Transformation from business process models to BPEL with overlapped patterns involved

Man Zhang
Institute of Computer Theory and Technology,
Xidian University,
Xi’an, 710071, China
and
Software Engineering Institute,
Xidian University,
Xi’an, 710071, China
Email: mzhang@xidian.edu.cn

Zhenhua Duan*
Institute of Computer Theory and Technology,
Xidian University,
Xi’an, 710071, China
Email: zhhduan@mail.xidian.edu.cn
*Corresponding author

Qingshan Li and Hua Chu
Software Engineering Institute,
Xidian University,
Xi’an, 710071, China
Email: qshli@mail.xidian.edu.cn
Email: hchu@mail.xidian.edu.cn

Abstract: The derivation of business process execution language (BPEL) for web services from graph-oriented process models has attained wide focus in the literature. It is a challenging work owing to the fundamental differences between graph-oriented models and BPEL. In this paper, a transformation of activity diagrams (AD) into BPEL is presented, which concentrates on a specific kind of structure in graph-oriented process models called overlapped patterns (OPs). The structures of AD models containing OP are analysed, and an important subclass of OP, first-order OP, is defined. Then in the context of first-order OP, the applicable ranges of two existing transformation strategies of OP are discussed, and a new method is proposed for the cases that neither of them can handle.

Keywords: business process execution language; BPEL; UML activity diagram; business process model; SESE decomposition; overlapped pattern.

Reference to this paper should be made as follows: Zhang, M., Duan, Z., Li, Q. and Chu, H. (2016) ‘Transformation from business process models to BPEL with overlapped patterns involved’, Int. J. High Performance Computing and Networking, Vol. 9, Nos. 1/2, pp.82–92.

Biographical notes: Man Zhang is a Lecturer at the Software Engineering Institute at the Department of Software Engineering of Xidian University in Xi’an. She received her PhD in Computer Application Technology at the Institute of Computer Theory & Technology of Xidian University in June 2012, under the supervision of Dr. Duan. Her research interests include model driven development, business process modelling and service-orient computing.

Zhenhua Duan is a Full Professor of the Department of Computer Sciences and the Chair of the Institute of Computer Theory and Technology at Xidian University. He is a senior member of IEEE. He received two PhD in Computer Science from Newcastle University in 1996 and Sheffield University in 1997. His primary research areas include internet computing, high confidence software theory and technologies, etc.
1 Introduction

The derivation of business process execution language (BPEL) from business process models has attained a certain level of focus in the literature. In fact, it is a challenging work owing to the fundamental differences between graph-oriented models and BPEL. The former is graph-based, which allows links between nodes in arbitrary topology, while the latter is mainly block-oriented (providing a graph-oriented construct with syntactical limitations) and belongs to structural models. Moreover, a full transformation of some graph-oriented models requires formal semantics to express its meaning with greater precision than is available today. Similarly, formalisation of BPEL is also in the initial state of development. All these reasons make the transformation a challenging work.

There has been plenty of research on the transformation of graph-oriented models into BPEL. Existing proposals commonly treat sequential and parallel structures separately. For sequential parts, traditional techniques for translating unstructured flowcharts into structured ones have been used widely. For parallel parts, flow activity of BPEL, together with control links, can tame them easily. However, there exist so-called overlapped patterns (OPs), which fall into neither sequential nor parallel structures but have well-defined behavioural semantics. For the completeness of transformation, this kind of pattern should be taken into account during transformation.

Existing methods for OPs mainly fall into two ways: duplication of related nodes and utilisation of control links of BPEL directly. The first way translates an OP to an equivalent structured form by first duplicating the activities between the merge nodes and the join node and then switching those two kinds of nodes. The other way treats an OP as segments of a flow activity of BPEL. Moreover, the control links method will be incapable owing to the appearance of cycles, and the duplication approach will lose its efficacy once the merge nodes in the OP point to more than one join nodes.

In this paper, we propose a transformation strategy from activity diagrams (AD) models to BPEL focusing on OP. The structures of AD models with OP are analysed, and a subclass of OP, called the first-order OP, is defined. Then the OP-involved transformation into BPEL is particularly discussed. The main contribution of our work is as follows:

1. the OP-involved transformation is discussed specifically and systematically
2. the application ranges of the two existing methods for OP are investigated in the context of the first-order OP respectively
3. a new approach is proposed to deal with the stubborn case neither of the two existing methods could handle.

Since the OP cannot be neglected, this paper makes a step to the completeness of derivation of BPEL from graph-oriented models.

This paper is organised as follows. Related works are discussed in Section 2. Semantically sound AD models and some known results about them are introduced in Section 3. The translation of first-order OP is depicted in Section 4. A case study is presented in Section 5 to illustrate the transformation strategy. Finally, conclusions and future work are given in Section 6.

