From Business Process Models to Web Services Orchestration: The Case of UML 2.0 Activity Diagram to BPEL

Man Zhang and Zhenhua Duan

Institute of Computer Theory & Technology, Xidian University,
Xi'An, 710071, P.R. China
zhangman705@gmail.com, zhhduan@mail.xidian.edu.cn

Abstract. The Business Process Execution Language for Web Services (BPEL) has emerged as the de facto standard for implementing business processes. At the same time, Model Driven Architecture (MDA) is being applied to the field of business process engineering by separating business logic from the underlying platform technology. However, due to the challenge of mapping graph-oriented modeling languages to block-structured ones and the informal description of UML 2.0 Activity Diagram (AD) and BPEL, transforming AD models to executable BPEL code is not trivial. This paper proposes an approach to transform AD to BPEL and paves the way for further general transformation between graph-oriented and block-structured process modeling languages.

1 Introduction

The Business Process Execution Language for Web Services (BPEL) [1] has emerged as the de facto standard for implementing business processes. At the same time, Business Process Management (BPM) has made traditional process-aware information systems completely distributed, global and closely integrated with Web services. It can be concluded that in the near future a wide variety of process-aware information systems will be realized using BPEL. On the other hand, Model Driven Architecture (MDA) is being applied to the field of business process engineering by separating business logic from the underlying platform technique. The core paradigm of MDA is the model transformation from Platform-Independent Models (PIM) to Platform-Specific Models (PSM). Particularly, in BPM's perspective, PIM is established through business process modeling languages while PSM is represented as runtime platform executable code, such as BPEL.

Complete business process models consist of a process model to describe the execution logic, an information model for the data types, an organizational model with some involved roles, and possibly other models [2]. In this paper, we focus on the execution logic, i.e. a model's control flow.

Graph-oriented BPM languages have become mature and been utilized by many existing systems. Among them, UML 2.0 Activity Diagram (AD) [3] has attained wide adoption and is supported by abundant software tools. There have been some research work on transforming AD into executable BPEL code [2,4,5,7,8,9]. However, they are suffering from the following limitations:

1) They need human intervention to identify activity patterns in AD models. Some approaches define UML profiles specific for structured activities in BPEL. For instance, UML 2.0 profile for BPEL proposed in [4] enables the graphical representation of BPEL with UML 2.0. However, it uses heavily stereotypes and requires quite amount of manual work. Another UML 2.0 based mapping of mobile processes to executable BPEL described in [5] can automatically identify two structured activity patterns, but for others it still needs human intervention.

2) They are not applicable to AD models with arbitrary topology. Most of such methods can handle only a limited subset of AD. Various restrictions are imposed on the structures of models, of which the most restrictive one is *structured limitation* [6]. Techniques for translating unstructured flowcharts into structured ones have been used to translate unstructured process diagrams into equivalent structured ones in [2,7,8]. However, these methods only address a piece of the puzzle of the transformation. Once parallel split and synchronization are involved these techniques are not efficient.

3) They do not make full use of two modeling structures of BPEL: block-structured and graph-based ones. Mendling et al. summarized four strategies of transformation of graph-oriented process modeling languages into BPEL in [9]: the first and second ones transform acyclic models by relying intensively on flow and control links, and the third one identifies block-structured patterns and folds them incrementally. Although the fourth strategy tries to use both of them, no concrete algorithms are presented. In addition, all of the four strategies have restrictions on the structures of source models.

These limitations are not surprising since transforming AD models to BPEL code is not trivial work. First, AD and BPEL belong to two fundamentally different classes of languages: AD is graph-oriented, allowing links between nodes in arbitrary topology, while BPEL is mainly block-structured; Secondly, model transformation of full AD requires formal semantics to express its meaning with greater precision than it is available today. Similarly, formalization of BPEL is also in the initial state of development [7].

In this paper, we propose a novel transformation to achieve completeness, automation, and full exploitation of the two modeling structures of BPEL under some conditions. Here completeness means that any meaningful AD can be transformed into BPEL code. It should be noted that our AD models only capture the basic control flow patterns defined in [10] in order to avoid the subtleties of full AD. The automation is realized by decomposing an AD model into regions and identifying structural patterns separately. By identifying block-structured patterns as often as possible, the readability of generated BPEL code is improved. With regards to the BPEL behavior model, we simplified the BPEL specification to contain only those elements needed to describe the execution logic extracted from the process model.

This paper is organized as follows. Section 2 introduces the subset of full AD we can handle as the basis of our process modeling language. Section 3 describes the transformation technique in detail. Finally conclusion and future work are outlined in the last section.

