. 2: Part of the meta model for transformation processes

In the following the entities are described in detail, including the most important of their attributes.

The *Transformation Process Definition* describes the process itself. It provides header-information for the transformation process, e.g. the *name* of the process, *description*, *author* etc. A process definition consists of *Transformation Flow Objects*.

The flow object *Subtransform* represents the invocation of a transformation sub process. Beside its attribute *name*, it also has a relation *“refer to”* which points to the sub process. It is used to structure the whole transformation process in logical parts.

The class *Navigate* provides the possibility to explicitly model navigations within the source context. The attribute *OfType* specifies the types of elements to be matched in the source context.

The class *Rule* is designed to create the destination output. It is used to copy existing model elements or create new ones in the target context. The abstract class *Rule* generalises the classes *simple Rule* and *complex Rule*. The attribute *type* specifies which kind of rule it is, for example model-, instance-, relation-, attribute rule etc. Each rule has parameters specifying further details of the target element to be created. Options are to either take values of the concerning source element or provide literal values for the target element. A complex Rule is characterised by its additional relation *“invoke parameter”*, which is used to generate parameter values via functions.

Start, end, split, join and decision describe the *control objects*. Together with the relation *“successor”* these objects describe the control flow of the transformation process.

The abstract class *Expression* describes the computation of a certain value. It generalises the classes *Function*, *Literal* and *Variable*. The relation *“has operands”* expresses that a function could use further expressions to evaluate the result value.

2.3 Graphical Representation

Given the different meta classes, relations and their attributes we now have to find an adequate graphical representation for these concepts.

The main problems of a graphical representation of transformation processes are:

- Simple/intuitive representation of non-trivial transformations [1].
- The balanced proportion of graphical elements and text.
- The elements should be self-explanatory. This means the concepts represented should be more concise and intuitive in graphical form compared to the textual one [10].
- Not too many different models and model elements should be provided.

The most difficult issue in graphically representing transformation processes is the first one – how to represent non-trivial transformations.

In the following two examples of models instantiated from our meta model are provided. These two examples show graphical representations of parts of transformation processes addressing the problems stated above.

First the problem is described. Then the graphical representation and its description is provided. Finally the XML-code representation of the solution is presented.

2.3.1 Example 1: Replacement of Strings

The first case deals with a problem which often occurs in the case of business model transformations, replacement of substrings. The problem is to create new models, one for each model of type “BusinessProcessModel” and replace all Ü, Ä and Ö by Ue, Ae and Oe in the model name. Fig. 3 illustrates the solution of this problem.

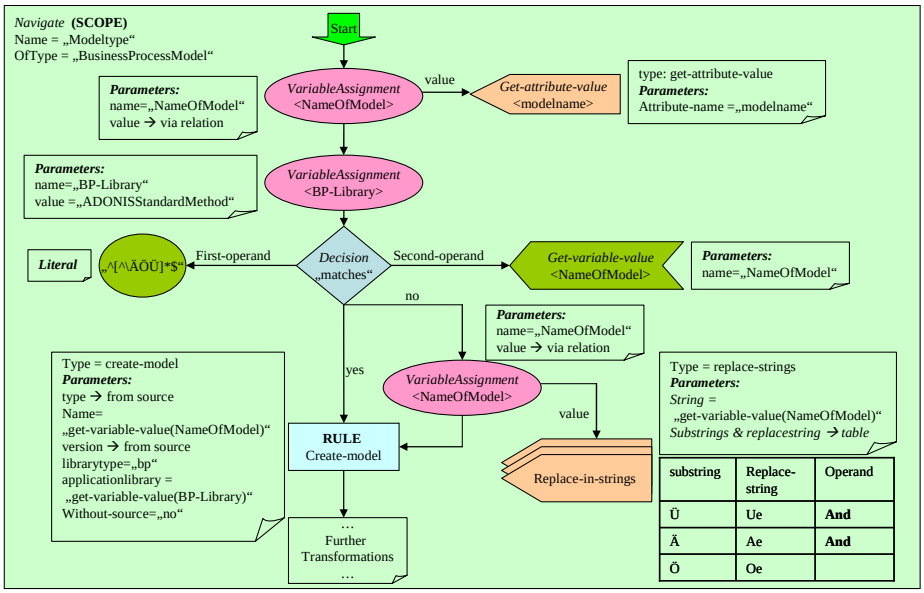

Fig. 3: Creation of models and replacing letters in the model name – graphically

The rectangle containing all of the other elements represents the navigation to each model of type "BusinessProcessModel" within the source context. This implies that during the whole process fragment - contained in this rectangle - the source relation are models of the type mentioned above. Navigations are comparable with "foreach"-loops used in programming languages. That means for each model of type "BusinessProcessModel" in the source context the process inside the rectangle is executed. Therefore the visualization as rectangle has been chosen.