2 Related work

The notion of OP originally derived from the so-called overlapped rule as a graph reduction rule to detect structural conflicts in acyclic workflows in Sadiq and Orłowska (2000). Then, it was discussed in Kiepuszewski et al. (2000) as the term of ‘overlapping structures’, and a transformation of them into structured forms was illustrated with a sample by duplicating related nodes. Liu and Kumar (2005) followed this strategy in their research on the analysis and taxonomy of unstructured workflows.

Many earlier transforming methods have made topological changes of control flow structures, such as Hauser and Koehler (2004), Koehler and Hauser (2004), Koehler et al. (2005), Zhao et al. (2006, 2007), and Ouyang et al. (2008). These proposals usually treat sequential and parallel structures separately. For sequential parts, traditional techniques of translating unstructured flowcharts into structured ones have been widely used to untangle unstructured cycles (Hauser and Koehler, 2004; Koehler and Hauser, 2004; Koehler et al., 2005; Zhao et al., 2006).
Parallel parts could be easily mapped to a flow activity of BPEL with control links. Furthermore, several strategies were proposed to optimise the generated code and improve their readability, such as the methods in Zhao et al. (2007) and Ouyang et al. (2008). Compared with our method, those translations mainly focused on untangling unstructured cycles beyond considering OPs, say nothing of translating them.

Afterward, several methods began to mention OP during translation (Hauser et al., 2008; van der Aalst and Lassen, 2008; Ouyang et al., 2009; Hauser, 2010a, 2010b). Ouyang et al. (2006) illustrated the control link-based translation with an example containing an OP. van der Aalst and Lassen (2008) and Ouyang et al. (2008) illustrated the control link-based translation and implemented a tool called WorkflowNetBPEL4WS. Hauser et al. discussed OP in detail in a series of work (Hauser et al., 2008; Hauser, 2010a, 2010b). In their research, a workflow model was decomposed into a region tree, which could be translated gradually using a set of region-reduction rules. A rule for OP was presented particularly. Then the rule set was complemented in their following works and was applied in the transformation of unstructured process models into BPEL code. Those proposals mainly handled OPs in the following two ways: duplication of related nodes or utilisation of BPEL control links. The first one translated OP to structured forms by first duplicating the activities between the merge nodes and the join nodes, and then switching those two kinds of nodes (Hauser et al., 2008; Hauser, 2010a, 2010b). The other way treated OP as segments of a flow activity of BPEL (van der Aalst and Lassen, 2008; Ouyang et al., 2009). Compared with our method, the discussions on the applicable range of the two strategies are absent in those literatures. Moreover, none of them have mentioned intractable cases which could not be dealt with.

Different from the general methods, a transformation method based on a strictly formal model is proposed by Polyvyanyy et al. (2010) and Dumas et al. (2010). A structuring method was given under a truly concurrent equivalence notion for acyclic process model. Although OP was not definitely mentioned, a duplicate-and-push procedure was conducted on a demonstrated unstructured process model containing an OP. So in the context of transforming OP, their strategy would be classified to the duplication method. In their structuring method, the generated process models were required to be strictly block-oriented, not like BPEL. The difference from our method lies in the structural requirement of generated models.

It should be noted that a complete business process model usually consists of three significant parts: a control flow model to describe the execution logic, an information model for the data types used in the process model, and an organisational model with a role or authorisation model. In this paper, we focus on the control flow models. Meanwhile, there are other aspects of process models worth being focused, such as research in Sun and Ding (2011), Liu et al. (2013), and Zhang et al. (2013).

### 3 Semantically sound AD models

#### 3.1 Definition of AD models

To briefly define AD models, we select a subset of the AD meta-model, including the necessary elements for modelling control flows. In the following, a brief and informal definition of AD models is presented.

**Definition 1**: AD model an AD model is represented as a tuple \( AD = (\text{Nodes}, \text{Edges}) \), where:

1. \( \text{Nodes} = EN \cup IN \cup FiN \cup BN \cup CN \). \( EN \) is a set of ExecutableNodes.
2. \( IN \) is a set of InitialNodes and \( FiN \) is a set of FinalNodes.
3. \( BN = DN \cup MN \) denotes a set of branching nodes, where \( DN \) is a set of DecisionNodes and \( MN \) is a set of MergeNodes.
4. \( CN = FN \cup JN \) denotes a set of concurrent nodes, where \( FN \) is a set of ForkNodes and \( JN \) is a set of JoinNodes.
5. \( \text{Edges} \subseteq \text{Nodes} \times \text{Nodes} \) indicates a set of control flows.

An AD model consists of nodes and edges. The five node types ‘\( EN, IN, FiN, BN, CN \)’ are exclusive. First type is ExecutableNode, which represents the atomic action executed in process model. The other four types are all ControlNodes, which connect control flows. InitialNodes and FinalNodes denote the start and the end of a process model containing an OP. So in the context of transforming OP, their strategy would be classified to the duplication method. In their structuring method, the generated process models were required to be strictly block-oriented, not like BPEL. The difference from our method lies in the structural requirement of generated models.

