Transforming BPMN to BPEL with EMF Tiger

Article · January 2009

CITATIONS
4

READS
137

2 authors, including:

Claudia Ermel
Technische Universität Berlin
137 PUBLICATIONS **1,706** CITATIONS

SEE PROFILE

Transforming BPMN to BPEL with EMF Tiger

Enrico Biermann and Claudia Ermel

Technische Universität Berlin, Germany
{enrico,lieske}@cs.tu-berlin.de

Abstract. This paper describes a model transformation from the Business Process Modeling Notation (BPMN) to the Business Process Execution Language for Web Services (BPEL4WS, or BPEL for short). We give the meta-models for both source and target language as EMF models and define EMF model transformation rules using our recently developed tool EMF Tiger, an Eclipse plug-in supporting modeling and execution for EMF model transformations, based on structured data models and graph transformation concepts.

1 Introduction

In this paper we describe a model transformation from the Business Process Modeling Notation (BPMN) to the Business Process Execution Language for Web Services (BPEL) according to the translation by van der Aalst et al. in [1]. Our meta-models of the source language BPMN and the target language BPEL are EMF model, i.e. they conform to the meta-modeling quasi-standard of the Eclipse Modeling Framework EMF [2]. In order to define EMF transformation rules, relations between source and target EMF models are given in a third EMF model, the so-called reference model. Our transformation approach is based on graph transformation concepts which are lifted to EMF model transformation by also taking containment relations in meta-models into account. Our recently developed tool, EMF Tiger [3,4] (**T**ransformat**I**on **GE**ne**R**ation) is an Eclipse plug-in supporting modeling and execution for EMF model transformations.

For modeling the case study, we made extensive use of the various control structures offered by EMF Tiger, e.g. constructs for non-deterministic rule choices or rule priority. Those constructs may be nested arbitrarily to define more complex control structures. Passing of model elements and parameters from one rule to another is also possible by using input and output parameters. For EMF transformations with multi-object structures, a special control unit exists which represents a basic interaction scheme consisting of one kernel and one multi-rule. The synchronization of rules along kernel rules forces a transformation step to be maximally parallel in the following sense: an amalgamated rule, induced by a scheme, is constructed by multi-rules being synchronized at the kernel rule. The number of multi-rule instances is determined by the number of different matches found such that they overlap in the match of the kernel rule. Hence, transforming multi-object structures can be described in a general way although the number of actually occurring objects in the instance model is variable.

Our model transformation is restricted to well-structured business process models (BPMs) without generating BPEL model elements of types *flow, pick* and *switch*.

2 EMF Tiger

EMF transformation rule applications in EMF Tiger change an EMF model instance in-place, i.e. an EMF instance model is modified directly, without copying it before. Moreover, control of rule applications is supported by EMF Tiger, as well as pre-definition of (parts of) the match. EMF Tiger currently consists of a *graphical editor* for visually defining EMF model transformation rules and a transformation engine which executes the defined rules. The transformation engine provides classes which can freely be integrated into existing projects which rely on EMF models. Another advantage is that EMF Tiger directly operates on those models. Moreover it is possible to define control structures over defined rules.

Currently there exist two implementations of the transformation engine. One is written in Java while the other translates the transformation rules to Agg [5,6]. This is useful for validation of consistent EMF model transformations which behave like algebraic graph transformations, e.g. to show functional behavior and correctness.

Fig. 1 shows a screenshot of the EMF Tiger meta model. It can be seen that the objects of graphs namely *nodes*, *edges* and *attributes* are typed over *EClass*, *EReference* and *EAttribute* which are classes from the *Ecore* meta model. It can also be seen how the transformation units are structured.

EMF Tiger rules and transformation units can be used in other projects by instanciating the class *GenericRule* or *GenericUnit*, respectively. The class *GenericRule* requires a *Rule* instance from the EMF Tiger meta model. Once instanciated, the rule can be applied by calling the *execute()*-method of *GenericRule*. Transformation units can be executed in a similiar way by using the class *GenericUnit*.

3 BPMN and BPEL models

In order to be able to access the transformation features of EMF Tiger we define the BPMN and BPEL meta models in EMF. In addition to the original BPMN meta-model in [1], the BPMN meta-model in EMF, shown in Fig.2, features a root container node called *BPMNRoot* that directly contains all flow objects and sequence flows.

