BPMN2BPEL using MoTMoT

Olaf Muliawan, Bart Meyers, Dirk Janssens
Universiteit Antwerpen
Antwerpen, Belgium
{olaf.muliawan, dirk.janssens}@ua.ac.be, bart.meyers@student.ua.ac.be

June 29, 2009

Abstract

The case of transforming BPMN to BPEL (and back) presents some interesting challenges. Due to the lack of a meta-model for both formalisms, we have implemented our own using MDR. Despite the limited timeframe we were able to implement structured and quasi-structured process models quite rapidly. The structure of the patterns to be converted is very suitable for graph transformation techniques, in our case MoTMoT, based on Story Driven Modeling (SDM). Furthermore, on the one hand, the emphasis on specific transformation guidelines provides a basis for graph transformations with control flows. On the other hand, the different rules presented by the folding algorithm (transform sequence pattern, transform while pattern, ...) are best scheduled non-deterministically. Knowing this, modeling such a transformation elegantly turns out to be quite complex: new language features are useful to successfully model this in SDM. We explain how we use higher order transformations (HOTs) to translate these new language features into original SDM constructs in a platform independent manner.

1 Introduction

The context of this paper is a tool contest of the GraBaTs 2009 workshop. The main goal within this contest is to showcase a graph-based transformation tool using a common reference case study: a transformation from BPMN to BPEL.

1.1 The case study

BPMN and BPEL are two industrial-standard formalisms to represent business processes. The two standards are complementary to each other, while at the same time sharing many characteristics. For this reason the proposed case is very interesting and relevant to the industry: providing BPMN to BPEL transformations [1]. However Chun et al. highlighted the problems of mapping BPMN to BPEL [4].

Providing round-trip engineering of BPMN and BPEL remains a challenge, even more so in the light of maintaining human readable code. Especially in the case of BPEL the generated code can be challenging to browse through and comprehend. This is due to the fundamental difference in notation: BPMN is modeled in acyclic graph patterns, while BPEL has operational semantics. For this reason we limit ourselves to (quasi-)structured BPMN models for which an exact mapping to BPEL is available, we elaborate more on this in Section 2.

1.2 The transformation tool

We chose MoTMoT [5] as transformation tool. This is a tool that transforms UML models of controlled graph transformations into executable Java code that conforms to the Java Metadata Interface (JMI), an API standard for accessing model repositories. It has been designed to illustrate how several model transformation problems of the Fujaba `CITE` tool can be solved.

1. MoTMoT is based on a UML compliant implementation of Story Diagrams. Story Diagrams support all constructs that have emerged from decades of research in controlled graph transformation. Therefore, MoTMoT's language may be considered to be expressive. Moreover, the UML 1.5 profile based implementation of this language enables industrial software engineers to specify graph transformations in their modeling tool of preference. Due to the maturity of industrial UML tools, features such as transformation views are available without any investment from the MoTMoT developers;

2. MoTMoT can be applied on OMG's MDA compliant inputs. On the one hand, models residing in a MOF repository can be transformed directly. On the other hand, file integration is supported by relying on the XMI standard;

3. MoTMoT is extensible. New language features can be added without investing time into modifications of the MoTMoT core. This has been shown with the implementation of features such as negative application conditions and non-determistic scheduling of rules (see Section 3).

For the purpose of this case study MoTMoT is going to be an asset in developing:

1. A standard mechanism of developing meta-models, conforming to MOF, stored in XMI and parsed by MDR. The developer can set up meta-models using UML drawing tools: each meta-model conforms to the JMI-MOF UML profile. Automatic generation of JMI interfaces is possible through MDR. This means the development of the case study is not dependent on a fixed meta-model: any adjustments in the meta-model are easily implemented. So, the approach of incremental development is possible through frequent updates and expansions of the meta-model until the final stage when the meta-model is fully constructed.

2. A flexible control flow mechanism that enables the transformation of different sorts of subpatterns to advance in a staged manner. A fine-grained control flow is possible using new language features, explained in Subsection 3.

We give an overview of the case study in Section 2. Section 3 will explain the relevance and application of higher order transformations within this case study. In Section 4, details on the implementation are provided. Finally, we finish with a conclusion in Section 5.