Since Petri net is a kind of strictly formal model, by mapping AD models to Petri net, the definition of soundness for process models is more feasible and strict. Based on such consideration, we adopt Petri net as the semantic domain. The semantics of an AD model is defined as a tuple \( PN = (P, T, F, M) \) where \( P, T, F \) and \( M \) have their usual meanings as in Petri nets, and \( M \) is the initial marking. The semantics \( PN = (P, T, F, M) \) of an AD model \( AD = (\text{Nodes}, \text{Edges}) \) is derived by a mapping from \( AD \) to \( PN \), which is defined as follows.

1. \( P = IN \cup FiN \cup BN \cup \{p_e \mid e \in \text{Edges}_1 \} \)
2. \( T = EN \cup CN \cup \{t_e \mid e \in \text{Edges}_2 \} \)
3. \( F = \{(s_e, d_e) \mid e \in \text{Edges}_3 \} \cup \{(s_e, t_e), (x_e, d_e) \mid e \in \text{Edges}_3 \times \text{Edges}_2 \} \)
4. \( M = iN_1 + \cdots + iN_n \) \( (iN_j \in IN, j = 1, 2, \ldots, n) \)

Note: (1) For \( \forall e \in \text{Edges} \), \( e = (s_e, d_e) \), \( s_e \) is the node \( e \) leaves from and \( d_e \) is the node \( e \) points to; (2) \( \text{Nodes} \) are divided into two sets: \( IN \cup FiN \cup BN \) and \( EN \cup CN \). An edge from \( \text{Edges}_1 \) connects nodes from the first set, and this node is mapped to a place in \( PN \). An edge from \( \text{Edges}_2 \) attaches nodes from the second set, and this node become a
transition in \(PN\). Other edges all belong to \(Edges_3\). They all represent flow edges in \(PN\). Therefore \(Edges\) are divided into three disjoint sets of \(Edges_1\), \(Edges_2\) and \(Edges_3\).

\[
Edges_1 = \{ e | e = (s_x, d_x) \land \{s_x, d_x\} \subset IN \cup FiN \cup BN \},
\]

\[
Edges_2 = \{ e | e = (s_x, d_x) \land \{s_x, d_x\} \subset EN \cup CN \},
\]

\[
Edges_3 = \{ e | e = (s_x, d_x) \land \{s_x, d_x\} \cap (IN \cup FiN \cup BN) \neq \emptyset \land \{s_x, d_x\} \cap (EN \cup CN) \neq \emptyset \}.
\]

In \(PN = (P, T, F, M)\), places and transitions are referred by the names of their corresponding nodes or edges. For an edge \(e \in Edges_1 \cup Edges_2\), \(x\) means the place or transition mapped from \(e\). The definition of \(M_i\) states that in the initial marking, every place mapped from an initial node must hold one token, while the others hold none. The semantics implies that our control flow models capture the basic control flow patterns, in order to avoid the subtleties of full AD.

**Definition 2: structurally soundness.** An AD model (Nodes, Edges) is structurally sound if:

1. \(IN\) and \(FiN\) both contain only one node denoted as \(iN\) and \(fN\) respectively
2. for \(\forall n \in Nodes\), there is a path from \(iN\) to \(fN\) going through \(n\).

In the view of the semantic interpretation \(PN\), the definition requires that there are exactly one place \(i \in P\) and one place \(o \in P\) such that \(i = \emptyset\), \(o = \emptyset\) and for \(\forall x \in P \cup T\), \(x\) is on a path from \(i\) to \(o\). Actually, this makes the semantic interpretation \(PN\) satisfy the definition of WorkFlow nets (WF-net) (van der Aalst, 1997). By utilising static analysis about WF-nets on \(PN\), we can verify the structurally soundness of an AD model. In the following, we assume that all AD models are structurally sound. Consequently a semantic interpretation \(PN = (P, T, F, M)\) is a WF-net and \(M_i\) is redefined as \(M_i = iN\).

**3.2 Semantically sound AD model**

Transforming models without well-defined behavioural semantics does not make any sense in real world. By well-defined behavioural semantics we mean that every execution of a process model can complete correctly. It has a formal definition called semantically sound. In this paper, we follow the definition of soundness about WF-nets in (van der Aalst, 1997).

**Definition 3: Semantically soundness.** A structurally sound AD model is semantically sound if its semantic interpretation \(PN = (P, T, F, M)\) satisfies:

1. \(\forall_M (M_i \rightarrow M) \Rightarrow (M \rightarrow fN)\)
2. \(\forall_M (M_i \rightarrow M \land M \preceq fN) \Rightarrow (M = fN)\)
3. \(\forall_{i \in T} \exists_{M, M'} M_i \rightarrow M \rightarrow M'.\)

The notion \(\rightarrow\) means firing several transitions successively from a marking to another marking in Petri net. The three requirements above restrict \(PN\) to a sound WF-net. There are two typical structural flaws in AD models that break these rules: deadlocks and multiple active instances of the same activity. AD models that contain these structural flaws may induce in problems in execution.

