http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (8): 1468-1479, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.1468.1479

A Control-flow Pattern based Horizontal Business Process Model Transformation Approach

[1]Zhaogang Han, [1]Zhang Li, [2]Huang Wenqing and [1]Gang Wang
[1]Software Engineering Institute, Beihang University, Beijing, China
[2]Department of Computer and Information Engineering, Heze University, Shan Dong, China

Abstract: Due to company mergers and business to business interoperability, there is a need for model transformations in the area of business process modeling to facilitate scenarios like model integration and model synchronization. General model transformation approaches do not consider the special properties of business process models and horizontal transformation scenarios. Since different business process modeling languages use different modeling elements and syntax constraints, a business process may have very different representations in different languages. So in many cases it is not easy to transform business process models correctly only using an element-2-element mapping. In order to solve this problem and improve the correction rate of horizontal business process model transformation, in this work we propose CP-BPMT(Control-flow Pattern based Business Process Model Transformation), a horizontal business process model transformation approach in which the operation granularity has been increased from model elements to model fragments illustrating certain control-flow patterns. CP-BPMT is feasible in practice and it can generate transformation results better than element-based approaches. In order to illustrate CP-BPMT approach, a UML-AD2YAWL case study is conducted.

Key words: Model transformation, business process modeling languages, control-flow patterns

INTRODUCTION

Since companies realized the importance of business process modeling, more and more business process models are emerging in different companies. At the same time, the importance of horizontal business process model transformation, which is used to bridge between languages on the same level of abstraction, is becoming an increasing concern. Horizontal business process model transformation can be used in these scenarios: (1) When companies merge with others, usually their business process needs to be intergrated. To illustrate the new business processes, intergrated business process modeles are required. Since different companies usually use different process modeling languages, we need to first transform them into a unified business process modeling language and then intergrated them. (2) Since new business process manage systems, which are built based on different process modeling languages, may be used to instead the older ones, there arises a need to translate process models with older business process modeling languages into modern ones. (3) With the aid of horizontal business process model transformation, analysis tools based on certain business process modeling languages could be used to analyze business process models built by other languages.

Defining a horizontal business process model transformation, however, is still a difficult task as several domain-specific problems remain to be solved (Murzek and Kramler, 2007a). Business process modeling languages are used for a distinct purpose, namely the illustration of business processes in different companies. So, they all provide very similar concepts to the modeler. But since different business process modeling languages use different modeling elements and syntax constraints, a business process may have very different implementations in different languages and in some cases it is difficult to transform a business process model correctly only using an element-2-element mapping. Since model transformation languages like ATL, QVT or frameworks for general purpose model transformation mainly provide solutions for 1:1 and 1:n element correspondences, they are not adequate for horizontal business process model transformation. Based on the assumption that increasing the operation granularity during horizontal business process model transformation may help to provide a better transformation results, this study proposes CP-BPMT, a generic approach that provides a control-flow pattern based solution for different horizontal business process model transformation problems.

Corresponding Author: Zhaogang Han, Software Engineering Institute, Beihang University, Beijing, China Tel: +8682339230

RELATED WORK

In the horizontal business process model transformation domain, lots of ad-hoc approaches have been proposed. Ye *et al.* (2008b) and Decker *et al.* (2008) transformed BPMN into YAWL. In (Mendling *et al.*, 2006) EPC was transformed into YAWL. Han *et al.* (2010) transformed UML-AD into YAWL. All these transformations aim at the benefits from verification tools based on YAWL, such as Woflan, WofYAWL and ProM (Wynn *et al.*, 2009). Lopez-Grao *et al.* (2004) transformed UML-AD into petri-net to achieve a performance analysis via petri-net based tools. In order to bridge the gap between business models and workflow specifications, a UML-AD to BPMN transformation was discussed by Dehnert and van der Aalst (2004). All these approaches accomplish the transformation on a basis of element-mapping, pattern-based transformation has not been reported.

General model transformation languages, like ATL (Jouault and Kurtev, 2005) and QVT (OMG, 2008), have been proposed for generic transformation problems. These languages have their limits when dealing with specific problems in horizontal business process model transformation scenario, since they are mainly designed to support vertical model transformations in MDA (Murzek and Kramler, 2007a). In (He *et al.*, 2011), an improved QVT-R with a "pattern factor" was proposed, which can support nested patterns in horizontal business process model transformation. But how to handle the multiplicity of pattern actors remains unresolved.

Besides efforts trying to make general transformation languages suitable for this specific domain, numbers of generic approaches especially designed for this domain have also been proposed in the literature. A "Model Morphing Approach" was proposed, which tries to achieve a generic horizontal business process model transformation technology (Murzek and Kramler, 2007b). But only element mapping rules are used in this approach, pattern based transformation has not been reported. Furthermore, no experiment has been conducted to verify the feasibility of the proposed approach, which also makes it un-comparable with other approaches. Murzek *et al.* (2006) suggest using patterns in horizontal business process model transformation scenario. A set of "transformation patterns" was proposed and the usage of them was illustrated in the ADONIS® model to EPC transformation. The proposal only transforms control-flow aspect of business process model. "Transformation patterns" can only cover 10 workflow patterns that most process modeling languages share, which limits its application in practical usage. Also the proposal has not been verified via any practical application.

PROCESS STRUCTURE MODEL AND CONTROL-FLOW PATTERN

Before introducing the CP-BPMT approach, we first define the concept of process structure model. After introducing the concept of control-flow pattern which is related with process structure model, we introduce the control-flow pattern based decomposition of it.

Process structure model: A business process model can be understood from a number of different perspectives. The control-flow perspective describes tasks and their execution ordering through different constructors. The data perspective layers business and processing data on the control perspective, such as business documents and other objects which flow between tasks, local variables of the process, pre- and post-conditions of task execution. The resource perspective provides an organizational structure anchor to the business process in the form of human and device roles responsible for executing tasks.

