Applying Model Transformation By-Example on Business Process Modeling Languages*

Michael Strommer[1], Marion Murzek[2,**], and Manuel Wimmer[1]

[1] Business Informatics Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology, Austria
{strommer,wimmer}@big.tuwien.ac.at
[2] Womens Postgraduate College of Internet Technologies
Institute for Software and Interactive Systems
Vienna University of Technology, Austria
murzek@wit.tuwien.ac.at

Abstract. Model transformations are playing a vital role in the field of model engineering. However, for non-trivial transformation issues most approaches require imperative definitions, which are cumbersome and error-prone to create. Therefore, Model Transformation By Example (MTBE) approaches have been proposed as user-friendly alternative that simplifies the definition of model transformations. Up to now, MTBE approaches have been applied to structural models, only. In this work we apply MTBE to the domain of business process modeling languages, i.e., Event-driven Process Chains and UML activity diagrams. Compared to structural languages, business process modeling languages cover static semantic constraints, which are not specified in the metamodel. As a consequence, reasoning on the abstract syntax level is not sufficient. The contribution of this paper is to extend our existing MTBE approach by new alignment operators on the user level, which further improves the transparency of model transformation code. Concrete syntax and the knowledge about mapping operators are to be the only requisite artifacts.

Keywords: Business Process Models, Model Transformation, MTBE.

1 Introduction

With the rise of model engineering, model transformations have been steadily put in the limelight in the past five years. Growing tool support indicates, that model transformations are not only attractive for researchers, but also for industrial parties accommodating to their customers needs. Model transformation

* This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-810806.0.
** 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.

J.-L. Hainaut et al. (Eds.): ER Workshops 2007, LNCS 4802, pp. 116–125, 2007.

scenarios include transformations between different refinement levels of models (*vertical transformations*) as well as transformations between different modeling languages which rely on the same abstraction level (*horizontal transformations*). Languages used for defining model transformations are ATL [4], QVT [8], Triple Graph Grammars [2], XSLT, and even general purpose languages such as Java [1]. Many model transformation languages are hybrid, meaning that besides a declarative style, an imperative style is provided for problems that cannot be solved by the provided declarative features. Nevertheless, the development of imperative fragments is often a tedious and error-prone task. In contrast, declarative solutions are compact descriptions but not always intuitive to find. We believe Model Transformation By-Example (MTBE) can help overcome the difficulties arising in the production of declarative and imperative transformation code.

Instead of focusing on the domain of *structural modeling languages*, what has been done in previous investigations [11], [10], in this paper we concentrate on *behavioral modeling languages*. More specifically, we apply MTBE on the domain of business process modeling (BPM), which, up to our best knowledge, has not yet been subject to the MTBE approach. The definition of requirements for MTBE in the context of business process modeling and the specification of proper mapping operators comprise the main contribution of this paper. Therefore, we present *main challenges encountered in business process (BP) model transformations*, and how these challenges can be tackled by extending already proposed MTBE *mapping operators*. The proposed extensions are explained by a running example in which two prominent BP modeling languages are used, namely the UML Activity Diagram and Event Driven Process Chains.

2 Motivation for MTBE

In this section we give a brief outline of our MTBE approach we introduced in [11]. We have recognized two main issues in conjunction with the task of defining model transformations. The first one is about the gap between the way a person thinks about models and the way a computer represents those models internally. And the second issue is about the way concepts are represented in the metamodel (MM), i.e., whether one needs to have expert knowledge to identify those concepts or not. We call the phenomenon of hidden concepts in a metamodel *concept hiding* [5]. Having those issues in mind one can easily accept the fact, that the task of creating model transformation rules is not a user-friendly one. This is why we have come up with the idea of MTBE, that can be seen as a semi-automatic approach for the generation of model transformation rules. One of the main benefits of MTBE is the shift in abstraction. Mostly all of the proposed model transformation approaches operate on the abstract syntax (AS), although modelers might not be familiar with the abstract syntax. Therefore, we intend to make the transformation task more concrete and operate on a level the modelers or designers are familiar with, i.e, on the concrete syntax (CS). Hence, with the application of MTBE one does not need any programming language experience or knowledge of the underlying metamodel to have some model

transformation defined and executed. Furthermore we envision MTBE as agile and iterative process, in which the user can consequently improve the transformation outcome by adjusting the mappings and mapping operators applied on the CS.

