Control-flow Pattern Based Transformation from UML Activity Diagram to YAWL

Zhaogang Han[1] , Li Zhang[1] and Jimin Ling[1], Shihong Huang[2]

[1] School of Computer Science & Engineering，Beihang University, China

[2]Department of Computer Science&Engineering，Florida Atlantic University,USA

{hanzhaogang@gmail.com，lily@buaa.edu.cn，lingjimin@126.com，shihong@fau.edu}

Abstract. Business process verification is an important topic in business process management (BPM). The verification of standard UML Activity Diagram is not easy due to lack of mature tools. YAWL (yet another workflow language) has a formal semantics based on Petri net; verification of YAWL model seems easier than other modeling languages such as UML-AD. A series of mature verification tools has been released (Woflan, WofYAWL, ProM) based on YAWL to find structural errors, such as deadlocks in the model. These tools can be used for verifying UML-AD models if they can be transformed to YAWL models. The most challenging problem is that some control-flow patterns in UML-AD can't be transformed via an element-to-element mapping. To solve this problem we provide a control-flow pattern based method for transforming a UML-AD model to YAWL. We regard these patterns that need to be transformed as whole model segments, pick them out from the UML-AD model and transform the left part using an element-to-element mapping as well as an object flow transforming method. We subsequently transform the picked-out patterns via patterns-based transformation and combine all the transformed YAWL segments to a new YAWL net.

Keywords: Business Process Verification, UML Activity Diagram, YAWL, Transformation，Control-flow Pattern.

1 Introduction

Business process verification is an important topic in business process management [1-5]. This involves identifying structural conflicts in the business process definition. UML Activity Diagram is intended for modeling computational and business processes [6-9]. UML Activity Diagram (UML-AD for short) has a wide usage in business process modeling and lots of research work have been done concerning its verification [10-12] , but these work usually take a subset of UML-AD as a verification basis and some advanced control structures (for example, cancel activity and cancel region) has been excluded from this subset.

YAWL [13-15] is a BPM system having a formal semantics based on Petri net. Because of its foundation on Petri net, its verification seems easier than other modeling languages such as UML-AD. There are lots of research works [16-19] and a series of mature tools [20-24] concerning the verification of YAWL. And YAWL models with advanced control structures like or-join and cancel region can be verified using these mature tools. In order to benefit from these mature tools, a transformation from UML-AD to YAWL is needed. In [25], the first step has been taken but only a small subset of UML-AD has been taken into account. So it is a good start but far from enough. In this paper we try to give a deep analysis of this transformation.

Some key questions in such a transformation are “Can we keep all the control aspect information without being lost in the transforming procedure” and “what technology should be taken to guarantee a “correct” transformation”. In this paper a method based on control-flow patterns is proposed to solve these problems.

We propose a three-step method to implement this transformation. Based on a rigorous analysis, numerous results are drawn on how UML-AD and YAWL support all the 43 control-flow patterns, control information of which patterns will lose during the transformation, which pattern need to be transformed as a model segment and which pattern can be transformed by element-to-element method. After picking out the model segments that need to be transformed by pattern-based way in the first step, we transform the rest parts of model based on an element-to-element mapping rule. In the second step we transform UML-AD object nodes and object flows into YAWL data and control flow since UML-AD object flows also contain control informations. In the third step we transform the picked-out model segments via patterns-based method and combine all the transformed YAWL segments to a whole YAWL net.

The rest of the paper is organized as follows. Section 2 provides background information regarding model transformation and UML activity diagram. In section 3 based on the assumption that the model segments that need to be transformed by pattern-based way have been picked out, we transform the rest parts of model based on UML-AD and YAWL meta-models. An element-to-element mapping is used first followed by transformation of UML-AD objects node and objects flow into YAWL data and control flow. Section 4 illustrates which patterns need to be transformed as a model segment by a pattern-based method and which patterns can be transformed by an element-to-element transformation based on a rigorous analysis. We then pick out the patterns (model segments) that need a pattern-based transformation and transform them via patterns-based method and combine all the transformed YAWL segments to a whole YAWL net. After presenting an example to illustrate our method (Section 5), we draw a conclusion and present future research issues in section 6.

2 Preliminaries

2.1 Model Transformation

Model transformation technology is first used in Model-Driven Software Development and now it has a widely application in many research domains [26]. Two kind of model transformation methods have been identified in [27]: Horizontal transformation where the source and target models reside at the same abstraction level and vertical transformation where the source and target models reside at different abstraction levels. The key point of model transforming is the definition of transformation rules. Traditional model transformation methods usually define their transformation rules based on the meta-models of the source model and the target model. Transformation rules are the mapping relations between the elements in the meta-models of the source and target model in majority of cases.

Model transformation is used to bridge the gaps between different business process languages on a different level of abstraction in BPM. Lots of research work has been done to transform other business process modeling languages to YAWL [25, 28-31]. In [28, 29] BPMN (business process modeling notation) has been transformed into YAWL. In [30] EPC (Event-driven Process Chains) has been transformed into YAWL. In [31] BPEL has been transformed into YAWL. These transformations all aiming at the benefits from the verification tools based on YAWL since these modeling languages support advanced control-flow patterns such as or-join and cancel region and YAWL verification tools can verify these patterns[32].