The control-flow perspective of a business process model, which is determined by tasks and their interaction and interrelation, is the most essential part of a business process (Murzek *et al.*, 2006; Ye *et al.*, 2008a). Any business process model should contain this part of information, also many transformation technologies manly focus on this perspective (Mendling *et al.*, 2006). In thisstudy, the control-flow structure of a business process is called process structure, which mainly decides the execution order of tasks in a business process. A process structure model is the abstraction of process structure in a business process.

A process structure model can be derived from existing business process model by removing control-flow unrelated elements like resources, task inputs and outputs and etc. easily. The generation procedure of process structure model is shown in Fig. 1. In this study, we also focus on the transformation of process structure model like many other transformation approaches have done, resource and data information will be discarded.

Process structure model decomposition: A control-flow pattern is the abstraction of a typical control-flow semantic unit recurring in different business process models. Sequence, parallel split, synchronization and structured loop are good examples of simple but frequently used control-flow patterns. Control-flow pattern is independent with business process modeling language, they could be used to evaluate the control-flow perspective express power of different languages (Van der Aalst *et al.*, 2003). A Control-flow pattern

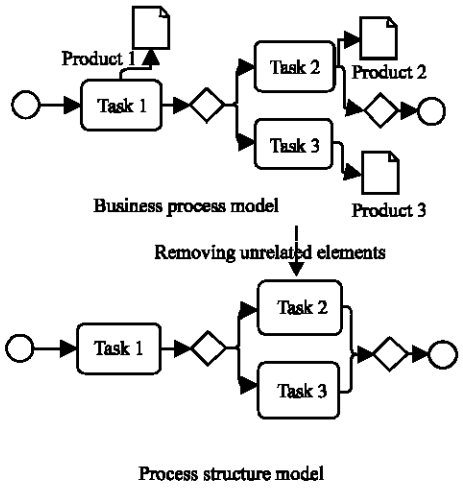


Fig. 1: Generation of process structure model

implementation is a possible expression form of a pattern in a certain language, thus it is specific with business process modeling language. A control-flow pattern may have different versions of implementation in different languages and even in a certain language. A control-flow pattern instance is a concrete model fragment in a business process model, which is instantiated from the corresponding control-flow pattern implementation and illustrating the semantic of its corresponding control-flow pattern. Pattern instance is specific with models.

Before discussing how to decompose a process structure model to a set of patterns instances instantiated from different patterns, we first define possible positional relationships between different pattern instances in a process structure model.

Definition 1 Positional relationship between control-flow pattern instances (cpi for short) in a process structure model can be defined as a tuple:

Relation = Disjoint, tangent, overlap, nest

where, relationship is defined on $cpi_m \times cpi_n$ and:

- $-cpi_m \times cpi_n$ = disjoint, iff $cpi_m \cap cpi_n = \Phi$
- $-cpi_m \times cpi_n$ = tangent, iff ' objects$\in$cpi $_m \cap$cpi $_n \rightarrow$ objects is kind of flow
- $-cpi_m \times cpi_n$ = overlap, iff ($\exists$object$\in cpi_m) \cap cpi_n \rightarrow$ objcet is kind of node)$\wedge(\neg(cpi_m \subset cpi_n)) \wedge (\neg cpi_n \subset cpi_m))$
- $-cpi_m \times cpi_n$ = nest, iff $cpi_n \subset cpi_m$ cpi_n is called bottom cpi and cpi_m is called top cpi in a nest relation

The possible positional relationships between control-flow pattern instances are illustrated in Fig. 2.

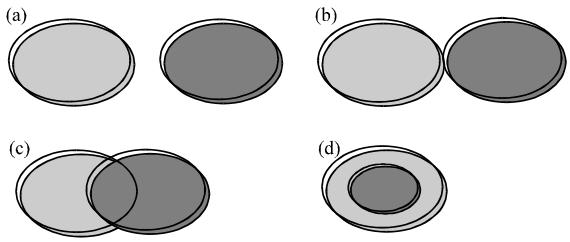


Fig. 2(a-d): Positional relationship of pattern instances, (a) Disjoint, (b) Tangent, (c) Overlap and (d) Nest

The process structure model decomposition procedure could be defined formally as below:

- Definition 2. A process structure model decomposition is defined as a dividing procedure from a process structure model (PSM for short) to a set of control-flow pattern instances (CPI for short) and a set of external constraints, which can be defined as:

PSM_Decomposition: = PSM$\rightarrow$<CPI, exterC>

where, CPI = (cpi_1,cpi_2,... cpi_n}, exterC = <exter $Constraint_1$, exter $Constraint_2$, ... exter $Constraint_m$>has defined the external constraints between different CPIs.

We supposed every task in a process structure model has at least one relationship with others, so a process structure model can be regarded as a strongly connected graph. If the description ability of predefined control-flow pattern set is adequate, i.e., the pattern set can cover all known control-flow semantics, a process structure model can always be decomposed into a set of control-flow pattern instances without any task left. This is because every task in a process structure model has at least one relationship with others, which is to say, it will at least participate in one control-flow pattern. Due to limited space, we have omitted detailed evidence that can back this claim. Process structure model decomposition is structure semantic consistent, if the resulted pattern instances could be reassembled into the original process structure model. To keep the consistence, external constraints between pattern instances should be well-defined.

Different types of external constraints will emerge in different positional relationship scenarios. Consider different positional relationships of control-flow pattern instances illustrated in Fig. 2. Obviously, there will be no external constraint between two pattern instances if they

have a disjoint positional relationship. A node-constraint appears if two pattern instances have an overlap relationship, which means they share the same nodes. A node-constraint can be defined as a tuple:

$$\text{nodeConstraint} = \langle cpi_1, cpi_2, node_1, node_2 \ldots node_n \rangle$$

where, cpi_1 and cpi_2 are two CPIs involved in this node Constraint, $node_1$, $node_2$,…$node_n$ is the overlapped nodes. A flow-constraint appears if two pattern instances have a tangent or nest relationship, which means they share the same flow (or we can say there is a flow f between them). A flow-constraint can be defined as a tuple:

$$\text{FlowConstraint} = \langle cpi_1, cpi_2, f, node_1, node_2 \rangle$$

where, cpi_1 and cpi_2 are two control-flow pattern instances involved in this flow constraint, f is the flow they share, $node_1$ and $node_2$ specify which nodes should adapt this flow in cpi_1 and cpi_2, respectively.