3 Models for Business Processes

Business process models are in use for quite a long time and continue to gain importance as support from the software engineering field is improving significantly. Particularly model engineering fosters research in the area of BPM. There exist several metamodels for existing languages in order to raise their acceptance and tool interoperability. Due to this growing interest in BPM and proper tool support, we believe MTBE can be advantageous for specifying model transformations between BP models. As real world scenario consider the case in which two companies , that use different BPM techniques, merge and have to integrate their BP models. In this work we use the two BPM languages UML 2.1 AD [9] and EPC [6] shown in Figure 1 for our running examples.

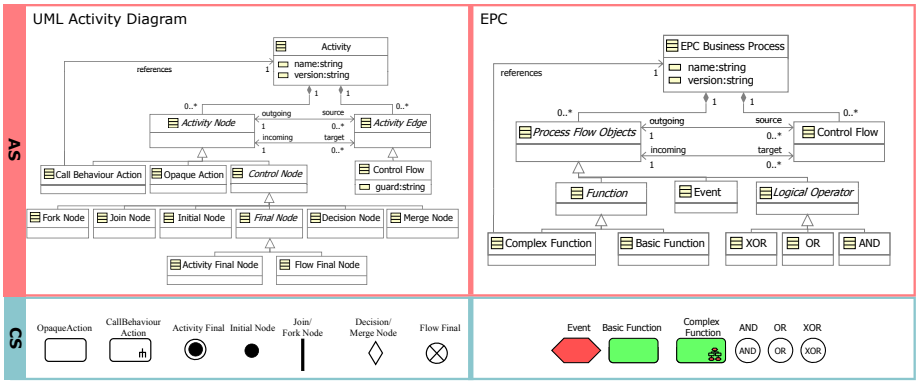

Fig. 1. Parts of the UML 2.1 AD and EPC meta models and their concrete syntax

4 Mapping Operators for MTBE in the Light of BPM

During our investigation of BP models we discovered, that there are considerable differences compared to structural models concerning the requirements for MTBE. To transform structural models, one has to be familiar with the notation and hidden concepts in the metamodels, especially when dealing with UML diagrams. Resulting ambiguities on the metamodel layer have to be solved either by reasoning algorithms or user input, as we described in detail in [11]. Now, with the task of transforming BP models we have to deal with quite different issues, in order to apply our MTBE approach. A lot of interesting aspects concerning the heterogeneity of BP models have been identified by Murzek et al. in [7]. The

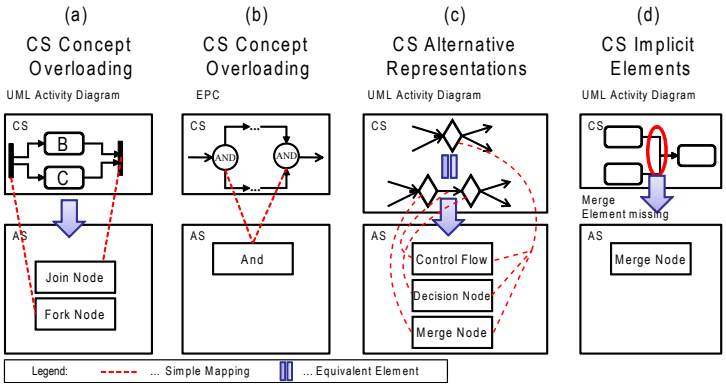

Fig. 2. Overview of BP models heterogeneities

heterogeneities shown in Figure 2 are partly based on their work. One of the special requirements coming along with BP models has its root in the mapping from concrete to abstract syntax layer (notation) and the number of modeling elements involved on each layer. As we now allow for zero or more elements on each layer in the CS-AS mapping, the notation can be now defined as

$$Triple := < as\_E^*, cs\_E^*, const(as\_E)? > \quad (1)$$

Note that the original definition of the notation defined in [11] has been extended in this work. In UML AD we have for example the notation:

$$< \{MergeNode, ControlFlow, DecisionNode\}, \{DecisionMergeFigure\}, \{\} >$$

as is illustrated in Figure 2 *c* for the CS modeling element on the very top. Note that the used modeling construct is here just an abbreviation on the CS layer and could be equivalently expressed by the following pattern of notation triples:

$$< \{DecisionNode\}, \{DecisionFigure\}, \{\} >$$
$$< \{ControlFlow\}, \{ConnectionFigure\}, \{\} >$$
$$< \{MergeNode\}, \{MergeFigure\}, \{\} >$$