**3.3 Some results on semantically sound AD models**

**3.3.1 Decomposition of AD models and SESE regions**

The decomposition of a graph-oriented process model into a hierarchy of regions is a crucial step to analyse it and translate it to block-oriented forms. Individual region could be transformed separately without affecting other parts. Generally, a single entry single exit (SESE) decomposition of a process model produces a set of SESE regions. Just as its name implies, an SESE region is a set of nodes and edges such that there exists exactly one edge entering the region, and one edge leaving it. Its definition is presented as follows.

**Definition 4: SESE region.** Let \(N, E\) be an AD model, \(R = (N', E')\) wherein \(N' \subset N\) and \(E' = E \cap (N' \times N')\) is a SESE region, if there exist two edges \(e, e' \in E\) such that \(E \cap ((N' \times N') \times N') = \{e\}\) and \(E \cap (N' \times (N' \setminus N')) = \{e'\}\). \(e\) and \(e'\) are called the entry and the exit edge of \(R\), respectively.

Figure 1 shows the decomposition of a semantically sound AD model. Rounded rectangle represents executable node, and other control nodes are compatible with the graphic notions of UML AD definition. In this figure, each SESE region is surrounded by a dotted rectangle. Those trivial SESE regions containing only one executable node are not displayed here. For an SESE region, we represent the SESE regions immediately enclosed in it as its child regions. For example, in Figure 1, \(R_1\) and \(R_2\) are the child regions of \(R_0\).

Two SESE regions are in sequence if the exit edge of one region is the entry edge of the other. A non-maximal sequence will lead to multiple decompositions of one process model. To avoid this situation, every SESE region should be canonical. It is required that if an SESE region consists of child regions all in sequence, it must contain child regions as many as possible. In other words, non-maximal sequences are disregarded.
Definition 5: Canonical region. An SESE region \( R = (N, E) \) is canonical if there is an edge \( e \in E \) such that \( e \) acts as the entry edge (exit edge) of another SESE region \( R' \), then the exit edge (entry edge) of \( R' \) called \( e' \) must satisfy \( e' \in E \) or \( e' \) is the exit edge (entry edge) of \( R \).

There are only two possible relations between two canonical regions in one AD model: disjoint or nested. Based on this, the unique decomposition of an AD model could be represented as a parse tree, called process structure tree (PST) (Vanhatalo et al., 2007). The PST imposes a hierarchical structure on a process model and inherits the properties of uniqueness and modularity from the programme structure tree. An AD model can be analysed and transformed by its PST.

3.3.2 Region growing rules of semantically sound AD models

Hauser et al. (2008) analysed the structural correctness of process models through region trees. Instead of SESE regions, a workflow is partitioned into more general regions, that is, regions with possible more than one entry and exit edges. The behaviour of the interfaces of regions is described by defining the input/output logic of a region, which consists of XOR and AND logic. A set of region-growing rules are proposed, which fall into three categories: one set of rules handle the sequential regions; a second set of rules resolve the parallel regions; the last set of rules are supposed to process all regions with mixed XOR and AND nodes.

Figure 2 shows the region-growing rule for OP defined in Hauser et al. (2008). An OP is a kind of structure, where two groups of regions are connected as shown in the left side of Figure 2 (we follow their graphical notations for regions). The group of \( n \) regions \( S_i \) \( (n \geq 2) \) is called the source regions of the OP and the group of \( m \) regions \( T_j \) \( (m \geq 2) \) is called the target regions of the OP. The connection manner satisfies that:

1. every \( S_i \) has no other outgoing edges except \( m \) paths with each one pointing to \( T_j \)
2. every \( T_j \) has no other incoming edges except \( n \) paths with each one starting from \( S_i \) (Hauser et al., 2008).

Applying the rule for OP to the two groups of regions will build a new bigger region with OP.

**Figure 2** The region-growing rule for OP

Separability is defined for those workflows whose sequential parts can be separated from the parallel parts (Hauser et al., 2008). Based on this, given a semantically sound AD model and its SESE decomposition, every SESE region falls into one of the following three cases:

1. **Sequential region**: a region that has only branching nodes as its child control nodes or no child control nodes at all, for example, \( R_0 \), \( R_1 \) and \( R_2 \) in Figure 1.
2. **Parallel region**: a region that has only concurrent nodes as its child control nodes. In addition, none of those edges that connect its child nodes can construct a cycle, for example, \( R_3 \) and \( R_4 \) in Figure 1.
3. **Region with OP** : a region that has both branching and concurrent nodes as its child control nodes. Furthermore, the only non-separable pattern it contains is OP, for example, \( R_5 \) in Figure 1.

