BPMN to BPEL case study solution in VIATRA2

Gábor Bergmann and Ákos Horváth

Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
H-1117 Magyar tudósok krt. 2, Budapest, Hungary
{bergmann, ahorvath}@mit.bme.hu

1 Introduction

Automated model transformations play an important role in modern model-driven system engineering in order to query, derive and manipulate large, industrial models. Since such transformations are frequently integrated to design environments, they need to provide short reaction time to support software engineers.

Graph transformation (GT) [1] based tools have been frequently used for specifying and executing complex model transformations. In GT tools, *graph patterns* capture structural conditions and type constraints in a compact visual way. At execution time, these conditions need to be evaluated by *graph pattern matching*, which aims to retrieve one or all matches of a given pattern to execute a transformation rule.

The objective of the VIATRA2 (VIsual Automated model TRAnsformations [2]) is to support the entire life-cycle, i.e. the specification, design, execution, validation and maintenance of model transformations defined by a combination Abstract State Machine (ASM) and graph transformation rule.

Model description VIATRA2 uses the VPM metamodeling approach [3] for describing modeling languages and models. The main reason for selecting VPM instead of a MOF-based metamodeling approach is that VPM supports arbitrary metalevels in the model space. As a direct consequence, models taken from conceptually different domains (and/or technological spaces) can be easily integrated into the VPM model space. The flexibility of VPM is demonstrated by a large number of already existing model importers accepting the models of different BPM formalisms, UML models of various tools, XSD descriptions, and EMF models.

Transformation description. Specification of model transformations in VIATRA2 combines visual rule and patternbased paradigm of graph transformation (GT) [1] and the very general, high-level formal paradigm of abstract state machines (ASM) [4] into a single framework for capturing transformations within and between modeling languages.

Transformation Execution. Transformations are primarily executed within the framework by using the VIATRA2 interpreter. For pattern matching both (i) *local search based pattern matching* (LS) and (ii) incremental pattern matching (INC) are available. This feature provides the transformation designer additonal opportunities to fine

tune the transformation either for faster exeuction (INC) or lower memory consumption (LS).

The rest of the paper is structured as follows. Sec. 2 gives an overview of the transformation presented in this paper, while Sec. 3 highlights the interesting parts of our implementation and finally Sec. 4 concludes the paper.

2 Overview of the Approach

The model transformation presented in this paper is a solution to the BPMN to BPEL [5] case study of the GraBaTs 2009 [6] tool contest. Both BPEL [7] and BPMN [8] are business process / workflow definition languages with high industrial relevance. While BPEL relies on a strict nested block structure to define its control flow, BPMN basically uses freeform sequence flow edges between its flow objects. This key difference is the main hurdle of the transformation problem.

The implemented workflow of the BPMN to BPEL transformation is summarized in Fig. 1.