Pattern-based model transformation is not a novel technique. Some researchers have earlier proposed the pattern-based transformation. They suggested selecting some certain model segment in the source model and transforming them to model segment in the target model. But how to determine these model segments (patterns) remains an unsolved problem. Most of current existing researches use design patterns to conduct the pattern-based model transformation when transforming PIM model to PSM model in a MDA environment. Graphic pattern was also used to transform a source graphical model to a target graphical model. This kind of pattern-based method only enlarges the operating granularity during the transformation (from element to model segment), it has no advantage in enhancing the conversion accuracy and ensuring the transforming correctness compared to the element-to-element method. Using the control-flow pattern based method proposed in this paper, we can expect a more precise and more "correct" result than the element-to-element method.

2.2 UML Activity Diagram

A business process model usually includes four parts of information: control aspect, data aspect, resource aspect and exception handling aspect. Since Woflan, WofYAWL and ProM mainly verify the control structures of YAWL model and our purpose is to check structural errors in UML-AD model (deadlock and etc.), we only show an analysis on how to transform control-flow aspect information to YAWL, other issues such as resources and exception handling is beyond this paper's scope.
Figure 1 is a simplified UML-AD meta-model that contains necessary meta-classes that are needed to model business processes and we will exclude the resource (the ActivityPartation meta-class) and exception handling (the ExecptingHandler meta-class) aspects informations since they have no effects on the control structure correctness of business process model.

3 Meta-model Based Transformation

3.1 UML Meta-model

In order to define the element-to-element transformation rule, UML-AD meta-model and YAWL meta-model is needed. UML is a language defined under the MOF framework, UML-AD meta-model is described in [33] (chapter 11 and chapter 12).

The meta-model is, however, scattered in many small segments and some meta-classes solve no purpose for transformation. In this paper we illustrate in Figure 1the recommended subset of UML-AD notation for process definition. Meta-classes in light color rectangles stand for meta-classes having no graphic notations and used as classifier. Meta-classes in dark color rectangle stand for meta-classes having graphic notations which are used in modelling. The details of attributions can be found in [33].

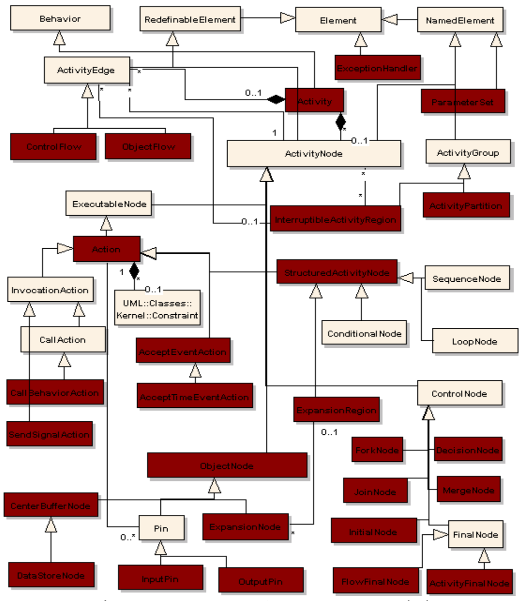


Figure 1. UML-AD meta-model

3.2 YAWL Meta-model

unlike UML-AD, YAWL is defined basing on a formal semantic of Petri net and it has no meta-model that fits the MOF framework. The YAWL graphic notations are shown in Figure 2.

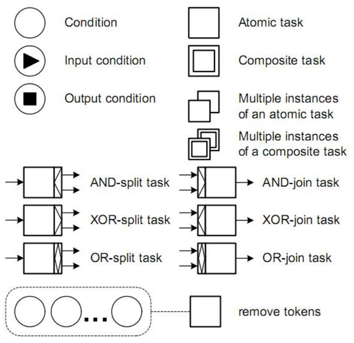


Figure 2. YAWL notations

After a careful study of paper [13, 34] and the YAWL 2.1 system, we come up with a YAWL meta-model that includes all the notations and meta-classes (without graphic notation) as shown in figure 3. Meta-classes in light color rectangles stand for meta-classes having no graphic notations which are used as classifier. Meta-classes in dark color rectangle stand for meta-classes having graphic notations which are used in modeling. Due to the limitation on space, the attributions have been left out.

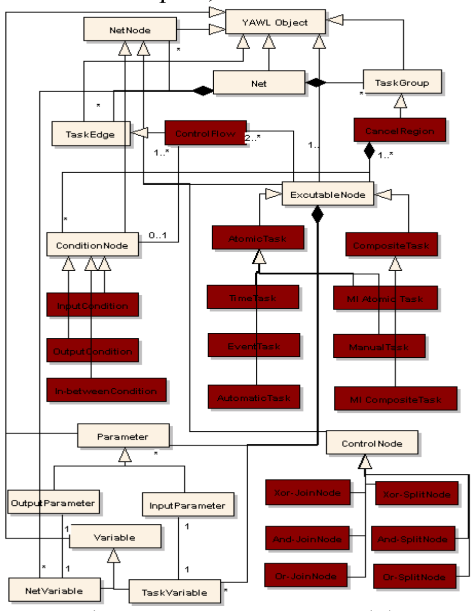


Figure 3. YAWL meta-model

3.3 Element-to-element Transformation Rule

Based on UML-AD meta-model and YAWL meta-model given in previous section, we can define an element-to-element mapping rule as shown in table 1.