Notice that for a SESE region \( R \) under question, we only focus on how the child regions of \( R \) are organised, and the internal structures of all child regions are out of account. Particularly, the child control nodes are those control nodes immediately enclosed in \( R \). In the view of the corresponding PST, we only focus on the child nodes of \( R \) and how they are organised in the model. By classifying the SESE regions of an AD model, structures with OPs can be separated from the sequential and parallel parts, and be analysed independently.

4 OP-involved pattern-based transformation

This pattern is proposed for SESE regions with OPs involved. An important subclass of OP, first-order OP, is defined, and then corresponding transformation strategies are discussed.

4.1 First-order OP

By classifying the SESE regions of an AD model, structures with OPs are separated from sequential and parallel ones. However, regions with OPs may be in various simple or nested forms. We define a subclass of OPs called first-order OPs, which has some favourable properties to facilitate the transformation. By defining first-order OPs, the complication of OPs is loosened at the price of losing the completeness.

Definition 6: First-order OP. Given an OP, if each source/target region contains no OPs (despite of the internal structure of any child SESE region of the source/target region), it is first-order.

We first present some properties about first-order OPs. These properties may refer to the set of region growing-rules in Hauser et al. (2008).

Property 1: Every source (target) region of a first-order OP has only one input (output) edge.

Proof: For a first-order OP, each of its source regions is a region with an XOR input and an AND output. Suppose \( S_i \) is such a region that has more than one input edges. If it is parsed by the set of region-growing rules, the last rule used must not be the one for OPs, or this OP is not first-order. Considering the input and output logic of \( S_i \), there are four
possible rules: \( C_{st}, P_{st}, C_s \) and \( P_t \) (refer to Hauser et al., 2008). If the last rule is \( C_{st} \) or \( C_s \), we concentrate on the successor region \( T \) contained in \( S_r \), or the predecessor region \( S \) is the focus. Then, a new region with an XOR input and an AND output is obtained. It is parsed again until it is divided into two basic regions containing only one control node. Because only a merge node followed by a fork node can form a bigger region with an XOR input and an AND output from two basic regions, \( S_r \) can be divided into two regions. One has an XOR input and a single output edge, and the other has a single input edge and an AND output. After updating the source region to be the latter one, the property of a source region can hold. The target regions have similar cases. Except in a target region, the former region from a division will act as a new target region.

Before the second property is presented, some definitions need to be introduced first.

**Definition 7:** ControlNode-like region. If a region has the similar interface to a ControlNode, we call it a ControlNode-like region, and assign it a type name depending on the concrete kind of ControlNode, that is, decision-, merge-, fork- or join-like region.

**Definition 8:** Matched decision (join)-like region of an OP. For a first-order OP \( op \), if there is a decision (join)-like region \( R \) of which the output (input) ControlFlows are connected to the input (output) ControlFlows of \( op \) one by one, \( R \) is the matched decision (join)-like region of \( op \).

**Property 2:** Given a SESE region with only first-order OPs, every OP has its matched decision-like or join-like region, at least one of them.

**Proof:** We replace each OP in the SESE region with a specific region composed of a MergeNode followed by a ForkNode. Because this kind of region has the same interface with an OP, the SESE region is still semantically sound. Due to the definition of first-order OP, each replacement does not influence each other. This procedure is illustrated in Figure 3. Since a SESE region without OPs is obtained, it should be separable. Apply SESE decomposition to the new SESE region. Then consider a region \( MF \) as the replacement of an OP. Since its MergeNode \( M \) and its ForkNode \( F \) will not be divided into one SESE region, there should be either a decision-like region \( D \) connected to \( M \), or a join-like region \( J \) followed with \( F \) to construct a new SESE region. And it is possible to have both of them. For the OP, \( D \) is its matched decision-like region and \( J \) is its matched join-like region. At least one of them exists.

**Figure 3** The matched decision-like/Join-like region of an OP

![Image](Image)

Obviously a first-order OP combined with its matched decision (join)-like region forms a fork (merge)-like region. Next, we identify a special class of first-order OPs, which are 'outermost' among all the first-order OPs in a SESE region in the perspective of matched decision-like or join-like regions.

**Definition 9:** Outermost OP. Given a first-order OP in a SESE region \( R \), if it is not contained by the matched join-like and decision-like regions of any other first-order OP in \( R \), we say the OP is outermost.

**Property 3:** In a SESE region \( R \) with OPs, if an outermost OP has both its matched decision-like and join-like region, it must be the only outermost OP in \( R \). Otherwise all the outermost OPs have only the same kind of matched regions.