When, reassembling pattern instances into a whole process structure model, if two pattern instances have a positional relation of tangent or nest in the original model, we have to decide which nodes in cpi_1 and cpi_2 should be connected to the flow f they share, respectively. We call this problem a port matching problem, which has been shown in Fig. 3. In case that cpi_1 and cpi_2 need to be combined during a process structure model assembling, which nodes in cpi_1 and cpi_2 should be connected to the flow f should be determined. Since, there are two nodes in cpi_1 and three nodes in cpi_2, 6 conditions have been identified during the combination procedure. Which condition should be chosen can be determined by the flow constraints recorded in the process structure model decomposition procedure. If two pattern instances have a positional relationship of overlap in the original process structure model, their combination could be solved by the node constraints. So, we can conclude that with the aid of external constraints, we can reassemble these pattern instances into the original process structure model.

KEY STEPS IN CP-BPMT APPROACH

We use (and only use) pattern-based transformation rules in CP-BPMT approach, which will obviously increase the operation granularity. A transformation rule in CP-BPMT approach could be defined as below:

Definition 3: A transformation rule in CP-BPMT could be defined as a tuple:

$$TR = \langle SCPImp, TCPImp, r \rangle$$

where, SCPImp is a control-flow pattern implementation in source business process modeling language, TCPImp is a control-flow pattern implementation in target business process modeling language, r is a 1:1 mapping relationship between SCPImp and TCPImp.

The overview of CP-BPMT approach is illustrated in Fig. 4. After determining the source process modeling language and target process modeling language for a horizontal business process modeling language scenario, The CP-BPMT process emphasize two main phases, each of them contain some key steps, which briefly are described below with reference numbers.

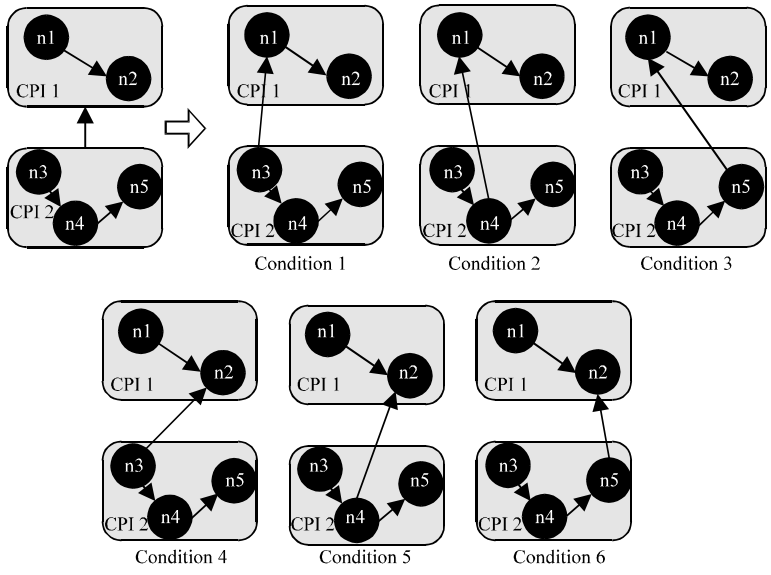


Fig. 3: Port matching problem in pattern instances composition

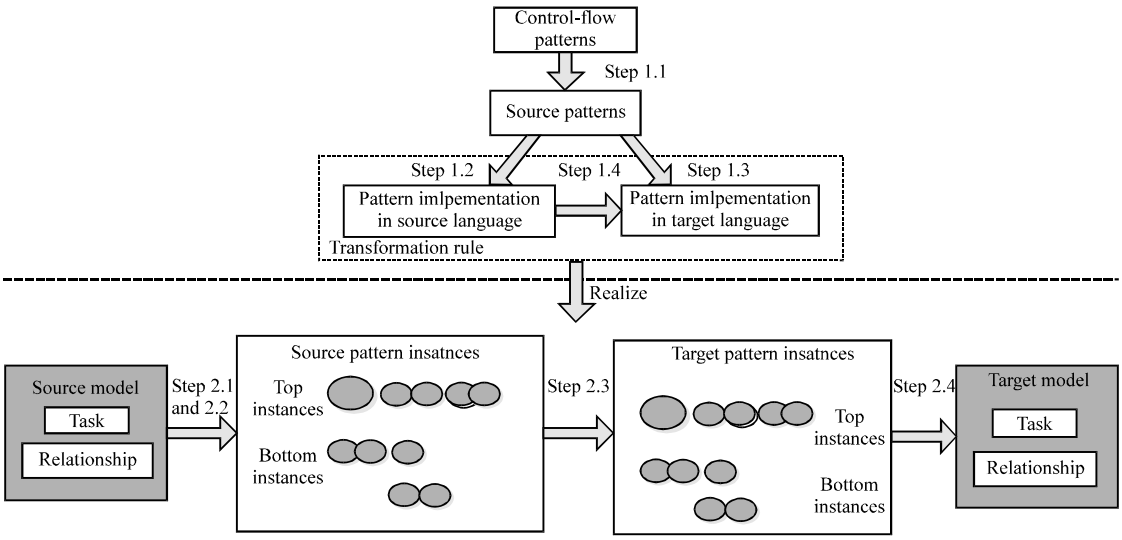


Fig. 4: Key steps in the CP-BPMT approach

Phase 1 transformation rule definition: In this phase transformation rules are defined by 4 steps, which will be introduced accordingly.