For the BPEL EMF model there also exists an additional root container node which is called *BPELRoot* (shown in Fig.3). Moreover, the parent-child relation on activities has been modeled as a containment relation. This leads to a structure similar to the XML representation of the BPEL language in [1].

At last, we have the reference meta model shown in Fig. 4 that allows us to define relations between BPMN flow objects and BPEL activities via *F2ARef*

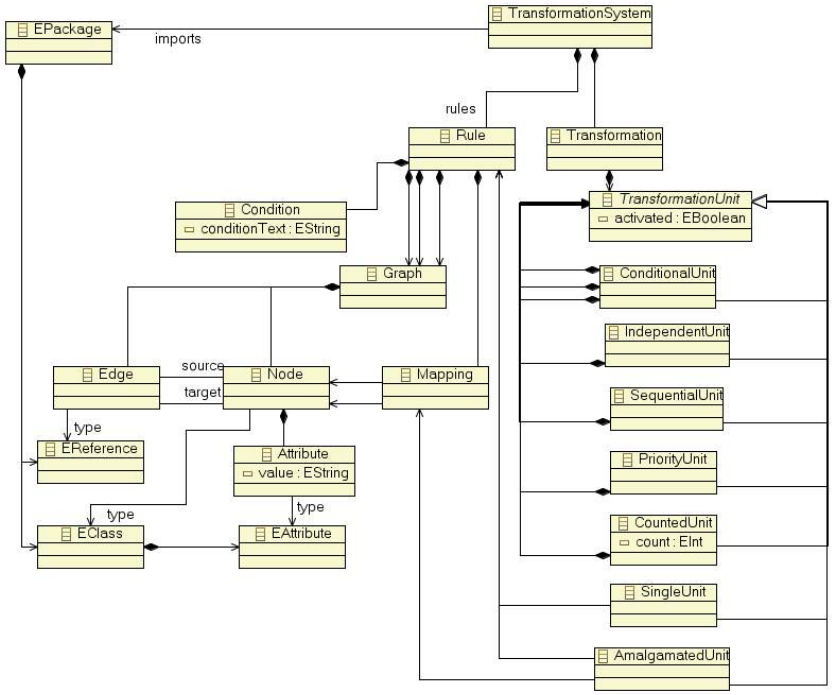

Fig. 1. EMF Tiger meta model

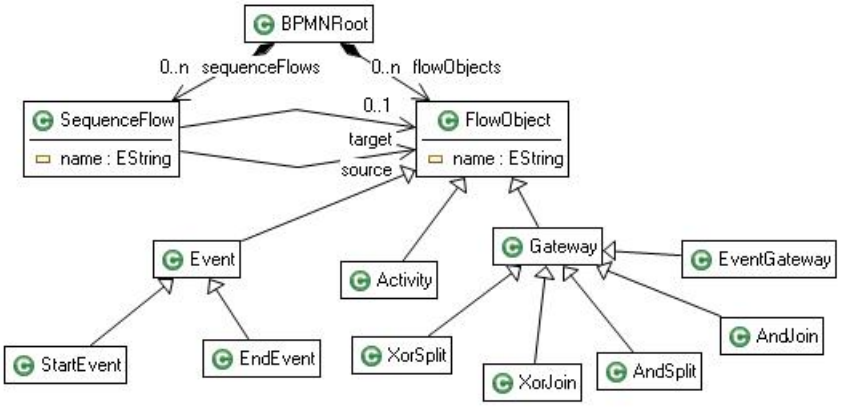

Fig. 2. BPMN meta model

nodes. *Copy Marker* nodes are used to copy all children of one BPEL activity to another which is necessary when translating *repeat* or *while-repeat* constructs.