**Proof:** If an outermost OP \( op \) has both the matched decision-like and join-like regions in \( R \), the three regions can form a SESE region, which is \( R \) definitely. So in \( R \) there are no other outermost OPs. Suppose that \( R \) contains more than one outermost OPs with different kinds of matched regions. For every outermost OP, replace the fork (merge)-like region that is composed of the OP and its matched decision (join)-like region with a ForkNode (MergeNode). All the substitutions do not interfere with each other due to the definition of outermost OPs. In addition, the semantically soundness of \( R \) remains and no child SESE regions are generated. Afterwards \( R \) is reduced to a SESE region without OPs but having both ForkNodes and MergeNodes, which contradicts the semantically soundness of \( R \). So the assumption does not hold, and all the outermost OPs have the same kind of matched decision regions.

4.2 First-order OP-involved pattern-based transformation

As mentioned in Section 1, the duplication method and the link-based approach have their respective applicable ranges. In the following, we will discuss them separately. Based on Property 1, translation is firstly conducted under the simpler situation where the source/target regions of an OP are basic regions (regions with one fork/merge node), then is extended to more general situations.

The duplication method is to copy the matched join-like region of an OP several times to disassemble it. So it is incapable of taming OPs without their matched join-like regions. Given a SESE region \( R \) with first-order OPs, the translation starts from all the outermost ones. Based on Property 3, the following three cases are considered in turn:

Case 1 There is only one outermost OP that has both the matched decision-like and join-like regions in \( R \). As Figure 4 illustrates, after duplication, each copy of the matched join-like region, together with a source region of the OP (denoted as a fork node), constitutes a new child SESE region of \( R \). Then these SESE regions newly generated can be handled independently. If there are no OPs in the
matched decision-like region, \( R \) is translated to a sequential region. Otherwise some OPs will be exposed as new outermost OPs in \( R \). If so, \( R \) should be analysed over again.

Case 2 All the outermost OPs have only their matched join-like regions in \( R \). The duplication method can breaks down all the outermost OPs in the same way as Figure 4 shows, while all the non-outermost ones are divided into SESE regions newly generated and can be handled independently. Since \( R \) is reduced to a separable SESE region with only MergeNodes, it must be a sequential region and can be translated.

Case 3 In a SESE region \( R \), all the outermost OPs have only their matched decision-like regions. The duplication method cannot handle such a region. For the link-based method, it is not necessary to consider outermost OPs specifically. All acyclic structures can be translated, and the translation is straightforward. Notice that the property of ‘transitionCondition/joinCondition’ of some source/target activities need to be deliberated carefully.

It is implied that there is only one stubborn case that neither of the two methods can handle: in a SESE region \( R \) with cycles and only first-order OPs, all the outermost OPs have only their matched decision-like regions. The incapable range can be further narrowed by the following property.

**Property 4:** If in a SESE region, all the first-order OPs have only basic regions as their source and target regions, and all the outermost OPs have only their matched decision-like regions, cycles may only appear in these decision-like regions.

**Proof:** Recall the proof of Property 3, the replacement of all the fork-like regions composed of an outermost OP and its matched decision-like region with ForkNodes makes \( R \) to be a region without OPs. Since it becomes separable and contains only ForkNodes, and no child SESE regions are generated, \( R \) must be a parallel region and contains no cycles now. Based on the precondition that the source and target regions of all the OPs are basic regions, cycles only may appear in these matched decision-like regions of the outermost OPs.

Now the stubborn case is that in a SESE region \( R \) only with first-order OPs, all the outermost OPs have only their matched decision-like regions, some of which contain cycles. If these matched decision-like regions with cycles have no any OPs, we can manage those cycles by the help of a technique called REL method, which is proposed by Zhao et al. (2006) to compile a process model with unstructured loops to BPEL code.

**Theorem 1:** If a decision-like region without OPs contains unstructured cycles, it can be translated into a decision-like region containing no cycles.

**Proof:** Let \( DR \) be such a region with \( n \) output ControlFlows denoted as \( e_i (1 \leq i \leq n) \) respectively. Apply the REL method to it. Firstly, \( DR \) is simplified to a FA. We label the target node of \( e_i \) with \( E_i \) \((1 \leq i \leq n)\) and denote it as a final node separately. It can be proved that each string the FA accepts must have an \( e_i \) only at the end. In other words, if every \( E_i \) \((1 \leq i \leq n)\) is not be substituted with the equation \( E_i = e \) and is regarded as a variable, the RE solution of the single start node can be organised to a polynomial of degree one in \( E_i \), without constants. Then the polynomial can be transformed to a basic region as follows: for each term, the coefficient of \( E_i \) is translated to BPEL code and its condition is extracted. Then a medi-node is generated with the BPEL code attached, and an input ControlFlow with the condition extracted and an output ControlFlow with the condition \( e_i \) represents, if any, are appended to the medi-node. Finally, a DecisionNode is added as the predecessor of the \( n \) medi-nodes. By doing this, a basic region having \( n \) output ControlFlows denoted as \( e_i \) \((1 \leq i \leq n)\) is obtained. Several strategies and optimisation rules can be applied to improve the efficiency of generated code. Please refer to Zhao et al. (2006) for more details.