Step 1: Source pattern definition: In order to accomplish the transformation rule definition, a set of patterns that source process modeling language supports is required, which are called source patterns. A control-flow pattern set should be proposed as the basis of source patterns definition. There are many pieces of works concerning control-flow pattern definition in the literature and "workflow pattern" proposed by Van der Aalst *et al.* (2003) is an influential one among them (2363 citations according to Google scholar when thisstudy is written). Since its significant influence and widely usage in evaluating express power of different business process modeling languages (Wohed *et al*., 2005; Van Der Aalst, 2003; Mendling *et al*., 2005; White, 2004; Wohed *et al*., 2006), we suggest using them as the basis for source patterns definition. The end user of CP-BPMT may also define source patterns based on a pattern set different with "workflow pattern", if they can ensure every element in source process structure model can be divided into at least one control-flow pattern instances according to source patterns they defined.

Step 2: Source pattern implementation table definition: As transformation rules in CP-BPMT are mapping relationships between source pattern implementations and target pattern implementations, a table covering all possible pattern implementations of source patterns in source business process modeling language is required. Note that this table should be defined by the end user.

Step 3: Target pattern implementation table definition: There is no guarantee that all source patterns can be supported by the target business process modeling language (we claim a pattern is supported by a process modeling language, if this language can illustrate the control-flow semantic of this pattern with its notations). According to whether supported by target language, source patterns defined in step 1.1 can be further divided into "walk-around patterns" and "convertible patterns", walk-around patterns refers to source patterns which are not supported by target business process modeling language and convertible patterns refers to source patterns which are supported in target business process modeling language. Walk-around patterns and convertible patterns in a transformation scenario should be defined by the end user, which requires a good understanding of target process modeling language. Users also need to define all possible pattern implementation of every "convertible pattern" in target language.

Step 4: Transformation rule definition: Source pattern implementations instantiated from "walk-around patterns" could not be mapped to target ones directly, as there are no corresponding pattern implementations in the target language. This implies the weakness in control-flow perspective expression power of it. These source pattern implementations should be transformed by giving their walk-around expression forms in target language, by doing which we can reduce the information loss.

Specifically, they should be mapped to target pattern implementations with similar semantics and the semantic differences should be illustrated by natural language notes. Source pattern implementations instantiated from "convertible patterns" could be mapped to target ones directly, users need to define all transformation rules between source pattern implementations and target ones of every "convertible pattern".

Phase 2: Transforming source process structure model to target one. In this phase, source models are transformed into target ones according to transformation rules defined in phase 1 according to 4 steps, which will be introduced accordingly.

Step 1: Control-flow pattern instance detection: The definition of transformation rule is the basic of pattern instance detection, all pattern instances in source model instantiated from source pattern implementation in transformation rules should be identified automatically. There are several issues we should pay attention to. Firstly, sometimes pattern instance detection is not an easy job to accomplish: overlapped pattern instances may share certain model elements, which will increase the difficulty of identifying them; top instance and its bottom instances need to be identified if any nested instance exists in source model, which is not easy to accomplish. Secondly, as we know the execution priority of transformation rules may have an effect on the transformation result, i.e., different execution priorities of the same transformation rules may lead to different transformation results. In CP-BPMT this problem can be resolved in source pattern instance detection by following the principle of maximum matching. In source pattern instance detection, a model fragment may be divided into different (more than one) pattern instances, which will lead to different pattern instance detection results. An example for this is a model fragment illustrating the sequence pattern, it can be regarded as an instance of sequence pattern, or part of instance of other control-flow patterns (structured loop, for example). The maximum matching principle should be used in the procedure of pattern instance detection, which means we should always detect the maximum instance preferentially. Pattern instance detection is successful, only if every model element in a process structure model is designated into at least one pattern instance.

Step 2: Source process structure model decomposition: The source process structure model will be decomposed into pattern instances and transformed separately. The decomposition is based on the result of pattern instance detection. Overlapped nodes should be duplicated and dispatched to overlapped pattern instances, bottom pattern instances should be stripped off from corresponding top instances. Node-constraints between overlapped pattern instances and flow-constraints between tangent and nested pattern instances should be recorded for further use. The result of process structure model decomposition is control-flow pattern instances derived from source process structure model which are mutually isolated with each other and external constraints emerged in this procedure.

Step 3: Mapping source pattern instances respectively: Each source pattern instance will be mapped to its corresponding target one automatically, according to transformation rules defined in step 1.4. Target pattern instances are mutually isolated with each other. External constraints between target pattern instances could be deduced from external constraints between source pattern instances got in source process structure model decomposition.

Step 4: Target process structure model generation: Target pattern instances got in step 2.3 will be assembled to target process structure model in this step. The port matching problem could be resolved by external constraints between target pattern instances, which have been deduced in step 2.3. Note that although a complete target model hasn't be generated in our approach, it can be deduced easily from the target process structure model by simply adding target resource and data information which is corresponding to its source, since target process structure model is the most important part of target model.

UML-AD2YAWL AS A CASE STUDY

In the previous section, we described main phases of CP-BPMT and gave details of its processes. This section builds on that description by showing how to use CP-BPMT to transform business process models illustrated through a case study. This case study has examined UML-AD2YAWL transformation from a control-flow pattern's point of view (Although YAWL is an executable language, it can be regarded as a conceptual level language since it has a complete set of graphical notations illustrating business process concepts).