Table 1. Element-to-element mapping rule

UML-AD Notation	YAWL Notation
StartNode&FinishNode	
InitialNode	InputCondition
ActivityFinalNode	OutputCondition
ExecutableNode	
Action	AtomicTask
SendSignalAction	AtomicTask
AcceptEventAction	EventTask
AcceptTimeEventAction	TimeTask
CallBehaviorAction	CompositeTask
ExpansionRegion	MultiInstanceCompositeTask
ControlNode	
ForkNode	And-SplitNode
JoinNode	And-JoinNode
DecisionNode	Xor-SplitNode
MergeNode	Xor-JoinNode

Edge(Flow)	
ControlFlow	ControlFlow
Containment	
Activity	Net

Not all notations have been listed in table 1. Note that there are no direct mapping relations between every element in UML-AD and YAWL. For example there are no corresponding YAWL notations for UML-AD object node and its subclasses.

3.4 Transform Object-Flows

Some business process modeling languages have more than one kind of flows connecting the basic modeling elements. For example BPMN has two types of flow namely sequence flow and message flow [35]. Some languages have only one kind of flow. The Event-driven Process Chains (EPCs) only has control flow connecting event, function and connector.

There are two kinds of activity edges in UML-AD, namely control flow and object flow [33]. While YAWL only has control flow connecting condition and task. YAWL has no graphic modelling notations for objects and object flow. How to transform object flow and different kinds of objects in UML-AD to YAWL is a big problem.

Objects and object flow are very important in UML-AD and there are several different kinds of object notations. These notations have been listed in figure 4. For more details about these notations refer to [33]. Not all object aspect informations will be transformed into YAWL since the majority of them have nothing to do with the control aspect information.

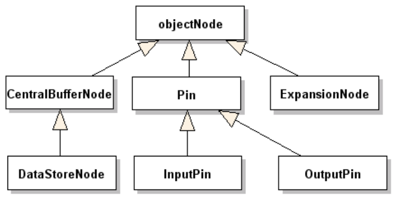


Figure 4. Object notations in UML-AD

As a business process modeling language, YAWL has provided modeling mechanism to describe objects and data, as well as the movement of objects and data transfer. In YAWL, data-aspects information is modeled as variables. There are two types of variables namely net variable and task variable, both of which can have a variable type pre-defined or defined by the user. Task can have input parameters and output parameters, an input parameter defines a data transformation from a net variable to a task variable and an output parameter defines a data transformation from a task variable to a net variable. For more information about data aspect of YAWL, refer to [36], chapter 5.

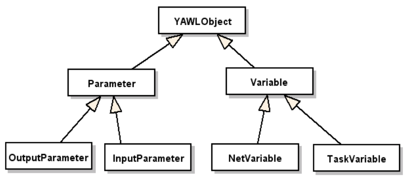


Figure 5. Data aspect meta-classes in YAWL

When transforming object flows in UML-AD to YAWL, we first transform all the object nodes to the "pin" form and then we transform pins attached to actions and object flow related with them to YAWL data attached to tasks and control flow between the tasks. Object flow attributes like weight and effect have been omitted during the transformation since they have nothing to do with the control-flow information contained in object flows.

As an example, consider a UML-AD segment with object flow shown in figure 6.

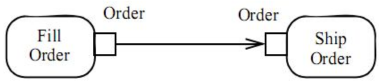


Figure 6. UML-AD object flow example

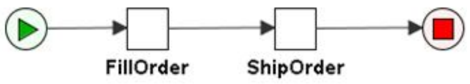


Figure 7. Corresponding YAWL notations of figure 6

We can model this UML-AD segment in YAWL editor. The graphic notations have been shown in figure 7. And in order to model the object "order" between tasks "FillOrder" and "ShipOrder", we need to define a data type named "OrderType" in the YAWL net that contains these two actions as shown in figure 8 and a net variable named "Order" with that data type. Then we define a task variable of task "FillOrder" and a task variable of task "ShipOrder" with the same name "Order", both of which have a data type of "OrderType". The transformation of "Order" then can be illustrated by two task parameters. One parameter is defined under task "FillOrder" with a parameter type "Out" to illustrate the flow of "Order" from the task "FillOrder" to the net and the other parameter is defined under task "ShipOrder" with a parameter type "In" to illustrate the flow of "Order" from the net to the task "ShipOrder".