Within the navigation element, first of all two variables are created and values are assigned. The variable-assignment for "NameOfModel" uses a function to get the value out of the source-model-attribute "modelname". To the second variable "BP-Library" the literal "ADONISStandardMethod" is assigned.

The next step is a decision whether or not the name of the model contains Ä, Ö or Ü. The comparison is visualized with two related objects, a literal which contains a regular expression and a function which gets the value out of the variable "NameOfModel". If the name does not contain any of these letters then the model will be created. Otherwise a variable assignment which uses the function "replace-in-strings" precedes the creation of the model.

The multipart object "replace-in-strings" illustrates that the string-replacement has to be done more than once, for each substring which is stated in the table. This kind of representation has been a solution of the problem "how to represent a complex function intuitively". In preceding graphical drafts many single functions have been used to express this circumstance. This has shown that a consolidation of equal functions is necessary.

The object "create-model" indicates that a target model will be created. Two of the parameters (librarytype and Without-source) of the object "create-model" are filled with literals, two (type and version) are taken out of source and two (name and applicationlibrary) use the expression "get-variable-value" to get the assigned values.

Fig. 4 shows the same transformation process written in XML statements. Due to shortage of space, only one of the three IF-statements, implementing the string-replacement is stated.

```
<MODELTYPE name=„BusinessProcessModel“>
<NAMEVALUEMAP>
  <ELEM name=„NameOfModel“>
    <select-value>
      <PARAM name="attribute">modelname</PARAM>
    </select-value>
  </ELEM>
  <ELEM name=„BP-Library“>ADONISStandardMethod</ELEM>
</NAMEVALUEMAP>
<IF>
  <COMPARE>
    <LEFT-VALUE>
      <get-map-value>NameOfModel</get-map-value>
    </LEFT-VALUE>
    <CONDITION>matches</CONDITION>
    <RIGHT-VALUE>^[^\ÄÖÜ]*$</RIGHT-VALUE>
  </COMPARE>
  <THEN/>
  <ELSE>
    <CALL name="replace-in-strings"/>
  </ELSE>
</IF>
    <RULE type="create-model">
      <PARAM name="Modeltype">Ownerpool</PARAM>
      <PARAM name="Modelapplib">
        <get-map-value>BP-Library</get-map-value>
      </PARAM>
      <PARAM name="Modellibtype">bp</PARAM>
      <PARAM name="Modelname">
        <get-map-value>NameOfModel</get-map-value>
      </PARAM>
    </RULE>
…
</MODELTYPE>
```

```
<RULESCONTAINER name="replace-in-strings">
  <IF>
    <COMPARE>
      <LEFT-VALUE>
        <str-find>
          <param name="string">
            <get-map-value>NameOfModel</get-map-value>
          </param>
          <param name="substring">Ü</param>
        </str-find>
      </LEFT-VALUE>
      <CONDITION>equals</CONDITION>
      <RIGHT-VALUE>-1</RIGHT-VALUE>
    </COMPARE>
    <THEN/>
    <ELSE>
      <NAMEVALUEMAP>
        <ELEM name="string2replace">
          <str-replace>
            <param name="string">
              <get-map-value>NameOfModel</get-map-value>
            </param>
            <param name="substring">Ü</param>
            <param name="replacestring">Ue</param>
          </str-replace>
        </ELEM>
      </NAMEVALUEMAP>
    </ELSE>
  </IF>
  <IF>
    … repeat for each string to replace
  </IF>
</RULESCONTAINER>
```

Fig. 4: Creation of models and replacing letters in the model name – textually

2.3.2 Example 2: Getting Attribute Values from Related Objects

The second example again demonstrates the use of navigations (see Fig. 5). The goal is to create a copy of all connectors (= relation between two model instances) of type "successor" in the destination context. For each successor pointing away from an "XOR"-instance the value of the attribute "transitioncondition" is filled with the value of the attribute "description" of the instance the successor points to.