Based on Theorem 1, the stubborn case can be handled by the link-base method after removing cycles. Confronted with the situation that a matched decision-like regions \( DR \) has both cycles and OPs, we extend the definition of outermost OPs to allow it to be defined in a non-SESE region and prove that every outermost OP in \( DR \) only has its matched join-like region. Then just like the operations in Case 2 of the duplication method, all those outermost OPs are tamed, while all the non-outermost ones can be handled independently. Afterwards, the cycles can be removed and the link-based approach can play its role again.

**Property 5:** Given an OP, if its matched decision-like region \( DR \) still contains OPs, all these outermost ones in \( DR \) has only their matched Join-like regions.

**Proof:** Replace the OP with a specific region that is composed of a merge node \( mn \) followed by a fork node. Then \( DR \) and \( mn \) form a new SESE region, which is denoted as \( R \). Then all the outermost OPs in \( DR \) become the outermost OPs in \( R \) and Property 3 holds. It is impossible that only one outermost OP with both matched decision-like and join-like region exists in \( R \) obviously, otherwise these structures can form a new SESE region. Consider the case that all outermost OPs have only their matched decision-like region, and replace all the fork-like regions they form with ForkNodes as Figure 5 shows. Afterwards, \( R \) should be reduced to a separable SESE region, which contradicts the fact that both a MergeNode \( mn \) and some ForkNodes exist.
Transformation from business process models to BPEL with overlapped patterns involved

in $R$. So all these outermost OPs in $DR$ only have their matched join-like regions.

**Figure 5** Illustration of the proof of Property 5

It is assumed that the source and target regions of a first-order OP are basic regions above. In the following, more general source/target regions are considered. Source regions are regarded as fork-like regions and target regions as merge-like regions now, and they are not supposed to contain any OP. It is easy to prove that such a source region does not contain any cycles. To tame cycles in target regions, a similar translation to that of Theorem 1 is conducted in Theorem 2. Afterwards, the link-based method becomes applicable.

**Theorem 2:** If a merge-like region without OPs contains unstructured cycles, it can be transformed to a merge-like region containing no cycles.

**Proof:** Suppose such a region with $n$ input ControlFlows denoted as $s_i \ (1 \leq i \leq n)$ respectively. Apply the REL method and simplify it to a FA. The source node of each $s_i$ is labelled with $S_i$ and is regarded as a start node separately. Then the solution of each start node can be solved respectively. Afterwards for each start node $S_i$, its solution is translated to BPEL code and a medi-node is generated with the code attached. Then an input ControlFlow with the condition $s_i$ represents, if any, is appended to it. Finally, a MergeNode is added as the successor of the $n$ medi-nodes. By doing this, a basic region having $n$ input ControlFlows denoted as $s_i \ (1 \leq i \leq n)$ is obtained. Similarly, several strategies and optimisation rules can be utilised.

For the duplication method, the generalisation of source regions will not influence its application. However, the situation about generalised target regions is complex to a certain extent. The transformation is illustrated in Figure 6. For each source region $SR_i \ (1 \leq i \leq n)$ of an OP, all the target regions and the matched join-like region need to be copied once. As the figure shows, in the copy of a target region $TR_i \ (1 \leq i \leq m)$ according to a source region $SR_i$, all its nodes unreachable from $SR_i$ are removed, together with all those dangling ControlFlows.

**Figure 6** Duplication method on an OP with generalised target regions

4.3 The algorithm for the OP-involved transformation

Based on the statements above, a recursion function $Tran(R)$ that translates a SESE region $R$ with first-order OPs into BPEL code is presented in Algorithm 1. In $Tran(R)$, the duplication method is utilised as frequent as possible. Another translation function having the link-based method with higher priority can also be defined.

**Algorithm 1** Translation of a SESE region with OPs into BPEL code

$Tran(R)$ begins with obtaining all the outermost OPs of $R$. Then the procedure is divided into three parts. Part 1 (lines 4 to 12): If all of them have their matched join-like regions, each one is disassembled by duplication. During every duplication procedure, new SESE regions are generated, where some OPs, if any, are exposed as outermost ones. These new regions are considered independently and thus passed to $Tran(R)$. Then revised $R$ with fewer OPs is passed to $Tran(R)$ again. Part 2 (lines 13 to 34): Otherwise, the link-based method is applied. For an outermost OP $ops[i]$'s matched decision-like region $DR$, all the outermost OPs in $DR$ are handled by duplication, and new generated SESE regions are passed to $Tran(R)$. Then the possible cycles in $DR$ and the target regions of $ops[i]$ are untangled using Theorems 1 and 2, respectively. Afterwards $R$ is translated using the link-based method. Part 3 (lines 2 to 3): Because the number of OPs in $R$ decreases in part 1, and new regions are formed in part 1 and part 2, a region without OPs may be passed to $Tran(R)$. 
5 Case study

In this section, a simple example is given to show how our method works. An AD model is shown in Figure 7, which describes the online payment process. This process contains only one SESE region with a first-order OP. This OP has only the matched decision-like region, denoted by DR. DR is surrounded by a dotted rectangle in Figure 7. According to the category of first-order OP involved structures, this model falls into the third case. Since DR is not acyclic, the cycle contained in it should be untangled by Theorem 1 before the link-based method is applied to the whole process.