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="OrderType">
    <xs:sequence>
      <xs:element name="OrderState" type="xs:enumeration{normal,final,pending}" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figure 8. Data type definition of OrderType

4 Control-flow Pattern based Transformation

4.1 An Introduction to Control-flow Pattern

Workflow pattern was first proposed by Van Aalst in 2003. In [37] Aalst systemically defined 20 workflow control-flow patterns. In 2006 Russell put forward a revised version of workflow control-flow patterns [38]. Shortly after the proposal of control-flow pattern, other workflow patterns like workflow data pattern [39, 40], workflow resource pattern [41, 42] and workflow exception pattern [43, 44] have also been identified. For more information about workflow patterns, refer to http://www.workflowpatterns.com/.

The research of workflow patterns have provided a thorough examination of various perspectives (control flow, data, resource, and exception handling) that need to be supported by a workflow language or a business process modeling language. The examined results can be used to evaluate the suitability of a particular process language or workflow system for a particular project, implement certain business requirements in a particular process-aware information system, and serve as a basis for language and tool development.

43 control-flow patterns are used to guarantee a "correct" transformation and enhance the precision of the transformation in this paper. Control-flow patterns are divided into eight categories, the first and the most simple category is the basic control-flow patterns, which includes sequence, Parallel Split, Synchronization, Exclusive Choice and Simple Merge. Figure 9 illustrates these five patterns in UML-AD. For detail description of these control-flow patterns, refer to [45].

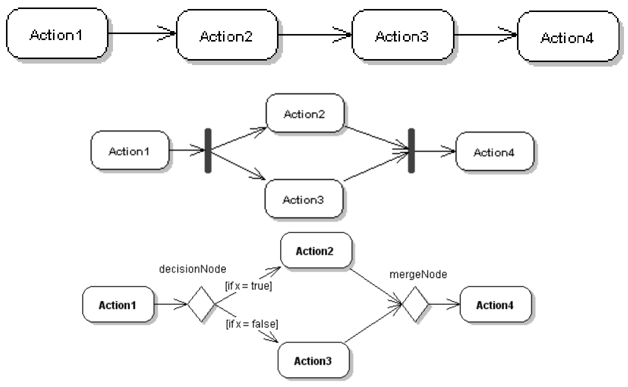


Figure 9. Basic control-flow patterns in UML-AD

4.2 Control-flow Patterns in UML-AD and YAWL

Neither YAWL nor UML-AD supports all the 43 control-flow patterns. For the patterns that are supported only by UML-AD, the control aspect informations that they stand for will be lost during the transformation, since they can't be described in YAWL. These patterns mainly belong to the categories of advanced branching and synchronization and trigger patterns; these patterns are colored as red in table 2. This implies the limits of YAWL when describing control aspect informations of business process model. An improved version of YAWL language has been proposed in [46], this new YAWL supports all the 43 control-flow patterns.

There are 10 patterns that are not supported by UML-AD though YAWL supports them. These patterns mainly belong to the category of advanced branching and synchronization and the category of state-based patterns. These are colored as light green in table 2. State-based patterns are not well supported since there are no notations representing state changing like a condition in YAWL.

Patterns that supported by both UML-AD and YAWL can be and should be transformed from UML-AD to YAWL without losing the control aspect information. But how to transform them properly remains a problem. After a thorough study of these patterns in UML-AD and YAWL, we identify 6 patterns that need to be transformed by a pattern-based way and 2 patterns that in some certain cases need to be transformed by a pattern-based way which are colored as orange in table 2. The patterns colored as green in table 2 represent those patterns that can be easily transformed using an element to element transformation. Detailed analysis will be given in the next section.

Besides these patterns, there are 3 patterns colored as blue in table 2 namely Pattern 22(Recursion), Pattern 27(Complete Multiple Instance Activity), and Pattern 36(Dynamic Partial Join for Multiple Instances) which are not supported by either UML-AD or YAWL.

Table 2. Control-flow pattern based evaluation of UML-AD and YAWL

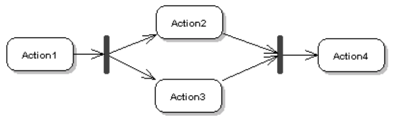
No	Pattern	AD	YAWL	No	Pattern	AD	YAWL
	Basic Control			34	Static Partial Join for MI	-	+
1	Sequence	+	+	35	Cancelling Partial Join for MI	-	+
2	Parallel Split	+	+	36	Dynamic Partial Join for MI	-	-
3	Synchronization	+	+		**State-based Patterns**		
4	Exclusive Choice	+	+	16	Deferred Choice	+	+
5	Simple Merge	+	+	17	Interleaved Parallel Routing	-	+
	Advanced Branching & Synchronization			18	Milestone	-	+
6	Multi-Choice	+	+	39	Critical Section	-	+
7	Structured Synchronizing Merge	-	+	40	Interleaved Routing	-	+
8	Multi-Merge	+	+		**Cancellation and Force Completion**		
9	Structured Discriminator	+/-	+	19	Cancel Task	+	+
28	Blocking Discriminator	+/-	-	20	Cancel Case	+	+
29	Cancelling Discriminator	+	+	25	Cancel region	+	+
30	Structured Partial Join	+/-	-	26	Cancel MI Activity	+	+
31	Blocking Partial Join	+/-	-	27	Complete MI Activity	-	-
32	Cancelling Partial Join	+	-		**Iteration Patterns**		
33	Generalised AND-Join	-	+	10	Arbitrary Cycles	+	+
37	Local Synchronizing Merge	+/-	+	21	Structured Loop	+	+
38	General Synchronizing Merge	-	+	22	Recursion	-	-
41	Thread Merge	+	-		**Termination Patterns**		
42	Thread Split	+	-	11	Implicit Termination	+	+/-
	MI Patterns			43	Explicit Termination	+	-
12	MI without Synchronization	+	+		**Trigger**		
13	MI with a Priori Design-Time Knowledge	+	+	23	Transient Trigger	+	-
14	MI with a Priori Run-Time Knowledge	+	+	24	Persistent Trigger	+	-
15	MI without a Priori Run-Time Knowledge	-	+				