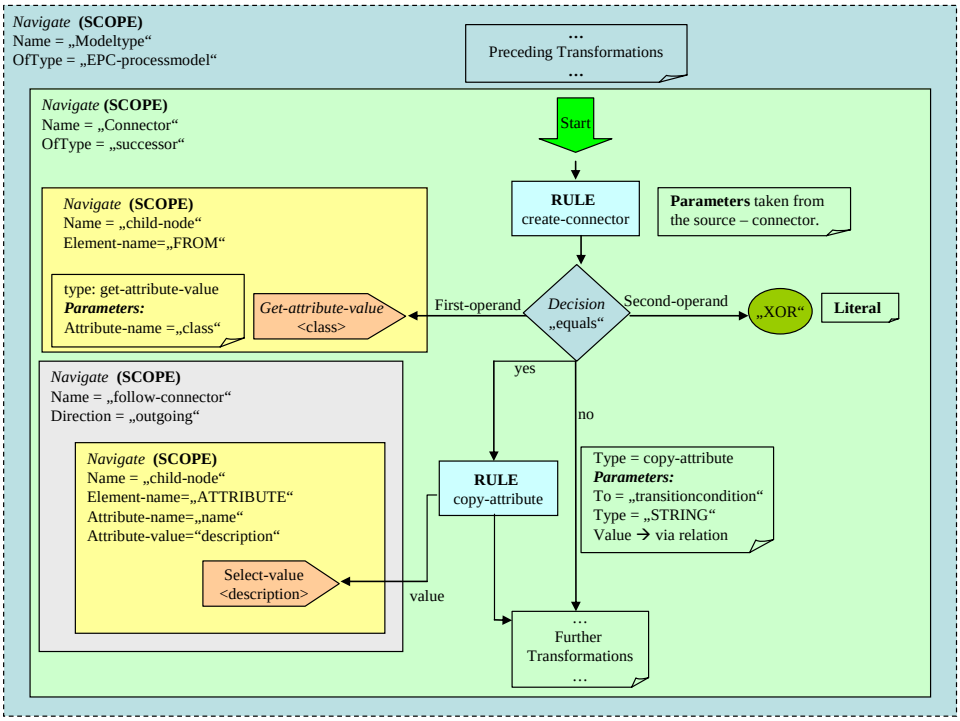

Fig. 5: Creating connectors with attribute values from related instances

The outermost rectangle illustrates the navigation to each model of type "EPC-processmodel" within the source context. The rectangle within shows the navigation to each connector of

type “successor” within a model of type “EPC-processmodel”. Then a connector is created (RULE create-connector) in the target context. The parameter values for the creation of the new connector are taken from the source connector.

The following decision compares the literal “XOR” with the name of the instance from which the connector is pointing away. This value received by navigating to the FROM-Element of the connector itself and selecting the value of the class-attribute.

If the value does equal “XOR” the attribute “transitioncondition” is created within the target connector. In doing so the value is evaluated as follows: First navigate to the instance where the connector is pointing to. Then navigate to the attribute “description” of this instance and select the value, visualized with the function “select-value”.

The same transformation process written in XML statements can be found in [5].

2.3.3 Graphical vs. Textual Representation

In the graphical representation the complexity is reduced by visualizing the language elements by the means of different colours and shapes. The visualization of navigations as rectangles makes sure that the user always knows in which part of the source context he or she is actually operating. Compared to the textual representation the graphical representation highlights the process flow, which is delimited by the elements *Start* and *End*. Furthermore the readability of the graphical representation is supported by summarizing repeating instructions, for example the “replace-in-strings” function in Fig. 3. People who are not familiar with the syntax of XML will develop transformation processes faster in the graphical way.

2.5 Related Work

Krzysztof Czarnecki and Simon Helsen identified design features to categorize Model transformation approaches [2]. According to these features the BMT could be categorized as *Hybrid Model-to-Model Approach*.