2 Case study overview

The case presents BPMN to BPEL transformations. The two formalisms do not have MOF-compliant meta-models. BPMN does not even have a formal XML Schema specification. While BPEL has such a schema, a MOF meta-model is lacking. We have based our meta-models on the case study description. These are the highlights regarding the implementation of our case in MoTMoT:

- The use of platform-independent *language extensions*. To implement the case study we need new language features. These language features are not implemented in MoTMoT itself, but are implemented as so-called higher order transformations (HOTs). HOTs can translate instances of these features into core MoTMoT elements;
- We implemented the *folding algorithm* as accurately as possible. This enabled us to support both structured and quasi-structured BPMN models. To transform acyclic graph BPMN models an intermediate formal representation is necessary to ensure soundness and safeness. Due to time constraints we do not support acyclic graph structures. In our implementation, we assume that all the input models are sound and safe. However, we can easily derive formal Petrinets from BPMN models to derive soundness and safeness properties;
- *Flexible meta-model developm*ent: extra BPMN elements are easily inserted during development of the transformation;
- We managed to follow an *incremental development* approach. This is possible because the different BPMN structures can be seen as rules that can be added independently of each other;
- Because the transformation model is UML compliant, we were able to use the versatile award-winning NoMagic MagicDraw as editing tool.

3 Higher order transformations

MoTMoT reads in a MOF-compliant model in XMI format into MDR. Within MDR the model is manipulated and in this case the transformation from BPMN to BPEL is performed. The model is then exported in XMI format.

HOTs can be used to implement new language features into a transformation language. In this case, HOTs take a transformation model conforming to the extended transformation language as input model, and transform it to a transformation model conforming to the core language, while preserving the semantics. This is possible if the metamodel of the transformation model is available. This is the case for the language of MoTMoT, as it is defined as a UML profile. The HOT transforms instances of new language features to their equivalent representation using only core elements.

In previous case studies, we already identified the usefulness of following features and implemented them as HOTs [3]:

- Negative application conditions (NACs): There are patterns where we explicitly look for the *absence* of an element or a subpattern within a particular subgraph. The use of NACs provides the answer: matching the pattern including the elements flagged for a NAC results in a negative match. Positive matches whereupon the check of the flagged elements fails are passed through;

- Non-deterministic scheduling of graph transformation rules: There are times when the use of a strict imperative control flow is cumbersome and constraining. This is often the case where rules are equivalent to each other and the order of execution is not relevant. Non-deterministic rule scheduling enables us to model a more clear and concise transformation for this case study. We elaborate more on this in the next Section.

4 Implementation of the case study

4.1 Overview

The BPMN and BPEL models are provided in XML format. However, MoTMoT cannot parse these models immediately. For this reason we have to introduce translations into a representation of these models in MDR.

When outlining the transformation implementing this case study, these steps are followed:

- Input translation: For the BPMN model as input, we provided a translation from XML into a MOF-compliant model in XMI. We have parsed the XML through a DOM parser and using analysis of the XML tree generated the corresponding elements in MOF format;

- The MoTMoT transformation from BPMN to BPEL is performed;

- Output translation: Generation of the XML code is done through the use of a DOM parser and an XML writer in BPEL format.