To achieve a + rating (direct support) or a +/- rating (partial support) the language should satisfy the corresponding evaluation criterion of the pattern. Otherwise a - rating (no support) is assigned. We use color gray to identify the 8 pattern categories. Blue color stands for patterns that not supported by both UML-AD and YAWL. Light green is used for patterns that UML-AD doesn't support. Red color represents the patterns that YAWL doesn't support. Green color stands for patterns that can be transformed by element-to-element mapping. Orange color stands for patterns that need to be transformed by pattern-based transformation.

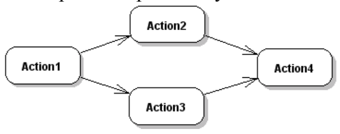
4.3 Patterns should be transformed by Pattern-based Transformation

There are five basic control patterns that are supported by all BPM languages. Due to their simplicity it seems unnecessary to transform them by the complex pattern-based way. But a careful study reveals that this is not the case. If parallel split (pattern 2) and synchronization (pattern 3) is modeled in UML-AD in the way shown in figure 10 (a), there will be no problem if we transform these two patterns by an element-to-element mapping: action in UML-AD to atomic task in YAWL, ForkNode in UML-AD to And-Split Node in YAWL and JoinNode in UML-AD to And-Join Node in YAWL. Then we will get a YAWL net as shown in figure 10 (c). The YAWL net we obtained is correct as it is in accordance with the YAWL syntax. We only need to first merge the Atomic Task 1 and AndJoin Task and then merge the AndSplit Task and Atomic Task 2 to finally get a result as shown in figure 10 (d).

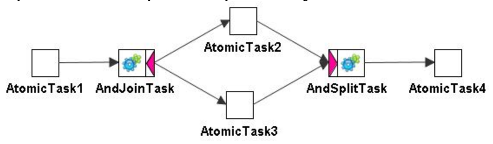
UML-AD language grammar allows another form of these two patterns. Consider a model segment shown in figure 10 (b). This is another form of parallel split (pattern 2) and synchronization (pattern 3) and they can't be transformed via an element-to-element mapping to YAWL model segment as figure 10 (c) since the ForkNode and JoinNode has been omitted and we can't get the "AndJoinTask" and "AndSplitTask" in figure 10 (c). And only after we have got figure 10 (c) can we simplify it to figure 10 (d). So we can draw a conclusion that parallel split and synchronization in figure 10 (b) can't be transformed to figure 10 (c) by element-to-element mapping and a pattern-based transformation is needed. We need to first recognize parallel split and synchronization in this form and then transform them as a whole to YAWL segment as shown in figure 10 (d).



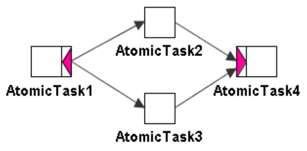
(a) Standard form of parallel split and synchronization in UML-AD



(b) Simplified form of parallel split and synchronization in UML-AD



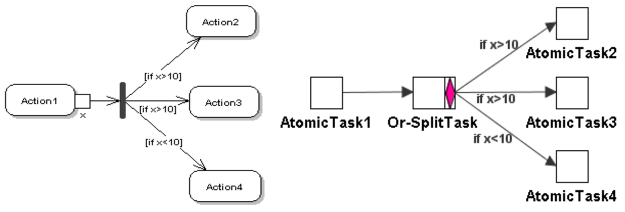
(c) Transformed YAWL



(d) Simplified YAWL

Figure 10. Different forms of parallel split and synchronization in UML-AD and YAWL

Another pattern that needs a pattern-based transformation is the multi-choice pattern (pattern 6). This pattern provides the ability for the thread of execution to be diverged into several concurrent threads on a selective basis. The decision as to whether to pass the thread of execution to a specific branch is made at runtime. This pattern is essentially an analogue of the Exclusive Choice pattern (pattern 4) in which multiple outgoing branches can be enabled [47]. Since there is no specialized notation for multi-choice in UML-AD, ForkNode is used to achieve this pattern as shown in figure 11(a). It is different from the parallel split pattern since there are guards on each of the output control flows. In the example of figure 11, after execution Action1 will output a ValuePin named "x" and the value of x has been determined during the execution of Action1. It is passed to the ForkNode and the routing is then determined by the value of x. If x>10 Action2 and Action3 will be executed and if x<10, Action4 will be executed. This is different from Exclusive Choice (Pattern 4) since in Exclusive Choice the guards on each of the outgoing control flows after ForkNode must be mutually exclusive. This pattern also needs a pattern-based transformation to get a result YAWL net as shown in figure 11(b) because if we use an element-to-element mapping the ForkNode notation in Multi-Choice pattern will be mapped to a YAWL And-JoinTask resulting in a wrong output.