UML-AD2YAWL with CP-BPMT: In order to apply the CP-BPMT approach in UML-ADYAWL scenario, control-flow pattern based transformation rules need to be defined. Firstly, 20 workflow patterns are emploied as the start point for the source pattern definition and transformation rule definition, since its significant influence and widely usage in evaluating express power

Table 1: Workflow pattern support in UML-AD and YAWL

Pattern	AD	YAWL	Pattern	AD	YAWL
1 (seq)	+	+	11 (impl-t)	+	+
2 (par-spl)	+	+	12 (mi-no-s)	+	+
3 (synch)	+	+	13 (mi-dt)	+	+
4 (ex-ch)	+	+	14 (mi-rt)	+	+
5 (simple-m)	+	+	15 (mi-no)	-	+
6 (m-choice)	+	+	16 (def-c)	+	+
7 (sync-m)	-	+	17 (int-par)	-	+
8(multi-m)	+	+	18 (mile-st)	-	+
9 (disc)	+	+	19 (can-a)	+	+
10 (arb-c)	+	+	20 (can-c)	+	+

of different process modeling language (Note that there is no guarantee that 20 workflow patterns can cover all control-flow semantics). Among all 20 workflow patterns, 16 of them are supported by UML-AD (Wohed *et al*., 2005), which is illustrated as "+" in Table 1. These workflow patterns have composed the "source patterns" we need. Note that all source patterns are "convertible patterns", since YAWL supports all 20 workflow patterns (Van der Aalst and ter Hofstede, 2005) (as illustrated as "+" in Table 1).

Then UML-AD and YAWL pattern implementations for source patterns are defined. How UML-AD supports 20 workflow patterns have been discussed by Wohed *et al*. (2005). Since source UML-AD pattern implementations can be deduced easily, we do not detail it due to limited space. Interested readers can refer to (Wohed *et al*., 2005) for details. The YAWL pattern implementation definitions are built on a basis of (Van der Aalst *et al*., 2003; Van der Aalst and ter Hofstede, 2005; Russell *et al*., 2006). We have omitted these YAWL pattern implementations, interested readers can refer to these studies for details. Finally, a set of transformation rules is defined, which can be conducted easily from UML-AD and YAWL pattern implementations of source patterns. A partial transformation rules have been given in Table 2. Note that they can hardly be transformed correctly via element mapping. These transformation rules were detailed analyzed in (Han *et al*., 2012), interested readers can refer it for details.

Based on these pattern-based transformation rules, a UML-AD2YAWL tool has been implemented. Since control-flow pattern instance detection is a critical issue in CP-BPMT, we have designed a UML-AD pattern instance detection algorithm and implemented it in this tool. The architecture of this tool is shown in Fig. 5. This tool takes UML-AD models produced by the StarUML Modeler as input. In StarUML, UML-AD models are represented as *.uml files, which contain the XMI representation of them. These models are parsed by the open source toolkit JDOM in the "reading model file" module. The tool also provides a UI guiding user specify the location of source model, convert them and specify the location and name of target model. The YAWL engine file produced does not contain layout information, since it can be imported into YAWL editor which applies an automated layout algorithm.

Experiment and evaluation: In order to evaluate the CP-BPMT approach, an experiment has been conducted, which is based on a repository of 215 UML-AD models. These models were transformed by CP-BPMT and existing ad-hoc approach (Han *et al*., 2010), respectively and transform results were compared. The comparison verifies the value of the CP-BPMT approach. Vertices/edges information of these models is given in Fig. 6. Among these 215 models, there is a great variation in purpose of modeling, business domain and experience of the modelers. For this reason, we think that the models represent a reasonable good sample of real-world models.

The experiment environment for this case study is illustrated in Table 3.

The overall experiment results are shown in Table 4, according to 5 indicators which are key requirements for a model transformation: (i) Completeness, i.e., applicable to input model with arbitrary topology. Completeness can be evaluated by correctness rate, which is calculated as:

$$C = \frac{num_2}{num_1}$$

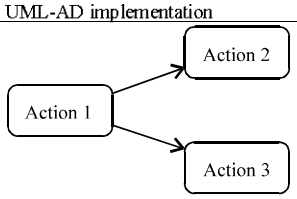
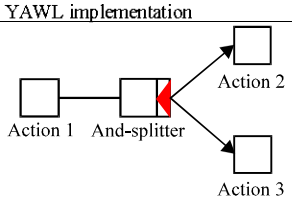
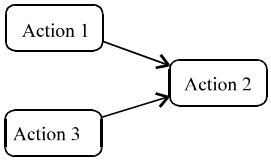
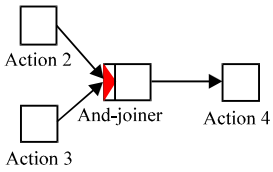
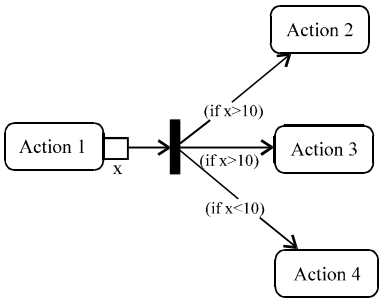
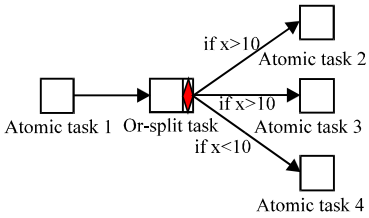
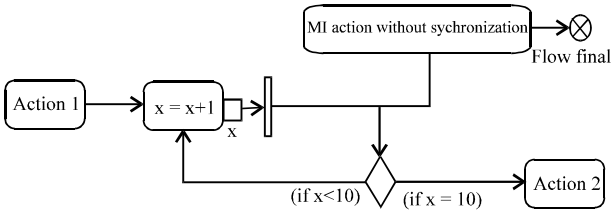
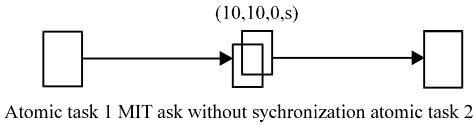
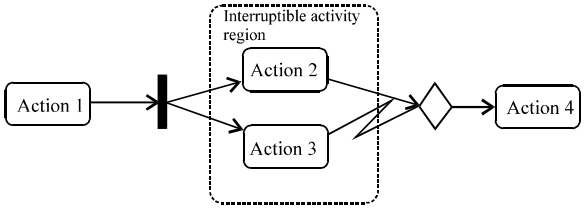
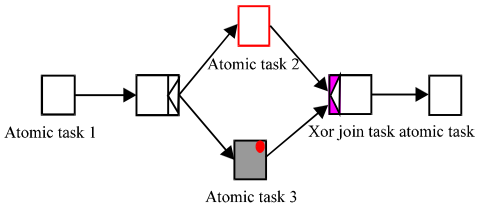
where, num_1 is the number of UML-AD models in this experiment and num_2 is the number of UML-AD models that a transformation tool can handle. (ii) Correctness, i.e., the semantic of produced target model is consistent with its corresponding source one. Correctness can be evaluated by correctness rate, which is calculated as:

$$C = \frac{num_2}{num_1}$$

where, num_1 is the number of YAWL target models a transformation tool generates and num_2 is the number of YAWL target models that are correct results. (iii) Efficiency, i.e., times required when transforming different input models. (iv) Automation, i.e., capable of producing target model without requiring human intervention. (v) Readability, i.e., target models are understandable by humans.