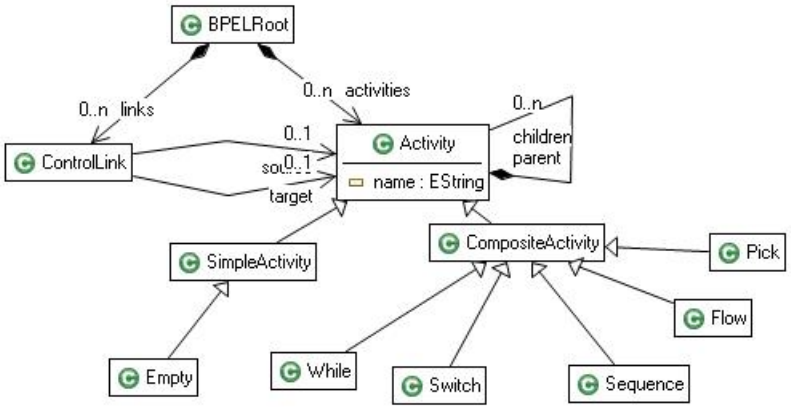

Fig. 3. BPEL meta model

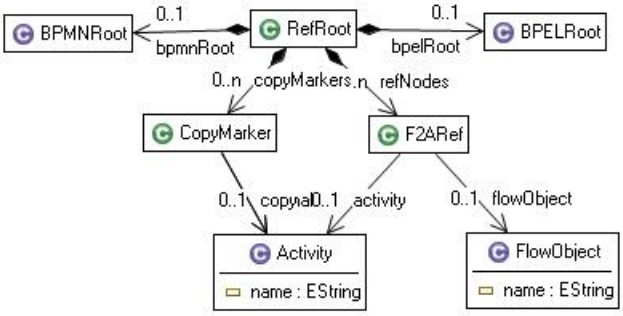

Fig. 4. reference meta model

4 Transformation rules

Basically the transformation from BPMN to BPEL via EMF Tiger is implemented in two steps:

1. translate all simple activities from BPMN to BPEL using rule *SimpleActivityRule* shown in Fig. 5,
2. collapse all complex structures to simpler activities in the source model while creating the corresponding structure in the target model, e.g. *sequences*, *while*, *repeat* or *while/repeat* structures.

While translating structures from BPMN to BPEL, nodes in the reference model are created. Those nodes connect source model activities to elements in the target model. The following sections describe the translation of complex structures.

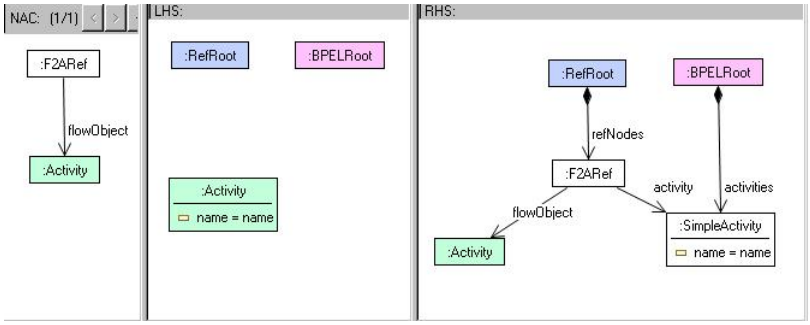

Fig. 5. SimpleActivityRule

4.1 Translating sequences

The translation of a sequences is done by two rules, shown in Fig. 6: rule *Sequence* creates a new sequence in the BPEL model and inserts the current activity into this sequence. Rule *Sequence2* deals with the case that the sequence has already been created in the BPEL model. Then, the current activity is added to the existing sequence.

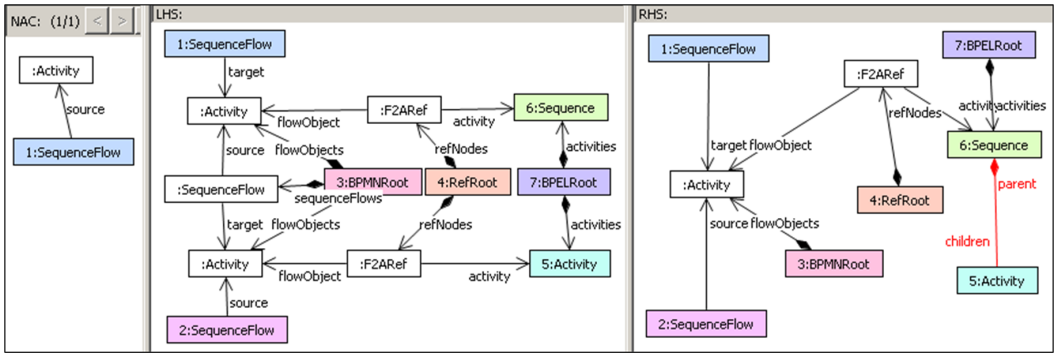

Fig. 6. Rules *Sequence* (top) and *Sequence2* (bottom)

4.2 Translating *While* constructs

A *While* construct is translated to a *While* container node which contains the activity (see Fig. 7).