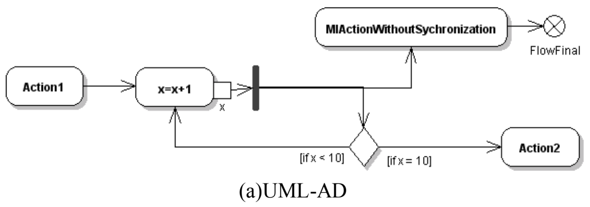


(a) UML-AD (b) YAWL

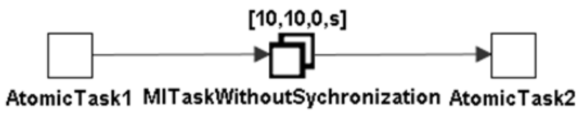
Figure 11. Multi-Choice (Pattern 6) in UML-AD and YAWL

Figure 12 is the multi instance without synchronization (pattern 12) in UML-AD and YAWL (refer to [6] and [36]). After Action1 (AtomicTask1), a multi-instance action namely "MIActionWithoutSychronization" (a MIAtomicTask named "MITaskWithoutSychronization") is created. The MI action (MITask) will create 10 instances of itself, leaving these instances executing and give the control-flow token to Action2(AtomicTask2) which means Action2(AtomicTask2) will start execution without the synchronization of any instances of the MI action (MI Task). Pattern 12 has a different form in UML-AD and YAWL due to the lack of specialized notation for MI actions in UML-AD.

As shown in figure 12(a), the initial value of x is 0, the action "MIActionWithoutSychronization" will be executed if x<10 so it will be executed 10 times and Action2 will not wait for the finish of its ten instances to start execution. The corresponding YAWL net is shown in figure 12(b). There is a specialized notation for MI actions in YAWL called MIAtomicTask having 4 parameters. The first parameter "10" means the task "MITaskWithoutSychronization" will have a minimum instance number of 10; the second parameter "10" means the task will have a maximum instance number of 10. So these two parameters define the instance number to be 10. The third parameter "0" means 0 instance of this MIAtomicTask needs to be synchronized and the last parameter "s" (short for static) indicates that new instances cannot be created dynamically during the execution of any instances. For more information of MIAtomicTask, refer to [36].



(a)UML-AD

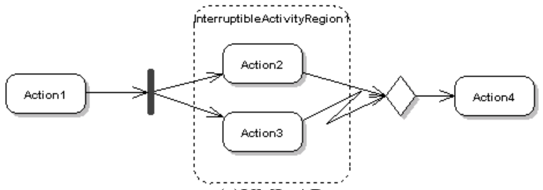


(b) YAWL

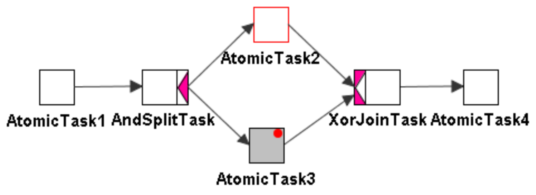
Figure 12. Multiple Instances without Synchronization (Pattern 12) in UML-AD and YAWL

InterruptibleActivityRegion can be used to model cancel-related patterns in UML-AD [6], whereas in YAWL CancelRegion is used to model these patterns. Element to element mapping transformation of these patterns will be troublesome. Due to the limited space we take cancel task (pattern 19) as an example to illustrate why a pattern-based transformation is needed and the other cancel patterns may be treated as same as cancel task pattern.

As shown in the UML-AD diagram of figure 13(a), the InterrruptibleActivityRegion has two actions in it: Action2 and Action3. Whereas in the corresponding YAWL net in figure 13(b), AtomicTask3 has a cancel region with only one task (AtomicTask2) in it (this is illustrated by the red point in the top right corner of the notation of AtomicTask3 and the red outline of AtomicTask2). So a pattern based transformation is also needed when transforming this pattern.



(a)UML-AD



(b) YAWL

Figure 13. Cancel Task (Pattern 19) in UML-AD and YAWL

After patterns that need to be transformed by pattern-based method have been identified, we can pick them out of the source UML-AD model, transform the left parts to YAWL using a meta-model based method and then transform the pick-out parts via a pattern based method. We subsequently combine all the transformed YAWL segments together and get a whole target YAWL model.

5 Example Illustrating Our Method

In order to illustrate our methods we take a UML-AD model shown in figure 14 as a source model and transform it to a target YAWL net as shown in figure 15.