One of the graphical representations of transformations which has been tested was the Bidirectional Object oriented Transformation Language (BOTL) [7]. It is a *Graph-Transformation-Based approach* which is already implemented in a tool called BOTL. It offers the possibility to specify rules graphically. The rules are structured in LHS/RHS in form of two graphs. The information in which order the rules are executed can not be influenced, it is left to the implementation itself. This is one of the main differences to the BMT which is focusing on modelling the transformation process itself.

A *Pattern-Based approach* for graphically representing transformations is MOLA (MOdel transformation LAnguage) [4]. It represents the transformations as structured flowcharts with pattern-based rules. They combine rule patterns with a graphical loop concept. The visualization of loops in MOLA is very similar to the concept of navigations in the BMT. But the representation of the contained elements differs completely.

The focus of the most graphical representations of model transformations lies on visualizing the source-element and the resulting target-element which represents a rule. The graphical language in this paper highlights the transformation process itself and how the rules are applied. The similar structure of graphical transformation processes and the underlying source- and target models make it easier for experienced business process modellers to create model transformations this way.

4. Summary and Statement

To cope with the complexity of the requirements for transforming business process models we introduced the graphical approach to model transformation processes with the BMT.

The advantages of the graphical representation are that no knowledge of XML is required. The graphical representation is concise and intuitive for both kind of users, developers and business experts. The user is able to control the transformation process by arranging the transformation elements due to his or her requirements.

This graphical modelling approach involves the vision of MDA [9] in two ways. First with the BMT, which is a model transformer for business models and secondly with the idea of creating a model of the transformation process and transform it into XML-code for the transformation tool.

Next steps are to evaluate the graphical modelling language of the BMT in practical use and transforming the output file of ADONIS® into an according rule file to automate the graphical transformation.

5. Acknowledgment

We would like to thank BOC Information Technologies GmbH for providing the BMT[1] and the Business Process Modelling Tool ADONIS® in which the meta model has been configured.

References

[1] Bettin, J.: Ideas for a Concrete Visual Syntax for Model-to-Model Transformations. OOPSLA'03, Workshop on Generative Techniques in the Context of Model-Driven Architecture.
[2] Czarnecki, K.; Helsen, S.: Classification of Model Transformation Approaches. OOPSLA'03, Workshop on Generative Techniques in the Context of Model-Driven Architecture.
[3] Junginger, S.; Kühn, H.; Strobl, R.; Karagiannis, D.: Ein Geschäftsprozessmanagement-Werkzeug der nächsten Generation – ADONIS: Konzeption und Anwendungen, Wien im April 2000. An abridged version of this BPMS-report was published in WIRTSCHAFTS-INFORMATIK 42 (2000) 5, S. 392-401 (in German).
[4] Kalnins, A.; Barzdins, J.; Celms, E.: Basics of Model Transformation Language MOLA. ECOOP 2004, Workshop on Model Driven Development.
[5] Kühn, H.; Murzek, M.; Bayer, F.: Horizontal Business Process Model Interoperability using Model Transformation. Workshop INTEREST at ECOOP 2004, Oslo, June 2004.
[6] Kühn, H.; Bayer, F.; Junginger, S.; Karagiannis, D.: Enterprise Model Integration. In: Bauknecht, K.; Tjoa, A M.; Quirchmayr, G. (Hrsg.): Proceedings of the 4th International Conference EC-Web 2003 - Dexa 2003, Prague, Czech Republic, September 2003, LNCS 2738, Springer-Verlag, pp. 379-392.
[7] Marschall, F.; Braun, P.: Model Transformations for the MDA with BOTL. In Rensink, A., ed.: CTIT Technical Report TR-CTIT-03-27, Enschede, The Netherlands, University of Twente (2003) 25–36
[8] Murzek, M.: Methodenübergreifende Modelltransformationen am Beispiel von ADONIS. Diploma Thesis, University of Vienna, April 2004 (in German).
[9] Object Management Group: MDA Guide, Version 1.0.1, 12. Juni 2003. http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf, access 12 April 2004.
[10] Sendall, S.; Kozaczynski, W.: Model Transformation – the Heart and Soul of Model-Driven Software Development. Software, IEEE , Volume 20 , Sept.-Oct. 2003, Pages: 42 - 45

[1] The BOC Model transformer (BMT) was designed and implemented by one of the authors during her employment at the BOC Information Systems GmbH.