4.3 Translating *Repeat* constructs

A special challenge when using EMF Tiger for the transformation are *Repeat* and *Repeat/While* constructs. Those two constructs need the ability to clone

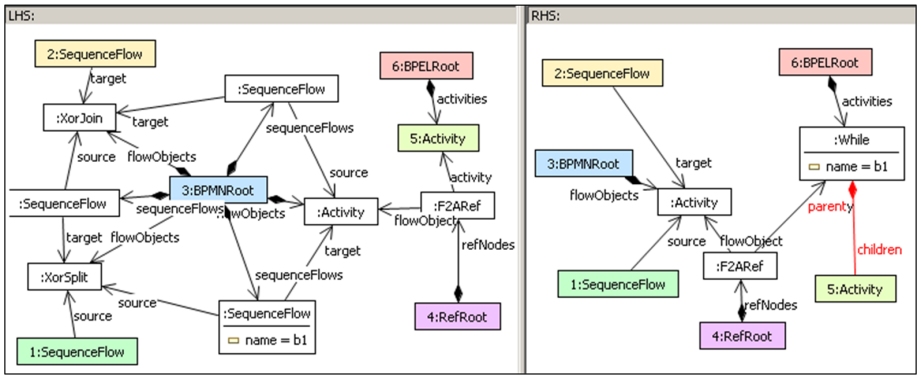

Fig. 7. Rule *While*

subgraphs because one graph structure in the left-hand side of a rule occurs multiple times in the right-hand side. Since this is not possible with the double-pushout approach, we needed to define rule schemes that copy the content of one BPEL subtree to another.

A *Repeat* construct corresponds to executing an activity and afterwards executing a While-loop with the same activity as body. Hence, a *Repeat* construct is translated to a Sequence which contains the activity and a *While* node containing a copy of this activity (see rule *Repeat* in Fig. 8).

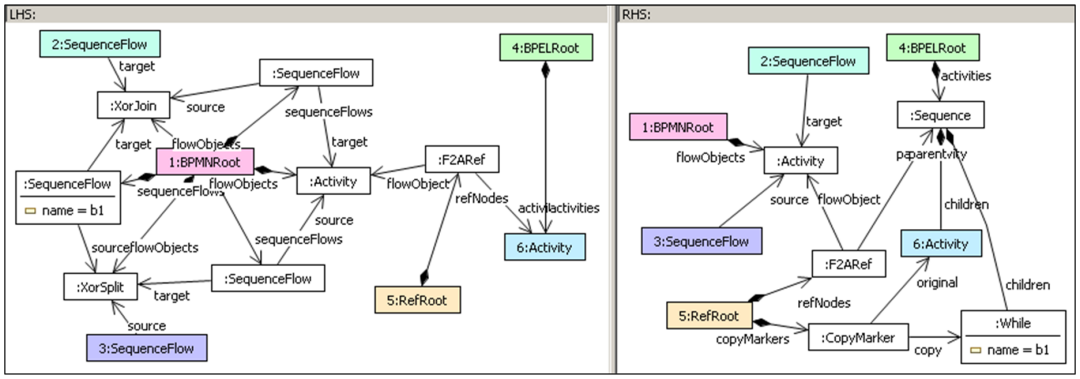

Fig. 8. Rule *Repeat*

If the activity itself is a sequence, then all activities in this sequence are copied to the body of the While-loop, i.e. inserted into the While container. We realize this by rule scheme *CopySequenceContents* (at the top of Fig. 9), where all activities in a sequence are copied to the While-construct in one step.