We also observed several heterogeneities between modeling languages, which pose further requirements for MTBE. Figure 2 gives four examples for the peculiarities we found in the two BP modeling languages we introduced in Section 3. Examples *a* and *b* in Figure 2 depict the case of so called CS overloading in UML AD and EPC. In example *a* we encounter no problems because with the help of the notation we can distinguish between the two concepts join and fork despite the CS overloading. In example *b* CS overloading represents a real challenge for MTBE as two equal CS elements, but in fact featuring two different meanings, are mapped to the same AS element.

When we have to deal with alternative representations in the CS, see Figure 2*c*, we can use the notation in MTBE to find them. The challenge arises not until we have to map two languages, where one consists of such variation points in the CS. Example *d* in Figure 2 shows the possibility in UML AD to merge

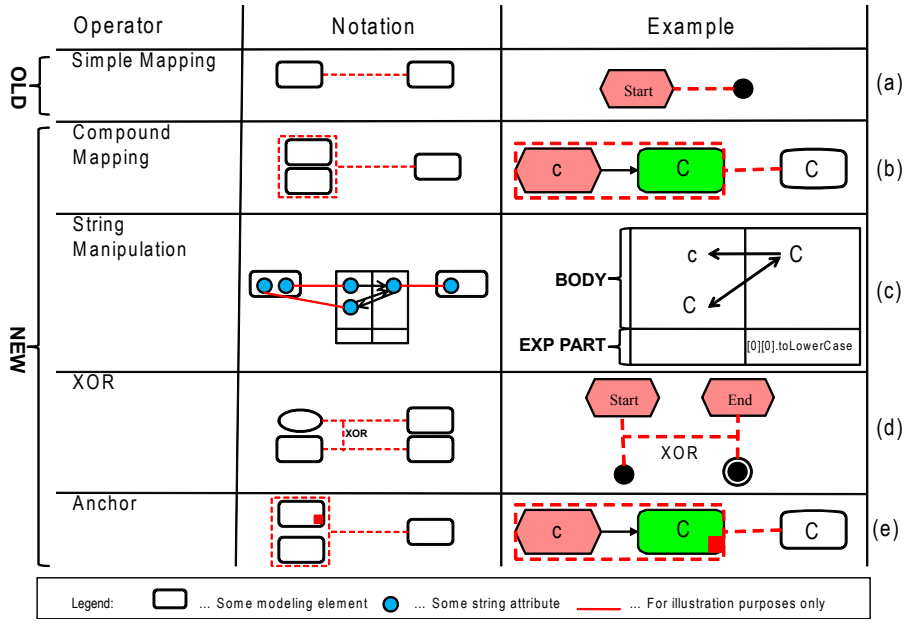

Fig. 3. Overview of MTBE Mapping Operators

parallel flows implicitly by omitting a merge/join node, i.e., we have no mapping from the AS to the CS.

In the following we present new mapping operators, which resolve heterogeneities, as expressed in the examples *a*, *b*, and *c*, in Figure 2. Unfortunately, up to now we are not able to cope in MTBE with implicit elements as shown in example *d*. The problem here is twofold. First we have to address the question how to map these implicit elements on the concrete syntax layer. And second we have to adjust the code generation process accordingly. One could now argue, that an enrichment of the presented metamodels could avoid a lot heterogeneities. But there exist amounts of legacy systems with legacy models, that can not be easily adopted to cooperate with new adjusted metamodels.