2 Semantically Sound AD Models

Here, we select a subset of AD's meta-model, in which unnecessary elements for modeling the control flow have been removed. According to the meta-model, an AD model consists of ControlFlows, ExecutableNodes (which refer to the necessary basic activities in BPEL), and ControlNodes. Six types of ControlNodes are distinguished: InitialNode, FinalNode, DecisionNode, MergeNode, ForkNode and JoinNode. Here the semantics of all the elements are inherited from the token flow semantics UML 2.0 adopts [3]. Furthermore, we impose some restrictions: DecisionNodes can only start a single flow and MergeNodes always have only one active flow arriving; ForkNodes must start all its concurrent flows and JoinNodes can only be triggered by the arrival of all its incoming flows. These restrictions imply that our AD models only capture the basic six control flow patterns, as mentioned above. In the following, without loss of generality, we assume that all AD models are structurally sound, i.e., they contain exactly one InitialNode and one FinalNode and for every node, there is a path from the InitialNode to the FinalNode going through this node.

When trying to achieve completeness, we should be convinced that transforming process models which cannot be executed normally in real world is meaningless. Based on this point, we introduce the property "semantically sound".

Definition 1 Semantically sound. *A structurally sound AD model is semantically sound if and only if all its possible executions terminate successfully. An execution terminates successfully if no other tokens are present in the process model as soon as the FinalNode consumes a token* [11].

To capture the common structural characteristics of semantically sound AD models, we first induce the concept *single-entry-single-exit(SESE) region*. Informally speaking, an SESE region is a set of nodes and edges such that there is exactly on entry edge entering the region, and exactly one exit edge leaving the region. We can decompose an AD model using the technique of SESE decomposition [12] into SESE regions. For an SESE region R, we represent the regions immediately enclosed in R as the child regions of R. For the decomposition to be unique, every region should be canonical. Roughly speaking, a canonical region requires that if a region contains child regions all in sequence it must contain child regions as many as possible; see cf. [12] for precise definition for SESE and canonical region.

Given the unique decomposition of a semantically sound AD model, every region can fall into one of the following three categories: 1) **sequential region**: a region having no ForkNodes and JoinNodes as its ControlNodes; 2) **parallel region**: a region having no DecisionNodes and MergeNodes as its ControlNodes, and no ControlFlows constructing a cycle; 3) **overlapped region**: we use the

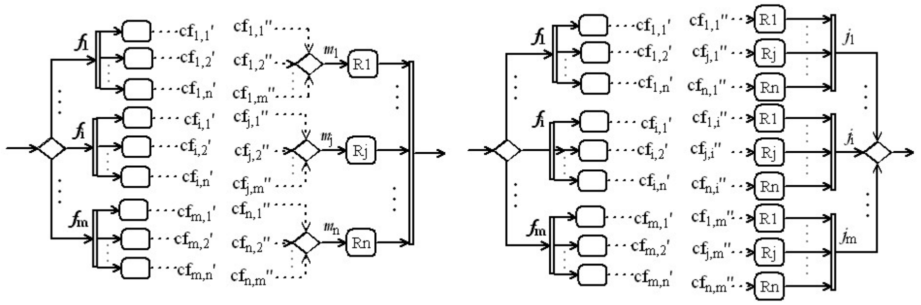

Fig. 1. A pseudo overlapped region and it functionally equivalent structured form

left part of Figure 1 to depict it, which shows a pseudo region whose partial control flows are under determination. It should be noted that when we identify the category of a region, its child regions are treated as ExecutableNodes and their internal structures are out of account.

The round rectangles represent child regions. As the figure shows, there are m ForkNodes and n MergeNodes, denoted by $f_i(i = 1, 2, ..., m)$ and $m_j(j = 1, 2, ..., n)$ respectively. Two types of doted edges exist: edges without an arrow denoted by cf' and edges with an arrow denoted by cf''. Let CF'_i be the set of n outgoing edges of f_i, CF''_i be the set of m incoming edges of m_j, $CF' = \cup_{i=1,2,...,m} CF'_i$ and $CF'' = \cup_{j=1,2,...,n} CF''_j$ .The set of control flows under determination, denoted by CF, are decided by connecting one of edges in CF' to one of edges in CF''. If CF is formed according to a family of bijective functions $g_i : CF'_i \longrightarrow \{cf''_{1,i_1}, cf''_{2,i_2}, ..., cf''_{j,i_j}, ..., cf''_{n,i_n}\}$ $(i = 1, 2, ..., m$, $j = 1, 2, ..., n)$ such that for any g_k and $g_l (1 \leq k \leq m, 1 \leq l \leq m)$ $k_j \neq l_j$, the generated region is overlapped.

3 Pattern Based Transformation from Semantically Sound AD Models to BPEL Code

Given the unique decomposition of a semantically sound AD model, the transformation can be iteratively operated on the model. For a region, the transformation only focuses on it and its child regions. Once its pattern has been identified, the relevant BPEL code for it will be generated, and then the region will be reduced to an abstract node. For the clarity of representation, we call the abstract node medi-node, to which the corresponding BPEL code is attached.

For the readability of generated code, we try to transform as many regions as possible into block-structured BPEL activities, while the remaining regions could be isolated to limited ranges. Therefore we establish four kinds of patterns: structured pattern for block-structured BPEL activities, unstructured sequential, unstructured parallel and overlapped pattern. The last three ones are applied to those regions the first pattern cannot handle. In the following, transformation methods for the four patterns will be explained in turn.