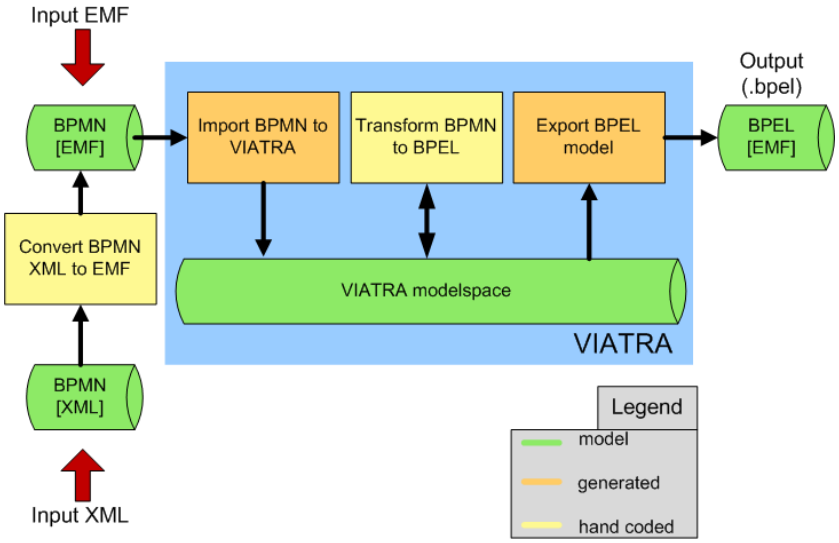

Fig. 1. Overview of the BPMN to BPEL transformation

Input Our approach supports two different input formats: (i) the XML format defined in the case study (depicted by a green cylinder) and (ii) its corresponding EMF [9] based version. As an experimental feature of the VIATRA2 framework the importer for the EMF format was (mostly) automatically generated based on its ecore files. For the XML version we have hand coded a simple parser that translates the XML file to its corresponding EMF model and uses the generated importer for the actual work.

Transformation The transformation itself is implemented as a VIATRA2 transformation program. It consists of 44 graph patterns and 27 ASM rules in approx. 700 lines of code and detailed in Sec.3.

Export Finally, to generate the BPEL source code we relied on the Eclipse BPEL project [10]. We adopted its EMF model and used our exporter generator to create the exporter that builds an EMF-based BPEL model from its VIATRA2 model space representation. As for source code generation, the Eclipse BPEL project serializes its EMF models to BPEL files automatically.

3 Solution

3.1 Overview

We have implemented a partial solution for the BPMN to BPEL transformation problem. Our approach conceptually follows the Structured Activity-based Translation described in Section 4.1 of [11]. This translation procedure involves identifying components of the BPMN graph that adhere to the strict block structure of the BPEL control flow. Identified components are repeatedly *folded* into single nodes so that larger components incorporating them are more easily recognized. If the BPMN model consists solely of BPEL-stlye blocks, the entire process is eventually folded into a single complex activity node, preceded by a Start Event and followed by an End Event. However, not all BPMN models are built this way, thus great care must be taken to enforce the neccessary conditions of well-structured components.

The transformation consist of two phases: (i) identifying well-structured components of the BPMN workflow through iterated folding as described above, and (ii) building a BPEL process model based on the identified and folded components. Our implementation is currently capable of identifying, folding, and translating the following well-structured components:

- Linear sequences of regular flow objects, referred to as *sequence components*. Tasks, intermediate events and folded components are considered regular here, while gateways are excluded.
- Fork/Join parallelism of regular nodes, referred to as a *flow component*.
- Decision/Merge branching of regular flow objects, referred to as a *switch component*.
- Three different flavors of looping regular nodes, referred to as *while, repeat, repeat-while components*.

One notable divergence from the algorithm presented in [11] is using the *If* activity of BPEL 2.0 instead of a *Switch*. More importantly, there is a difference in the handling of nested sequence components. To achieve better readability of the resulting BPEL model, it is recommended to avoid the nesting of sequences. The original solution achieves this by folding only so-called maximal sequence components, no extension of which is a sequence component itself. Due to the folding of other types of components, however, a sequence component that was previously considered maximal may

become prefixable / postfixable by newly created substitute tasks, and therefore nested sequences may still appear in the output. Our approach does restrict sequence folding to maximal sequence components, but flattens the hierarchy of folded sequences during the translation to BPEL instead.

3.2 Program structure

The transformation program achieving these goals consists of the following notable parts:

- Entry point and main control of the program, to initialize the trace model and invoke first the folding, then the actual transformation rules.
- Patterns that identify foldable components corresponding to one of the structured activites of BPEL. At least one complex pattern is needed for each component type, as well as and some auxiliary patterns to help enforcing the criteria associated with the component type.
- Rules for each structured component type to fold an individual identified component into a single *substitute task*, and weave the newly created node back into the BPMN workflow.
- A folding control rule to supervise the behaviour described in the previous two bullets.
- Rules translating a completely folded, well-structured BPMN model into a BPEL model. The translation is fairly straightforward, and carried out in top-down fashion. A separate translation rule is required for each regular BPMN flow object type, and for the substitute tasks of each structured activity component type.
- Patterns to direct these rules in the top-down traversal of the modified BPMN model.