So far our MTBE approach as presented in [11] has only dealt with *simple 1:1 mappings* on the concrete syntax, see Figure 3 *a*. In case of BP model transformations it is necessary to introduce new kinds of mapping operators. Based on the specialties and problems stated above we developed new operators. In the following we describe the semantics of these new operators to provide a notion of how they can be used. However, we still have to develop some formal specification for these operators.

The first new operator is the *compound mapping operator* (cf. Figure 3 *b*). This mapping operator allows for n:m mappings on the CS layer. Although we encountered only 1:n mappings so far, we want the user to have the feature of n:m mappings. With this mapping operator we intended to support the mapping

of common work flow patterns, such as the one we show in the corresponding example for this operator. The basic idea of our compound mapping operator is however to support pattern matching on the object graphs.

Along with the compound mapping operator comes a *string manipulation operator*, that works in the context of compound mappings but is not restricted to them. A first notation approach is shown in Figure 3 *c* together with an example. Note, that this operator is used for *Attributes* specified in the metamodel, which are represented as labels in the model. This operator consists of two main components, i.e., a body and an expression part, each separated into left and right hand side. The two body parts consist of a list containing references to *Attributes*, that are going to be mapped. Furthermore each *Attribute* of the body manages a list containing the unidirectional mappings from itself to some other *Attributes*. During the transformation rule generation from one language to the other only one list of mappings is of interest. In the expression part one can use some simple string operations or regular expression. In the example given in the Figure above we apply a *toLowerCase* operation on the first mapping of the first *Attribute* on the right hand side.

For the *XOR operator* depicted in Figure 3 *d* there are two ways to use it, i.e., in an explicit way or in an implicit way. In Figure 3 we only illustrated the explicit use of this operator. In general an XOR mapping shall indicate that only one element should be created although one CS element in one language is mapped to more than one element in the other language. Omitting the XOR in the example in Figure 3 *d* would lead to the creation of an *Initial* and an *Activity Final Node* for every *Event* that is matched by the corresponding transformation rule. When using the XOR operator in an implicit way the whole issue is hidden from the user. Instead all XOR mappings are derived automatically in the metamodel as will be shown in Section 5. The drawback of this approach is that one looses the possibility of multiple object creations as mentioned before.

At last we introduce the *anchor operator*. The notation and an example are given in Figure 3 *e*. The anchor operator marks the element, which the transformation rule shall use as single source pattern element. It is thus always used in conjunction with the compound mapping operator, which usually leads to the creation of multiple source pattern elements in the rules.

5 MTBE with UML AD and EPC By-Example

Our MTBE approach for the domain of Business Process modeling can be best explained in a by-example manner. Therefore, we use the two BP languages EPC and UML AD described in Section 3. Although the example given in Figure 4 is rather simple, it still covers a lot of interesting aspects for MTBE.

For the case study we assume that on the concrete syntax layer in EPC's *Events* and *Basic Functions* to always occur pairwise connected through a *Control Flow* edge. Furthermore, in UML AD modeling it could be possible to omit a *Join* node and therefore model joins implicitly. However, in our first MTBE approach for BPM we do not jet cope with implicit joins or merges.