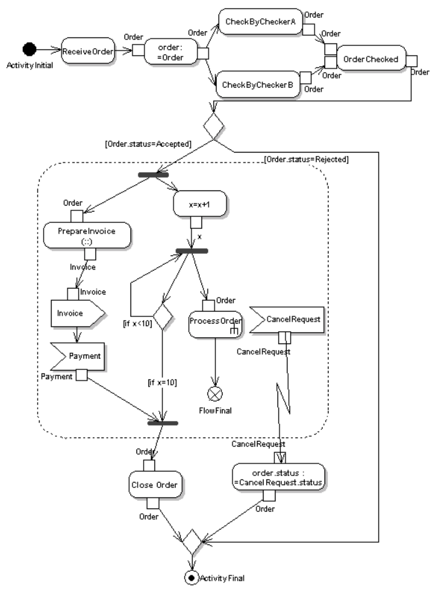


Figure 14. An order process model in UML-AD

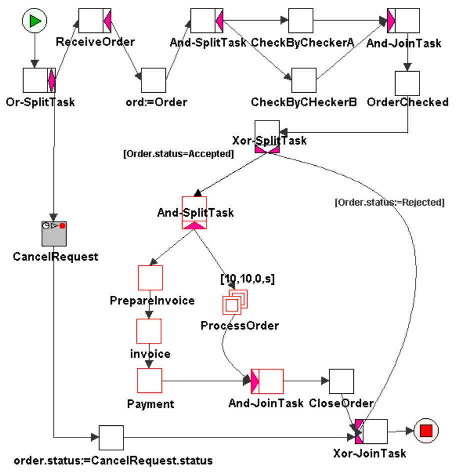


Figure 15. The target YAWL net

On top left corner of figure 14, pattern 2 and pattern 3 are used in a form shown in figure 10 (b) except that there are pins attached to actions and the flow type is object flow. Orders need to be checked by two order checkers namely checker A and B. The status of Order will be "Accepted" only if both of the checkers accept it, otherwise the status will be "Rejected" and the activity will be terminated. The "ProcessOrder" action will be executed for ten times by different workers so it is a multi-instance action. An InterruptableActivityRegion is used as cancel region (pattern 25), the whole order-processing process can be canceled if a "CancelRequest" is received.

Since there are three patterns that need a pattern based transformation in the source UML-AD model, we have to first pick these patterns out and transform the left parts of the model to YAWL via a meta-model based meethod, that is, an element-to-element mapping and a transformation of object flow. Note that we have omitted the data aspect of the target YAWL net due to a limited space. Then we transform the picked-out patterns using a pattern based transformation. Element-to-element mapping and object flow transformation may also be used during this procedure. The final target YAWL net is as shown in figure 15. Note that there is an additional Or-Split Task in the target YAWL net connecting the InputConditon and the TimeTask "CancelRequest". This signifies a problem caused by the multi-start phenomenon in UML-AD. For details about this problem, refer to [25].

6. Conclusion and Future Work

In order to benefit from the mature verification tools (Woflan, WofYAWL, ProM) based on YAWL, UML Activity Diagrams need to be transformed to YAWL nets. Since our goal is to verify UML-AD model and to check if it contains structural errors like deadlock and lack of synchronization, therefore our study has been limited to transformation of control-flow aspect information only. A transformation method that transforms all information that can be transformed to YAWL is beyond our discussion.

Based on a proposed YAWL meta-model, this paper has solved the UML-AD to YAWL transformation problem by a three step method. First UML-AD object nodes and object flows are transformed into YAWL control flows by adding necessary data to the corresponding task nodes. Secondly all control-flow patterns that UML-AD supports are analyzed and classified based on their transformation either by element-to-element mapping or by pattern to pattern methods. Finally the remaining model segments are transformed by element-to-element mapping based on UML-AD meta-model and the proposed YAWL meta-model. Since this method is more precise than the method proposed in [25], we can expect a better result when verifying UML-AD models using YAWL verification tools.

The future work includes applying this control-flow pattern based transformation method to transform other business process modeling languages to YAWL to check if we can get better transformation results than other transformation methods [28-31]. Moreover we aim to use this transformation method to convert the large amount of UML-AD process models to YAWL and analyze them with verification tools such as Woflan, WofYAWL and ProM.

Acknowledgments This work is supported by the National Natural Science Foundation of China under Grant No. 61170087.

References

1. Burlton, R., *Business process management: profiting from process*. First ed. 2001, Indianapolis, IN, USA: Sams.
2. ter Hofstede, A., et al., *Business Process Management: A Survey.* 2003. p. 1019-1019.
3. Hepp, M., et al., *Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management.* E-Business Engineering, IEEE International Conference on, 2005. p. 535-540.
4. Smith, H. and P. Fingar, *Business Process Management: The Third Wave*. 2006: Meghan-Kiffer Press.
5. Weske, M., *Business Process Management: Concepts, Languages, Architectures*. 2007, Secaucus, NJ, USA: Springer-Verlag New York, Inc.
6. Russell, N., et al. *On the suitability of UML 2.0 activity diagrams for business process modelling*. in *APCCM '06*. 2006. Darlinghurst, Australia, Australia: Australian Computer Society, Inc.
7. Dong, Y. and Z. ShenSheng, *Using π - calculus to Formalize UML Activity Diagram.* Engineering of Computer-Based Systems, IEEE International Conference on the, 2003. p. 47.
8. List, B. and B. Korherr, *A UML 2 Profile for Business Process Modelling.* 2005. p. 85-96.
9. Korherr, B. and B. List, *Extending the UML 2 Activity Diagram with Business Process Goals and Performance Measures and the Mapping to BPEL.* 2006. p. 7-