Completeness: From Table 4, we can see that approach proposed by Han *et al*. (2010) has covered 100% of all 215 models. This is easy to understand, since in element-mapping based transformation approaches, if

Table 2: Transformation rules in UML-AD2YAWL (Partial)

Pattern	UML-AD implementation	YAWL implementation
2(par-spl)		
3 (synch)		
6(m-choice)		
12(mi-no-s)		
20 (can-c)		

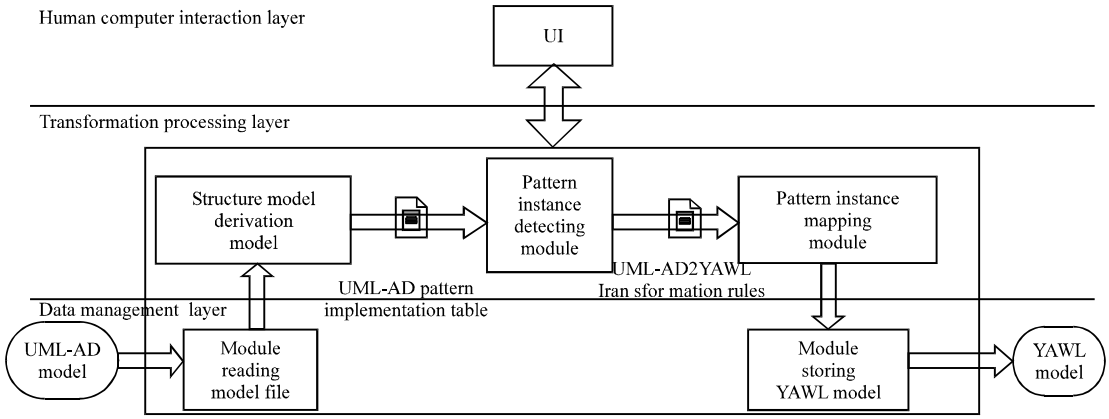


Fig. 5: System architecture of UML-AD2YAWL tool

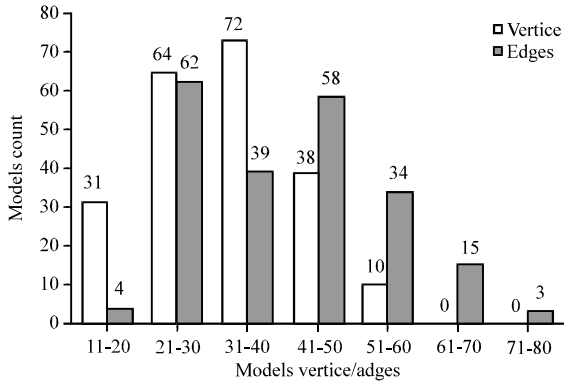


Fig. 6: Vertices/edges information of 215 models

Table 3: Experiment environment

Hardware configuration	Processor:Intel Core 2 Duo CPU 3.00 GHz RAM:4.00 GB
Operation system	Windows 7SP1(32 bit)
Runtime environment	By starting the Java virtual machine with the option-Xmx1536 m, we allowed a heap size of 1.5 GB to be used

Table 4: Comparison of two UML-AD2YAWL approaches

Approaches	Completeness (%)	Correctness (%)	Efficiency	Automation	Readability
Approach by Han *et al.*	100	79	Within 1 sec	Completely	No layout algorithm
CP-BPMT	100	92	Several sec	Completely	No layout algorithm

transformation rules have covered all meta-classes in source process modeling language, then all source models will be transformable. While CP-BPMT approach has also achieved a 100% completeness rate, which has implied that our control-flow pattern instance detection algorithm decomposed all source UML-AD models into separated pattern instances, which are instanced from 20 workflow patterns, no model element was left.

Correctness: If target YAWL result is consistent with its corresponding source UML-AD input, we claim it is a correct result. Judging the consistency between UML-AD input and YAWL result is not an easy job to accomplish, while judging the consistency between different YAWL results of different tools is much easier. In this experiment, we judge the correctness of a YAWL result like this: if transformation results of a source model by different tools are consistent with each other, we claim both of them to be correct results; if they are inconsistent, we identify their inconsistent points and analyze their correctness. By using this judging method, the correctness rates of these tools were identified as shown in Table 4. Since CP-BPMT approach has achieved a 92% correctness rate, while approach by Han *et al.* (2010) can only achieve a 79% correctness rate, we conclude that compared with element-mapping based approaches, CP-BPMT can better guarantee the correctness of horizontal business process model transformation.

Efficiency: To measure the efficiency of these tools, we transform all their convertible models and record the execution times of them. Both tools run as Java programs, approach by Han *et al.* (2010) could finish a single transformation within 1 second (even for the largest models) and CP-BPMT needs several seconds due to the pattern instance detection procedure. This is easy to understand, since the pattern instance detection in our approach would take some additional time compared to element-based approach by Han *et al.* (2010). But generally speaking there is no obvious efficiency difference between them for the given computing environment.