1. Structured pattern. This pattern is defined for those regions that can be suitably mapped to one of the five structured constructs: Sequence, If, Flow, While and RepeatUntil. They all have directly relevant block-structured activities in BPEL 2.0, hence the mappings are straightforward.
2. Unstructured sequential pattern. Those sequential regions which the structured pattern cannot tackle with are handled by this pattern. Informally speaking, this kind of regions contains either improperly nested DecesionNodes and MergeNodes or unstructured cyclic flows, even both. In this paper we use an approach proposed in [13] based on continuation semantics for this pattern. Although this solution is intended to untangle unstructured cyclic flows, study shows that it works well for the unstructured sequential regions without unstructured cycles. With the assistance of continuation semantics, an unstructured sequential region can be mapped to its functionally equivalent BPEL code, which could yield the same output as the original model when provided with the same input data. For space limitation, the detailed algorithm is omitted here.
3. Unstructured parallel pattern. This pattern is applicable to those parallel regions which cannot be transformed to Flow activity using the structured pattern. Informally speaking, they have improperly nested ForkNodes and JoinNodes. We could easily tame them by mapping all its child regions to BPEL activities nested in a Flow activity and mapping all the edges to control links. However, analysts or programmers may be confused with the generated code full of control links owing to lack of readability. In order to generate as few control links as possible, we try to preserve two structured patterns: Sequence and Flow as much as possible.

For space limitation, only the main idea is briefly described. We first generate a link for each edge connecting a ForkNode to a JoinNode (The ForkNode's immediately predecessor acts as the link's source and the JoinNode's immediately successor as its target), and then remove the edge from the region. Afterwards we apply SESE decomposition on the altered region over again. If new child regions are produced, they can be matched to Sequence or Flow pattern.
4. Over-lapped pattern. As the name shows, this pattern is for over-lapped regions. We turn an over-lapped region into its functionally equivalent structured form by duplicating the medi-nodes between MergeNodes and the JoinNodes and switching these two kinds of ControlNodes. Recall the pseudo region in the left part of Figure 1, its functionally equivalent structured region is shown in the right part. Apparently the premise of transformation is that those control flows under determination are formed according to valid connecting functions. Afterwards the new region can be easily handled using the structured pattern. This method is easy to practice but at the expense of code redundancy.

The whole algorithm starts transformation from those regions only having one ExecutableNode by turning them into medi-nodes, and then gradually folds outer regions into medi-nodes by matching them to the four patterns above, until there is only one single medi-node left. At that time the BPEL code attached to it is the desired resulting code.

4 Conclusion and Further Work

In this paper, we presented the transformation of AD process models into executable BPEL code. First, we make our transformation complete by exploiting the characteristic of semantically sound AD models. Secondly, by adopting SESE decomposition and separate pattern based analysis the transformation process is automated. Thirdly, two modeling structures of BPEL are fully utilized in the transformation. Furthermore, we emphasize the readability of generated code throughout the process. In the future, we will try to extend our method to more general control-flow models.

References

1. OASIS: Web Services Business Process Execution Language Version 2.0 (April 11, 2007), `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
2. Hauser, R., Koehler, J.: Compiling Process Graphs into Executable Code. In: Karsai, G., Visser, E. (eds.) GPCE 2004. LNCS, vol. 3286, pp. 317–336. Springer, Heidelberg (2004)
3. OMG: UML 2.0 Superstructure (8/8/05), `http://www.omg.org/cgi-bin/doc?ptc/2004-10-02`
4. Ambühler, T.: UML 2.0 Profile for WS-BPEL with Mapping to WS-BPEL. Universität Stuttgart (2005)
5. Pajunen, L., Ruokonen, A.: Modeling and generating mobile business process. In: Proc. ICWS 2007 (2007)
6. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On Structured Workflow Modelling. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789. Springer, Heidelberg (2000)
7. Koehler, J., Hauser, R., Sendall, S., Wahler, M.: Declarative techniques for model-driven business process integration. IBM Systems Journal 44(1), 47–65 (2005)
8. Zhao, W., Hauser, R., Bhattacharya, K., Bryant, B.R., Cao, F.: Compiling business processes: untangling unstructured loops in irreducible flow graphs. IJWGS 2(1), 68–91 (2006)
9. Mendling, J., Lassen, K.B., Zdun, U.: Transformation strategies between block-oriented and graphoriented process modelling languages. In: Multikonferenz Wirtschaftsinformatik 2006. Band 2, pp. 297–312 (2006)
10. Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, BPMcenter.org (2006)
11. Hauser, R., Friess, M., Küster, J.M., Vanhatalo, J.: An incremental approach to the analysis and transformation of workflows using region trees. IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and reviews 38(3) (May 2008)
12. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through sese decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)
13. Koehler, J., Hauser, R.: Untangling Unstructured Cyclic Flows – A Solution Based on Continuations. In: Meersman, R., Tari, Z. (eds.) OTM 2004. LNCS, vol. 3290, pp. 121–138. Springer, Heidelberg (2004)