For rule schemes, we use an integrated notation, where we define the kernel rule and a multi-rule within one rule picture. We distinguish objects belonging to the multi-rule by drawing them as multi-objects (with indicated multiple boxes instead of simple rectangles). The kernel rule consists of all simple objects which are not drawn as multiple boxes. All arcs adjacent to multi-objects belong to the multi-rule only, but not to the kernel rule. When applying a rule scheme, an amalgamated rule, induced by the scheme, is constructed by multi-rule instances being synchronized at the kernel rule. The number of multi-rule instances is determined by the number of different matches found such that they overlap in the match of the kernel rule. Thus, a maximally parallel transformation step is enforced.

Analogously, for copying simple activities and While constructs we provide rules CopySimple (at the center of Fig. 9) and CopyWhile (at the bottom of Fig. 9).

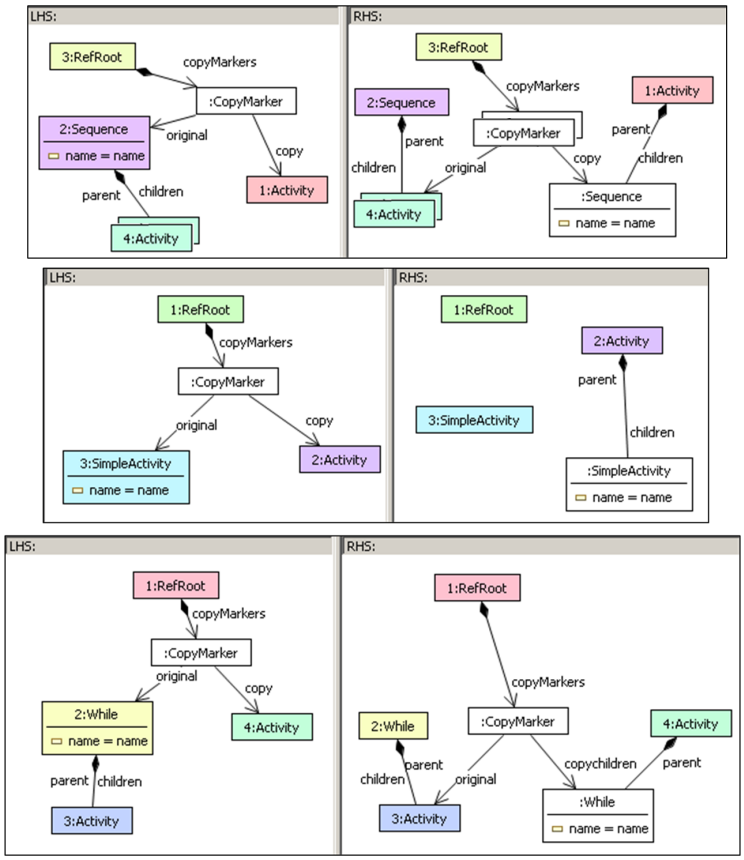

Fig. 9. Rule scheme *CopySequenceContents* (top), rule *CopySimple* (center), and rule *CopyWhile* (bottom)

4.4 Translating *Repeat-While* constructs

Repeat-While constructs are translated in a similar way. Again, in rule RepeatWhile (at the top of Fig. 10), a *Sequence* containing the *Activity* and a *While* Construct is inserted. A *Copy Marker* is used as auxiliary structure to help inserting the items from a sequence corresponding to the abstract *Activity* (Rule Scheme CopyWhile at the bottom of Fig. 10).