Figure 8 shows the untangling procedure. In (a), DR is simplified as a FA with two supposed final nodes, and every edge is labeled with a unique id. After using the REL method, the solution of the start node ‘check balance’ is obtained, which is \( (\alpha[d]bc)^*(\delta e_1 E_1 + e_2 E_2) \). In (b), the decision-like region corresponding to the solution is shown. A medi-node is generated for \( (\alpha[d]bc)^* \), followed by a basic decision-like region generated for \( (\delta e_1 E_1 + e_2 E_2) \). The output ControlFlows of the new region in (b) are connected to their original successors according to the labels on the supposed final nodes. The RE \( (\alpha[d]bc)^* \) is compiled to BPEL, which is shown as the following pseudo code.

```
invoke "check balance"
thisExit = FALSE
while (\{insufficient balance\} AND NOT thisExit)
    \{ invoke "prompt"
        if [choose C.O.D] thisExit = TRUE
        else
            if [continue] invoke "recharge account"
        invoke "check balance"
    \}
```

Figure 7 A payment process model

Figure 8 Taming the unstructured cycle in the ‘decision-like’ region
After the translation above, a new SESE region composed of an ExecutableNode ‘pay online’ and the medi-node in Figure 7 forms after a SESE decomposition. Then the region is transformed to a sequence activity. Afterwards, the link-based method is applied to the whole AD. The generated code is shown as follows. Note that a sketch while activity serves as the place holder for the while structure in the pseudo code above.

```xml
<process name="OnlineProcess">  
   <flow>
      <link name="seq2COD"/>
      <link name="seq2get"/>
      <link name="seq2ded"/>
      <link name="COD2gua"/>
      <link name="COD2con"/>
      <link name="gua2rec"/>
      <link name="get2rec"/>
      <link name="ded2con"/>
      <link name="con2sen"/>
      <link name="ver2sen"/>
     <invoke name="verification">
        <source linkName="ver2sen"/>
     </invoke>
     <sequence name="payment">
        <source linkName="seq2COD">
           <transitionCondition={[ choose C.O.D]} >
        </source>
        <source linkName="seq2get">
           <transitionCondition={[ sufficient balance ]}> 
        </source>
        <source linkName="seq2ded">
           <transitionCondition={[ sufficient balance ]}> 
        </source>
     </sequence>
     <invoke name="payOnline"/>
     <invoke name="checkBalance"/>
     <while>
        ...
     </while>
     <invoke name="C.O.D">
        <target linkName="seq2COD"/>
        <source linkName="COD2gua"/>
        <source linkName="COD2con"/>
     </invoke>
     <invoke name="guaranteePoints">
        <source linkName="COD2gua"/>
        <source linkName="gua2rec"/>
     </invoke>
     <invoke name="getPoints">
     </invoke>
     <target linkName="seq2get"/>
     <source linkName="get2rec"/>
   </flow>
   <invoke name="deductAmount">
        <target linkName="seq2ded"/>
        <source linkName="ded2con"/>
   </invoke>
   <invoke name="recordPreference">
        <target linkName="gua2rec"/>
        <source linkName="get2rec"/>
   </invoke>
   <invoke name="confirmOrder">
        <target linkName="COD2con"/>
        <source linkName="con2sen"/>
   </invoke>
   <invoke name="sendReceipt">
        <targets>
           <joinCondition="bpws:getLinkStatus(ver2sen) and bpws:getLinkStatus(con2sen)"/>
        <target linkName="ver2sen"/>
        <target linkName="con2sen"/>
   </targets>
   </invoke>
   </flow>
</process>

6 Conclusions and further work

In this paper, we present the technique of transforming graph-oriented process models into BPEL focusing on OP structures. A subclass of OP called first-order OPs is defined. In the context of first-order OP, two existing strategies of handling OPs are discussed, and their applicable ranges are summarised respectively. Then the stubborn case that they cannot handle is described, and a new approach is proposed for it. By systematically discussing the OP-involved transformation, this paper makes a step forwards to complement the derivation of BPEL codes from graph-oriented process models.

In the future, we will work on two aspects: first, concrete algorithms and prototypes will be implemented to analyse their performance; secondly, we are trying to explore the transformation of the cases related to non-first-order OPs.

Acknowledgements

This research is supported by the National Natural Science Foundation of China (Nos. 61303031, 61420106004, 61133001, Nos. 61173026, 61373045).
References