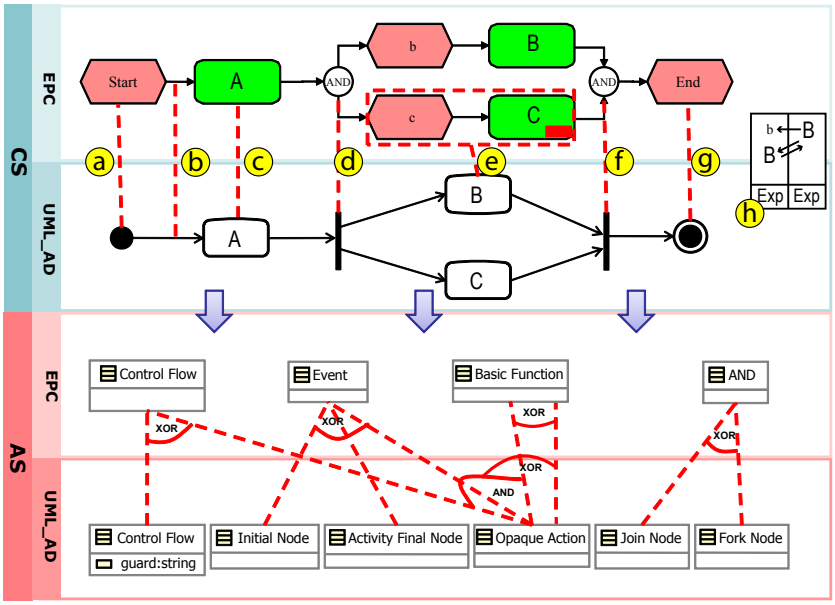

Fig. 4. Mapping EPC and UML Activity diagram - CS + AS perspectives

5.1 Model and Metamodel Mappings

As a first step one has to define manual mappings between two languages, which the transformation model shall be derived from. In the example in Figure 4 we specified eight mappings that capture all concepts being used in the two sample models. Mappings *a,b,c,d,f,* and *g* are of type simple mapping.

Mapping *e* is of type compound mapping with multiplicity 1:3. Consequently, whenever the pattern *Event*, *Control Flow*, *Basic Function* is matched this corresponds to a single *Opaque Action*. We also marked the *Basic Function C* in our compound mapping as anchor element, which has implications specific to transformation code generation. In our case the ATL code generator would use this *Basic Functions* metamodel element as single source pattern element instead of using multiple source pattern elements. During our implementation attempts we realized, that an anchor feature can be desirable in some transformation scenarios.

Mapping *h* in our example takes care of the labels used in *Events*, *Basic Functions* and *Opaque Actions*. To maintain usability this string manipulation operator is used in a separate modeling element and references the involved labels. To define string equivalences one can use only unidirectional mappings, which will be applied transforming from one set of labels to another. An optional expression allows us for example in mapping *h* to apply a *toLowerCase()* operation on the first mapping of the right hand side set of labels.

In EPC's there are no distinct metamodel elements nor distinct concrete syntax elements for start and end nodes, although these concepts are used in the

modeling language implicitly. In UML AD we do have explicit concepts for start and end nodes both, in the model and the metamodel. If a transformation from EPC2UML_AD has to be performed the transformation model must know how to distinguish between start and end nodes even without having these concepts specified in EPC.

To keep our illustration in Figure 4 transparent and clear we omitted the mappings between CS and AS. Also these mappings are quite straightforward to define, as there are no constraints specified in the notation.

At last the mappings between the two metamodels can be derived from the user mappings and the notation. To highlight the existence of a compound mapping in the metamodel we marked the three involved mappings with an *and* operator. On the metamodel mapping level we now make use of our new XOR operator we introduced in Section 4. To keep the mapping task user-friendly the XOR between mappings can be reasoned automatically based on information in the metamodels. Whenever a meta class contains at least two outgoing mapping edges, an XOR relation can be set in an implicit way.

5.2 How to Make Mappings Executable

As the automatic generation of transformation rules is a difficult task, we do not claim to support fully automatic rule generation. Instead we believe in a semi-automatic approach. To face the new domain of business process models we implemented a paradigm, which can be best compared to Architecture-Centric MDSD. First of all we have implemented correct ATL transformation code, which acts as reference implementation. Thereby we have avoided imperative code sections and concentrate on coding in a declarative fashion.

In the next step we have developed the mapping operators described in Section 4. During this step we have turned our attention to the user-friendliness.

Next we have looked at the example models, the user mappings and the metamodels and tried to deduce the reference implementation. Code segments that could not be deduced automatically then lead to further refinement of the underlying heuristics. After refinement we tried again to deduce the reference implementation. This process can be seen as an iterative way to deduce heuristics on how to generate ATL transformation rules from a given set of models, metamodels and user mappings. The aim of this process is to optimize the relation between user-friendly mapping operators and the ability to generate executable transformation rules.

6 Related Work

To our best knowledge, there exists no approach for finding semantic correspondences between business process models so far. However, there exists general approaches that allow the definition of semantic correspondences between two (meta)models, which have been applied in the area of structural models. The first approach is a *model-based* approach from Varró, while the second approach

from Fabro et al is *metamodel-based* which allows automatically finding correspondences directly between metamodels.