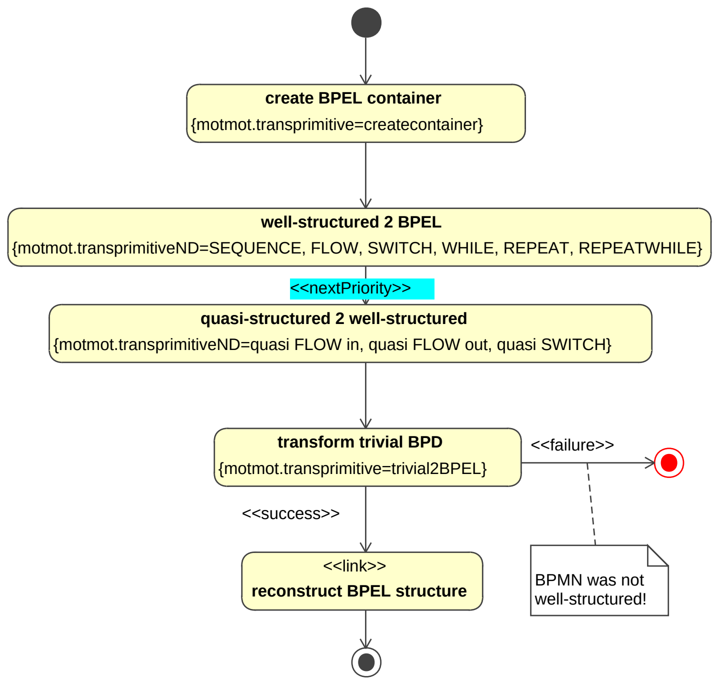
create BPEL container
{motmot.transprimitive=createcontainer}
well-structured 2 BPEL
{motmot.transprimitiveND=SEQUENCE, FLOW, SWITCH, WHILE, REPEAT, REPEATWHILE}
<<nextPriority>>
quasi-structured 2 well-structured
{motmot.transprimitiveND=quasi FLOW in, quasi FLOW out, quasi SWITCH}
transform trivial BPD
{motmot.transprimitive=trivial2BPEL}
<<failure>>
<<success>>
BPMN was not well-structured!
<<link>>
reconstruct BPEL structure

Figure 1: BPMN2BPEL flow

4.2 Meta-models for BPMN and BPEL

We extend the meta-models of BPMN and BPEL for two reasons:

- We introduced an additional meta-element in the BPMN meta-model, called *CompositeTask*, which inherits from *Task*. A *CompositeTask* represents a folded subgraph and can be used in subsequent foldings by other patterns;
- We introduced a mechanism for traceability. All used meta-elements inherit from a class called *GenericNode* which has *traceability links* to other *GenericNode*s. This means that traceability links between the BPMN elements and their corresponding BPEL representation are possible.

4.3 Main flow

In MoTMoT, Story Driven Modeling (SDM) implements graph transformations in an imperative control flow [2]. The following steps can be identified in the MoTMoT transformation (see Figure 1):

1. The first step is the creation of the BPEL container. This container will be the top node of the BPEL tree;

2. The next step is the application of the different patterns in the folding algorithm for well-structured and quasi-structured BPMN models:

    (a) On the one hand, the 6 well-structured BPMN patterns to BPEL are considered. There is no particular order in these rules. Consequently, scheduling them explicitly would be a case of overspecification. This would clutter the transformation flow. In other words, they should be scheduled non-deterministically. What we want to model is the following: "Keep evaluating these patterns until none of them match anymore". Note that when using the folding algorithm, for example, at some point the *sequence* pattern might mismatch, but after matching the *flow* pattern the *sequence* pattern might match once again for the folded graph. Therefore these rules are best scheduled non-deter-ministically. Note that *sequence, flow* and *switch* are links to control flows where each has a different set of rules to match. On the other hand *while, repeatwhile* and *repeat*;

    (b) The quasi-structured BPMN patterns are also scheduled non-deterministically, but they have a lower priority than the previous 6 patterns. This is denoted by the <<nextPriority>> transition. This means that a quasi-structured pattern can only be evaluated if all well-structured patterns fail to match. Consequently, for example, if a *quasi-switch* pattern matches, the next pattern that matches will be a *switch* well-structured pattern;

3. In the third step we transform BPMN containing only a trivial BPD. However, if the algorithm was not able to completely fold the BPMN model, this pattern will fail. Using such a condition highlights the usefulness of explicit rule scheduling support;

4. In the last step, the BPEL tree is reconstructed from the different pattern applications in the algorithm. Reconstruction is done using the traceability links created during the folding algorithm.