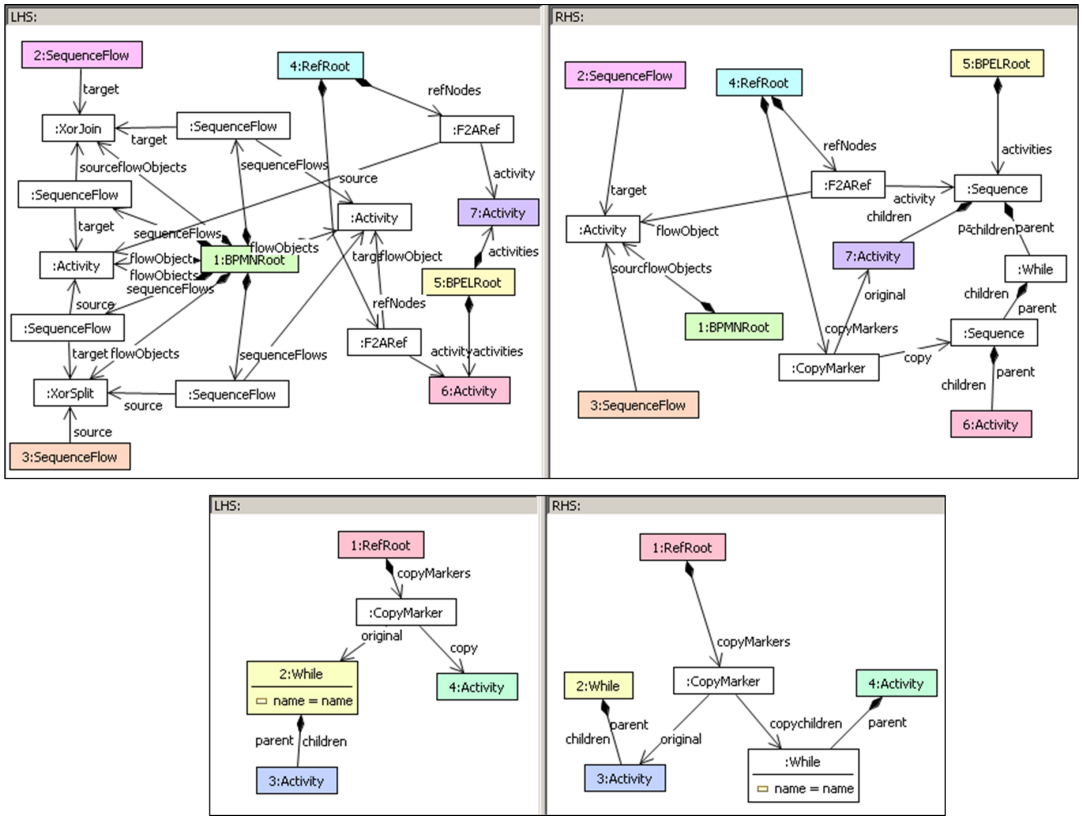

Fig. 10. Rule *RepeatWhile* (top) and rule scheme *CopyWhile* (bottom)

5 Conclusion

We presented a transformation from BPMN models to the language BPEL using the EMF transformation tool EMF Tiger. Up to now, we restricted our model transformation to the simplest case of structured BPMs. The current state of our model transformation is provided via SHARE at `http://is.tm.tue.nl/staff/`

pvgorp/share/?page=handleNewSessionRequest&vdi=XP_GB9_emftiger-bpmn.vdi. Using control structures and rule schemes helped a lot to define a straightforward translation algorithm without backtracking. An extension of model transformation to model also the translation of quasi-structured BPMs seems possible, especially when using rule schemes to model alternatives with more than two branches. Moreover, including also model elements like *pick, switch* or *flow* components seems to be a straightforward extension. Here, also the EMF models as basis of the underlying languages have to be extended.

References

1. Ouyang, C., Dumas, M., ter Hofstede, A.H.M., van der Aalst, W.M.P.: From bpmn process models to bpel web services. In: IEEE International Conference on Web Services (ICWS 2006), 18-22 September 2006, Chicago, Illinois, USA, IEEE Computer Society (2006) 285–292
2. Eclipse Consortium: Eclipse Modeling Framework (EMF) – Version 2.4. (2008) http://www.eclipse.org/emf.
3. Tiger Project Team, Technische Universität Berlin: EMF Tiger (2009) http://tfs.cs.tu-berlin.de/emftrans.
4. Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., Weiss, E.: Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In Nierstrasz, O., Whittle, J., Harel, D., Reggio, G., eds.: Proc. 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06). Volume 4199 of lncs., springer (2006) 425–439
5. TFS-Group, TU Berlin: AGG. (2009) http://tfs.cs.tu-berlin.de/agg.
6. Taentzer, G.: AGG: A Graph Transformation Environment for Modeling and Validation of Software. In Pfaltz, J., Nagl, M., Boehlen, B., eds.: Application of Graph Transformations with Industrial Relevance (AGTIVE'03). Volume 3062 of LNCS. Springer (2004) 446 – 456