Automation: All these tools can finish the transformation with a complete automation. Note that it doesn't mean that the CP-BPMT approach is completely automated, on the contrary it requires a number of manual steps: source pattern definition, source and target pattern implementation table definition and transformation rule definition. Also the pattern instance detection algorithm has to be given manually. Whereas achieving an element-mapping based transformation also requires a manual definition of element mapping rules. Generally speaking, it is very hard to achieve a completely automated model transformation (Murzek and Kramler, 2007b).

Readability: None of these tools have achieved a layout algorithm, that is, they only produce a YAWL engine file which doesn't contain any layout information. This file can be imported into the YAWL editor which applies an automated layout algorithm. So the readabilities of them are exactly the same.

The overall experiment results are shown in Table 4. Based on the comparison results, we can draw a conclusion that our approach has done a good job in practice. Although there is a one-to-one element correspondence between UML-AD and YAWL in majority of 215 UML-AD models, a control-flow pattern based transformation is still necessary in a significant proportion of cases. This confirms the value of CP-BPMT. Our experiment shows the effectiveness of workflow pattern as a basis for source patterns definition, compared to the existing element-mapping based tools, the CP-BPMT tool based on it has achieved a satisfactory result.

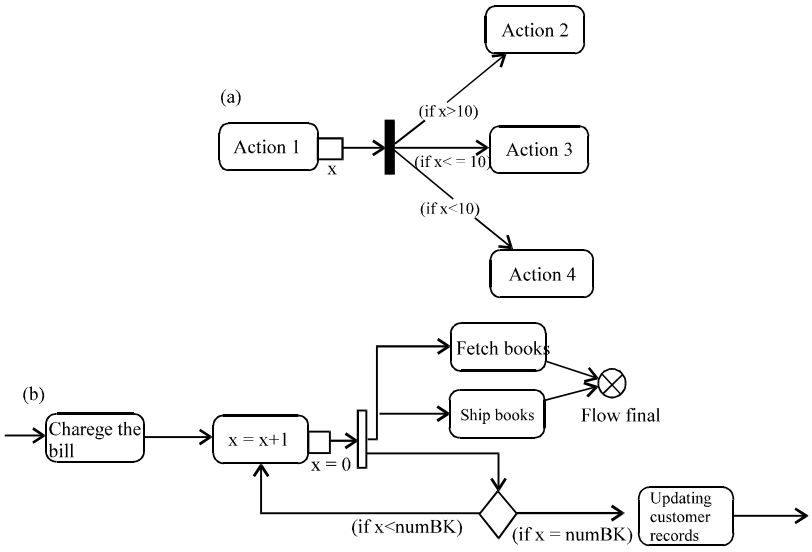


Fig. 7(a-b): Patterns that may be misjudged or missed

DISCUSSION

Firstly, although the feasibility of CP-BPMT and its advantage compared to element-mapping based approaches have been illustrated in this case study, still CP-BPMT fails to achieve a 100% correctness rate, which has revealed the inadequate of the pattern instance detection algorithm. This inadequate is caused by that the algorithm fails to handle some pattern implementations. Theoretically speaking, if transformation rules defined by end user are correct and pattern instance detection algorithm can perfectly detect all pattern instances, all UML-AD topologies should be handled correctly. But at this stage this is not the truth. Since some control-flow patterns may have extremely complex implementations in UML-AD which is hard to recognize, the pattern instance detection algorithm may misjudge and miss some pattern instances, both of which will lead to incorrect transformation results.

Consider UML-AD model fragments shown in Fig. 7, which are illustrating exclusive-choice pattern and MI without synchronization pattern, respectively. The pattern instance shown in Fig. 7(a) is illustrating exclusive choice, only when judgment conditions on successor control-flows following a ForkNode are totally exclusive. If these judgment conditions are partially exclusive then it is illustrating multi-choice. When these conditions are totally the same, or there is no judgment condition on these control-flows, this pattern instance turns out to be illustrating the parallel split pattern. In pattern instance detection algorithm employed in thisstudy, when there are judgment conditions on these control-flows we determine this instance to be instantiated from multi-choice pattern, which will lead to misjudgments when judgment conditions are totally the same or totally exclusive. For pattern instance shown in Fig. 7(b) which illustrates the MI without synchronization pattern (workflow pattern 12), since there are additional nested structures (the parallel structure of “Fetch Books” and “Ship Books”) in the loop body which contains the FlowFinal node, the detection algorithm will miss this pattern instance.

Secondly, we would like to discuss the versatility of the CP-BPMT approach. That is, how much additional efforts are required, if it is applied to other transformation scenarios. Since their effectiveness has been shown in this case study, 20 workflow patterns can still be employed as the basis of source pattern definition (which is not required, of course). Also the source pattern implementation table and target pattern implementation table should be defined, if involved languages in that scenario are different with UML-AD and YAWL. And the pattern instance detection algorithm for source language in that scenario has to be given by the end user of CP-BPMT.

CONCLUSION

Based on the discussion of process structure model and control-flow pattern, thisstudy proposes CP-BPMT, a control-flow pattern based generic approach that can be applied to different horizontal business process model transformation scenarios. CP-BPMT is

different with most existing approaches, in particular the transformation granularity is increased from model element to control-flow pattern instances. In order to illustrate CP-BPMT, a UML-AD2YAWL case study is elaborated, in which 20 workflow patterns are employed as the basis of source pattern definition. An experiment based on 215 real-world UML-AD models demonstrate the value and advantage of CP-BPMT compared to the existing element-mapping based approaches.

The main contribution of this work is threefold: Firstly, the concept of pattern instance based decomposition of process structure model has been defined in this study, which is theoretical basis for the CP-BPMT approach. Secondly, the CP-BPMT approach and its main steps are defined, which can be specialized for different horizontal business process model transformation scenario. Finally, we present a UML-AD2YAWL case study based on 215 real world UML-AD models, by which both the feasibility of CP-BPMT and its advantage compared to element-based approach, i.e., CP-BPMT can give result with a higher correctness rate, have been verified.

Control-flow pattern instance detection algorithms have to be given manually by the end user at this stage. In our future work we plan to develop a generic pattern instance detection algorithm, which is able to handle different process modeling languages automatically. This will obviously improve the automation degree of CP-BPMT approach.