Model-based: Parallel to our MTBE approach [11] Varró proposed in [10] a similar approach. The overall aim of Varró's approach is comparable to ours, but the concrete realizations differ from each other. Our approach allows the definition of semantic correspondences on the concrete syntax, from which ATL rules can then be derived. In contrast, Varró's approach uses the abstract syntax to define the mappings between source and target models, only. The definition of the mapping is done with reference nodes leading to a mapping graph. To transform one model into the other, graph transformation formalisms [2] are used. Furthermore, there is no publication on applying Varro's approach on business process models which is the general aim of this paper.

Metamodel-based: Orthogonal to MTBE there exists the approach of using matching transformations combined with weaving models [3] in order to generate an ATL transformation model. Matching transformations are defined such that they use common matching algorithms or modifications, and then create a weaving model from the calculated similarity values. Afterwards these weaving models are taken as input for another transformation, called higher order transformation (HOT), to produce the desired ATL model describing the transformation rules between two metamodels. Because there will be always some mappings that can not be matched fully automatically, this approach is also to be considered semi-automatic. The model transformation generation process described in [3] currently focuses on using mappings between metamodels and is therefore based on the abstract syntax, while our approach aims at generating model transformation code from M1 mappings.

7 Conclusion and Future Work

In this work we have proposed an MTBE approach for the area of business process modeling languages. We introduced special mapping operators to give the user more expressivity for defining model mappings. Still, there remain a few heterogeneity issues we can not target with the introduction of new alignment operators. These heterogeneity issues will be on the one hand subject to special reasoning algorithms and on the other hand as seldom as possible to manual code adjustment done by the user.

Concerning future work, we particularly strive for first, the refinement of the proposed MTBE approach. We want to experiment with different versions of mapping models which possess different levels of granularity, size and modeling patterns. In particular we want to evaluate how much a model can be altered compared to the original model, which is mapped by-example, and can still be transformed properly. Therefore, we want to map two models A and B, generate the transformations, and then alter the models to A' and B' and test the generated transformations with the new versions of the models. Also, we plan to

conduct student experiments to underpin the benefits coming along with MTBE in comparison to common model transformation approaches.

References

1. Akehurst, D.H., Bordbar, B., Evans, M.J., Howells, W.G.J., McDonald-Maier, K.D.: SiTra: Simple Transformations in Java. In: MoDELS/UML 2006. Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Genova, Italy, pp. 351–364 (October 2006)
2. Ehring, H., Engels, G., Kreowsky, H.-J., Rozenberg, G.: Handbook on Graph Grammars and Computing by Graph Transformation, vol. 2. World Scientific, Singapore (1999)
3. Fabro, M.D.D., Valduriez, P.: Semi-automatic Model Integration using Matching Transformations and Weaving Models. In: SAC. Proceedings of the 2007 ACM Symposium on Applied Computing, Seoul, Korea, pp. 963–970. ACM Press, New York (2007)
4. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
5. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: Lifting Metamodels to Ontologies - A Step to the Semantic Integration of Modeling Languages. In: MoDELS/UML 2006. Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Genova, Italy (2006)
6. Keller, G., Nüttgens, M., Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". Technical report, Institut für Wirtschaftsinformatik Universität Saarbrücken
7. Murzek, M., Kramler, G.: Business Process Model Transformation Issues. In: Proceedings of the 9th International Conference on Enterprise Information Systems, Madeira, Portugal (2007)
8. OMG. QVT-Merge Group: Revised submission for MOF 2.1 Query/View/Transformation, version 2.0 formal/05-07-04 edition (2005)
9. OMG. UML 2.1 Superstructure Specification. Object Management Group (April 2006), `http://www.omg.org/docs/ptc/06-04-02.pdf`
10. Varró, D.: Model Transformation By Example. In: MoDELS/UML 2006. Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Genova, Italy (October 2006)
11. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards Model Transformation Generation By-Example. In: HICSS-40 2007. Proceedings of the 40th Hawaii International International Conference on Systems Science CD-ROM / Abstracts Proceedings, Big Island, HI, USA, p. 285 (2007)