The transformation makes use of a trace model. The first role of the trace model is to contain the substitute tasks representing folded componenents of BPMN, and to maintain the mapping between these representative nodes and the original constituents of the component. The second role of the trace model is to establish correspondence between the source BPMN processes and the target BPEL processes; while this mapping is not strictly needed during the runtime of the transformation, it may be useful afterwards. While not implemented, this aspect of trace information could be extended to provide deep traceability between BPEL contructs and elements of the folded BPMN process. A simplified version of the trace metamodel is depicted in Figure 2.

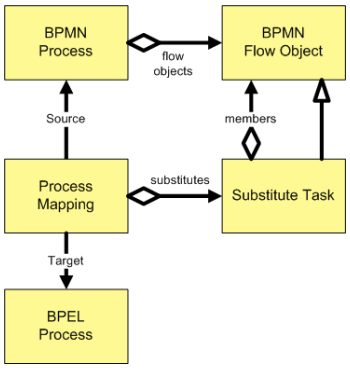

Fig. 2. Simplified trace metamodel

3.3 Missing features

While our solution is useful to showcase VIATRA2, it is not meant to be a feature-complete product. Naturally, there are several shortcomings. Most importantly, only

the first algorithm presented in [11] is implemented, and therefore only those BPMN models can be transformed that consist of well-structured components corresponding to BPEL activies. For the sake of simplicity, we have also omitted folding support for one kind of Decision/Merge construct, the event-based *pick component*. In either case, the transformation reports that it was unable to fold a specific BPMN process.

As the explicit focus of the case study was the transformation of workflow *structure* as opposed to condition, event and task mappings, our solution merely copies the ID of tasks. Edge guards are also ignored, except on branches of a switch component. Conditional branches, however, are supposed to be ordered; as the input format specification of the case study does not clearly indicate this ordering, our solution makes no guarantees on correctly preserving the condition evaluation order.

It is also important to point out that no reverse transformation was implemented.

4 Conclusion

In the current paper we have presented our VIATRA2 based implementation for the BPMN-to-BPEL case study [5]. Due to the lack of resources we implemented only a partial solution. However, we believe that the complete implementation is possible in the VIATRA2framework if sufficient time is invested into the implementation.

References

1. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook on Graph Grammars and Computing by Graph Transformation. Volume 2: Applications, Languages and Tools. World Scientific (1999)
2. VIATRA2 Framework: An eclipse GMT subproject (http://www.eclipse.org/gmt/)
3. Varró, D., Pataricza, A.: VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML. Journal of Software and Systems Modeling **2**(3) (October 2003) 187–210
4. Börger, E., Stärk, R.: Abstract State Machines. A method for High-Level System Design and Analysis. Springer-Verlag (2003)
5. Marlon Dumas: Case study: Bpmn to bpel model transformation, grabats 2009 (2009) `http://is.tm.tue.nl/staff/pvgorp/events/grabats2009/cases/grabats2009sy%nthesis.pdf`.
6. GraBaTs - Graph-Based Tools: The Contest: official website (2009) `http://is.tm.tue.nl/staff/pvgorp/events/grabats2009`.
7. OASIS Standard: Web Services Business Process Execution Language version 2.0. (2007) `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`.
8. OMG Specification: Business Process Modeling Notation version 1.2. (2009) `http://www.omg.org/spec/BPMN/1.2`.
9. The Eclipse Modeling Framework project: http://www.eclipse.org/emf/
10. Moser, S., Chmielewski, M.: The eclipse BPEL project, http://www.eclipse.org/bpel/
11. Ouyang, C., van der Aalst, W.M., Dumas, M., ter Hofstede, A.H.: From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way. http://eprints.qut.edu.au/5266/ (2006)