4.4 A pattern

The control flow determines which graph transformation patterns are matched. Each of these patterns is called a *story pattern* in SDM. We highlight one of these story patterns, more specifically in the case of creating *sequence* BPEL elements. We summarize the creation of *sequence* in Figure 2:

1. We try to fold a series of BPMN tasks into a single composite task. All tasks which form a series between two non-task elements are part of a single BPEL sequence. The matching algorithm is incremental: a sequence appends an activity one by one until no matches are found. In step 1, in Figure 2, *ct1* and *t*2 match the sequence pattern. It turns out that *ct1* is a pre-existing *CompositeTask* with a sequence attached through a

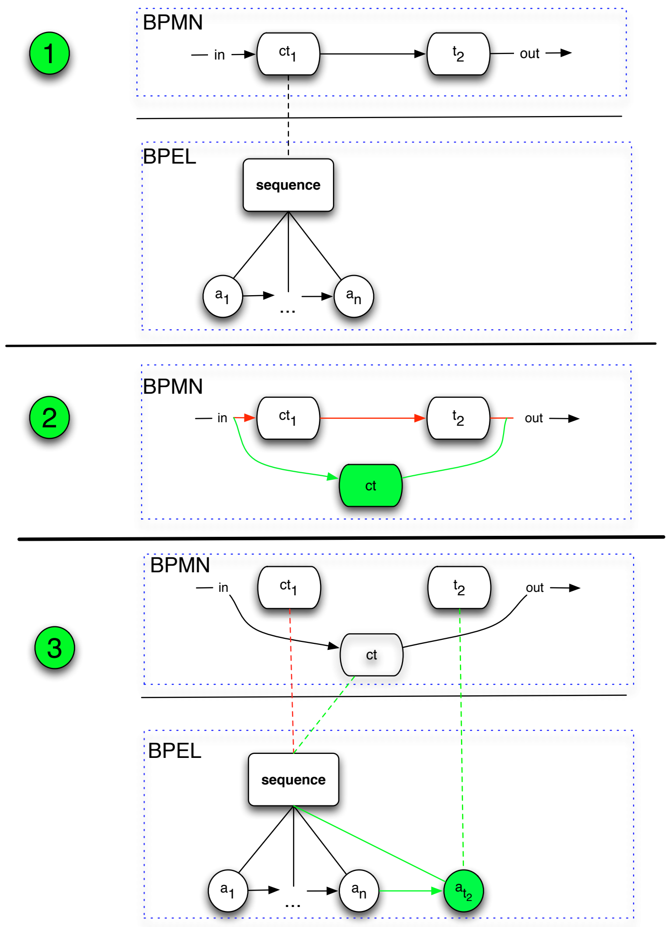

Figure 2: Overview of sequence transformation flow

traceability link (these links are denoted by dashed lines). As opposed to *ct1*, *t2* is not attached to any BPEL construct with a traceability link;

2. The actual folding is done by creating a new *CompositeTask ct* and reassigning the incoming and outgoing BPMN *SequenceFlows.*;

3. In this step we create a new BPEL *SimpleActivi*ty representing *t2*. Because in this case there already was an existing *sequence*, the new element is added to it. Traceability links are created between *t2* and *at2*. This is important for reconstructing the BPEL structure: after all, *t2* might also be a *CompositeTask* representing another BPEL construct such as *flow,* ....