ACKNOWLEDGMENT

This study is supported by the National Natural Science Foundation of China (Grant No. 61170087).

REFERENCES

Decker, G., R. Dijkman, M. Dumas and L. Garcia-Banuelos, 2008. Transforming BPMN diagrams into YAWL nets. Proceedings of the 6th International Conference on Business Process Management, September 2-4, 2008, Milan, Italy, pp: 386-389.

Dehnert, J. and W. van der Aalst, 2004. Bridging the gap between business models and workflow specifications. Int. J. Cooperative Inform. Syst., 13: 289-332.

Han, Z., L. Zhang and J. Ling, 2010. Transformation of UML Activity Diagram to YAWL. In: Enterprise Interoperability IV: Making the Internet of the Future for the Future of Enterprise, Popplewell, K., J. Harding, R. Poler and R. Chalmeta (Eds.), Springer, London, UK., ISBN-13: 9781849962568, pp: 289-299.

Han, Z., L. Zhang, J. Ling and S. Huang, 2012. Control-flow pattern based transformation from UML activity diagram to YAWL. Proceedings of the 4th International IFIP Working Conference on Enterprise Interoperability, September 6-7, 2012, Harbin, China, pp: 129-145.

He, X., Z. Ma, Y. Zhang and W.Z. Shao, 2011. Extending QVT relations for business process model transformation. J. Software, 22: 195-210.

Jouault, F. and I. Kurtev, 2005. Transforming models with ATL. Proceedings of the 5th International Conference on Model-Driven Engineering Languages and Systems, MoDELS 2005. Montego Bay, Jamaica, October 2-7, 2005, Springer, Berlin Heidelberg, pp: 128-138.

Lopez-Grao, J.P., J. Merseguer and J. Campos, 2004. From UML activity diagrams to Stochastic Petri nets: Application to software performance engineering. ACM SIGSOFT Software Eng. Notes, 29: 25-36.

Mendling, J., G. Neumann and M. Nuttgens, 2005. Towards workflow pattern support of event-driven process chains (EPC). Proceedings of the 2nd Workshop on XML4BPM, March 1, 2005, Karlsruhe, Germany, pp: 23-38.

Mendling, J., M. Moser and G. Neumann, 2006. Transformation of yEPC business process models to YAWL. Proceedings of the ACM Symposium on Applied Computing, April 23-27, 2006, Dijon, France, pp: 1262-1266.

Murzek, M., G. Kramler and E. Michlmayr, 2006. Structural patterns for the transformation of business process models. Proceedings of the 10th IEEE International Conference on Enterprise Distributed Object Computing Workshops, October 16-20, 2006, Hong Kong, China, pp: 18-28.

Murzek, M. and G. Kramler, 2007a. Business process model transformation issues. Proceedings of the 9th International Conference on Enterprise Information Systems, June 12-16, 2007, Madeira, Portugal, pp: 1-8.

Murzek, M. and G. Kramler, 2007b. The Model Morphing Approach-Horizontal Transformations Between Business Process Models. In: Proceedings of the 6th International Conference on Perspectives in Business Information Research, Nummenmaa, J. and E. Soderstrom (Eds.). University of Tampere, Tampere, Finland, ISBN-13: 9789514470967, pp: 88-103.

OMG, 2008. Meta object facility (MOF) 2.0 query/view/transformation specification. Object Management Group, OMG Document Number: formal/2008-04-03. http://www.omg.org/spec/QVT/1.0/PDF/.

Russell, N., A.H. Ter Hofstede and N. Mulyar, 2006. Workflow controlflow patterns: A revised view. BPM Center Report BPM-06-22. http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf

Van Der Aalst, W.M., 2003. Patterns and XPDL: A critical evaluation of the XML process definition language. Queensland University of Technology, Technology Report, FIT-TR-2003-06. http://www.ww.workflowpatterns.com/documentation/ documents/ce-xpdl.pdf

Van der Aalst, W.M.P. and A.H.M. ter Hofstede, 2005. YAWL: Yet another workflow language. Inf. Syst., 30: 245-275.

Van der Aalst, W.M.P., A.H.M. Hofstede, B. Kiepuszewski and A.P. Barros, 2003. Workflow patterns. Distrib. Parallel Databases, 14: 5-51.

White, S.A., 2004. Process Modeling Notations and Workflow Patterns. In: Workflow Handbook, 2004, Fischer, L. (Ed.). Future Strategies Inc., USA., ISBN-13: 9780970350961, pp: 265-294.

Wohed, P., W. Van der Aalst, M. Dumas, A.H.M. ter Hofstede and N. Russell, 2006. On the suitability of BPMN for business process modelling. Proceedings of the 4th International Conference on Business Process Management, September 5-7, 2006, Vienna, Austria, pp: 161-176.

Wohed, P., W.M. van der Aalst, M. Dumas, A.H. ter Hofstede and N. Russell, 2005. Pattern-based analysis of the control-flow perspective of UML activity diagrams. Proceedings of the 24th International Conference on Conceptual Modeling, Conceptual Modeling, October 24-28, 2005, Klagenfurt, Austria, pp: 63-78.

Wynn, M.T., H.M.W. Verbeek, W.M. Van Der Aalst, A.H. Ter Hofstede and D. Edmond, 2009. Business process verification-finally a reality!. Bus. Process Manage. J., 15: 74-92.

Ye, J., S. Sun, L. Wen and W. Song, 2008a. Transformation of BPMN to YAWL. Proceedings of the International Conference on Computer Science and Software Engineering, Volume 2, December 12-14, 2008, Wuhan, China, pp: 354-359.

Ye, J., S. Sun, W. Song and L. Wen, 2008b. Formal semantics of BPMN process models using YAWL. Proceedings of the 2nd International Symposium on Intelligent Information Technology Application, Volume 2, December 20-22, 2008, Shanghai, China, pp: 70-74.