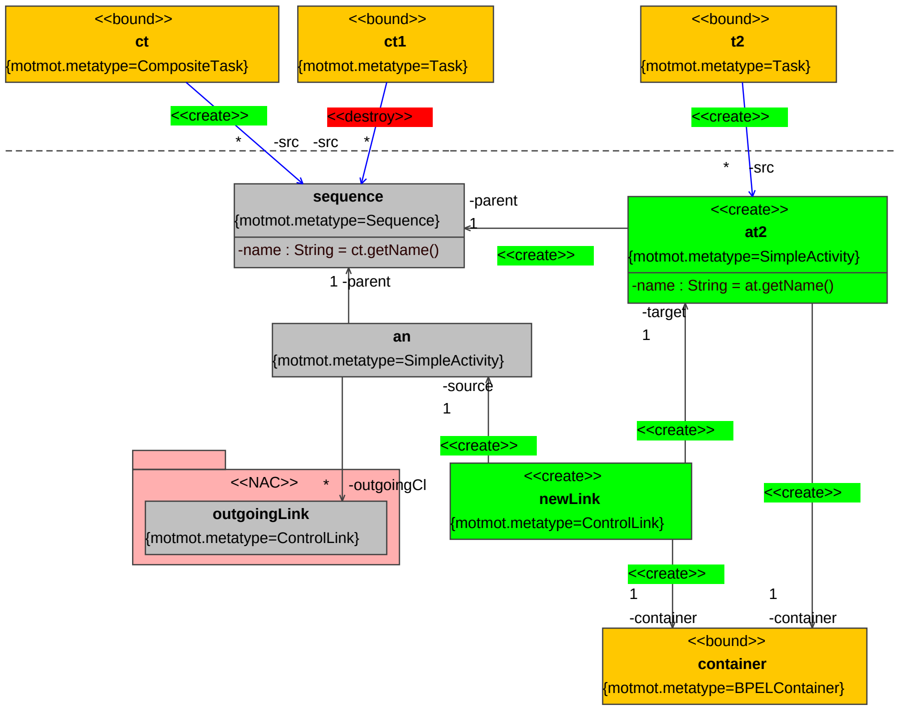

Figure 3: Attaching a new activity to a BPEL *sequence* in SDM

The MoTMoT implementation of step 3 is shown in Figure 3. The Figure shows the formal abstract syntax of the *story pattern* (as opposed to the concrete synta in Figure 2). We made use of a NAC to match the last node of the sequence (the *SimpleActivity* called *an*). The last node is the node with no outgoing *ControlLinks*. Just like non-determinism, NACs are also implemented using HOTs. While non-determinism is used at the control flow level (activity diagrams), NACs are used at the *story pattern* level (class diagrams).

5 Conclusion

We explored the implementation of the BPMN2BPEL case using Higher Order Transformations. This technique enabled us to conceptually envision the transformation of the different BPMN elements in a declarative manner. Using the MDR framework, we followed incremental development stages to model the case study.

First, it was very easy to implement a designated traceability mechanism using MoTMoT for this case study. The meta-model design using MOF is quite flexible and extending the original BPMN and BPEL meta-models to support traceability was easy. The same is true for the *CompositeTask* meta-element. Second, combining non-deterministic rule scheduling with the standard imperative paradigm of SDM we implemented *hybrid* rule scheduling in SDM. Language features for non-deterministic rule scheduling and NACs turned out to be very useful in this case study. We managed to create concise and clear transformation models.

As future work, the following language features were identified as useful for our transformation, but have not been implemented yet:

- OR feature: At the moment MoTMoT demands exact typing on objects. However, sometimes story patterns are valid for a number of different types. The OR feature would find patterns which match for any of the types mentioned, and no longer be limited to one exact type. This prevents duplication of patterns and allows for a more flexible matching mechanism.

- Multi-objects: In a matched pattern, every possible match of the multi-object is matched separately and side effects are executed each time [6].

- Expanded *closure*: At the moment MoTMoT supports closure of elements through associations. For example, a class can follow inheritance links to its children subclasses, subsubclasses, etc. However, this closure is quite limited in scope. An expanded closure would keep identifying identical subpatterns until no other match is found. This is for example useful to identify the subpattern representing two tasks in a sequence. The closure would then continue to match until a non-task element is discovered. In this manner a full sequence of tasks can be identified.

References

[1] Marlon Dumas. Case study: Bpmn to bpel model transformation, April 2009.

[2] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story Diagrams: A New Graph RewriteLanguage Based on the Unified Modeling Language and Java. In *Proceedings of the 6th International Workshop on Theory and Application of Graph Transformation (TAGT)* , volume 1764 of *LNCS*, pages 296–309. Springer Verlag, November 1998.

[3] Bart Meyers and Pieter Van Gorp. Towards a hybrid transformation language: Implicit and explicit rule scheduling in story diagrams. In *Sixth International Fujaba Days (FDÕ08)*, 9 2008.

[4] Chun Ouyang, Marlon Dumas, Stephan Breutel, and Arthur H. M. ter Hofstede. Translating standard process models to bpel. In Eric Dubois and Klaus Pohl, editors, *CAiSE*, volume 4001 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 2006.

[5] Hans Schippers, Pieter Van Gorp, and Dirk Janssens. Leveraging UML profiles to generate plugins from visual model transformations. *Software Evolution through Transformations (SETra). Satellite of the 2nd Intl. Conference on Graph Transformation*, 127(3):5–16, 2004.

[6] A. Zündorf. Rigorous object oriented software development. Habilitation Thesis, University of Paderborn, 2001.