18.
10. Eshuis, R. and R. Wieringa. *Verification support for workflow design with UML activity graphs*. in *ICSE '02*. 2002. New York, NY, USA: ACM.
11. Baldan, P., A. Corradini and F. Gadducci, *Specifying and Verifying UML Activity Diagrams Via Graph Transformation.* 2005. p. 18-33.
12. Eshuis, R., *Symbolic model checking of UML activity diagrams.* ACM Trans. Softw. Eng. Methodol., 2006. p. 1--38.
13. van der Aalst, W.M.P. and A.H.M. ter Hofstede, *YAWL: yet another workflow language.* Inf. Syst., 2005. p. 245--275.
14. van der Aalst, W.M.P., et al., *Design and Implementation of the YAWL System.* 2004. p. 281-305.
15. Hofstede, T.A.A., et al., *Modern business process automation:YAWL and its support environment.* 2010: Springer.
16. Verbeek, H.M.W., W.M. van der Aalst and A.H. ter Hofstede, *Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Relaxed Soundness and Invariants.* 2007.
17. Verbeek, H.M.W., et al., *Verifying workflows with cancellation regions and OR-joins: an approach based on invariants*. 2006, Proceedings of BPM 2006, volume 4102 of LNCS.
18. Wynn, M., et al., *Verifying Workflows with Cancellation Regions and OR-Joins: An Approach Based on Reset Nets and Reachability Analysis*, in *Lecture Notes in Computer Science*, S. Dustdar, J. Fiadeiro and A. Sheth, Editors. 2006, Springer Berlin / Heidelberg. p. 389-394.
19. Wynn, M.T., et al., *Business Process Verification - Finally a Reality!.* Business Process Management Journal, 2010.
20. Verbeek, E. and W. van der Aalst, *Woflan 2.0 A Petri-Net-Based Workflow Diagnosis Tool.* Application and Theory of Petri Nets 2000, 2000: p. 475-484.
21. van der Aalst, W., et al., *ProM 4.0: Comprehensive Support for Real Process Analysis.* Petri Nets and Other Models of Concurrency – ICATPN 2007, 2007: p. 484-494.
22. Verbeek, H.M.W., T. Basten and W.M.P. van der Aalst, *Diagnosing Workflow Processes Using Woflan.* THE COMPUTER JOURNAL, 1999. p. 2001.
23. van Dongen, B.F., et al., *The ProM Framework: A New Era in Process Mining Tool Support.* 2005. p. 444-454.
24. Verbeek, E., *WofYAWL.* Technical report, 2005. *http://home.tm.tue.nl/hverbeek/wofyawl03.pdf.*
25. Han, Z., L. Zhang and J. Ling, *Transformation of UML Activity Diagram to YAWL.* 2010: p. 289-299.
26. SENDALL, et al., *Model transformation: The heart and soul of model-driven software development.* 2003. p. 4.
27. Mens, T. and P. Van Gorp, *A Taxonomy of Model Transformation.* Electronic Notes in Theoretical Computer Science, 2006. p. 125-142.
28. JianHong, Y., et al. *Transformation of BPMN to YAWL.* in *Computer Science and Software Engineering, 2008 International Conference on.* 2008.
29. Decker, G., et al., *Transforming BPMN Diagrams into YAWL Nets.* Business Process Management, 2008: p. 386-389.
30. Mendling, J., M. Moser and G. Neumann. *Transformation of yEPC business*

process models to YAWL. 2006. New York, NY, USA: ACM.

31. Brogi, A. and R. Popescu, *From BPEL Processes to YAWL Workflows.* Web Services and Formal Methods, 2006: p. 107-122.
32. Wynn, M.T., et al., *Business Process Verification Finally a Reality!.* Business Process Management Journal, 2010.
33. *The Unified Modeling Language™ (UML) specification version 2.3.* 2010, Object Management Group.
34. van der Aalst, W.M.P., et al., *Design and Implementation of the YAWL System.* 2004. p. 281-305.
35. *Business Process Modeling Notation (BPMN) Version 2.0.* 2009, Object Management Group/Business Process Management Initiative.
36. ter Hofstede., *YAWL User Manual Version 2.0*. 2009.
37. van der Aalst, W.M.P., et al., *Workflow Patterns.* Distributed and Parallel Databases, 2003. p. 5-51.
38. Russell, N., A.H.M. Ter Hofstede and N. Mulyar, *Workflow ControlFlow Patterns: A Revised View*. 2006.
39. Russell, N., A.H.M. Ter Hofstede and D. Edmond, *Workflow data patterns*. 2004.
40. Russell, N., et al., *Workflow Data Patterns: Identification, Representation and Tool Support.* 2005. p. 353-368.
41. Russell, N.N., et al., *Workflow resource patterns.* 2005, Technische Universiteit Eindhoven, BETA.
42. Russell, N., et al., *Workflow Resource Patterns: Identification, Representation and Tool Support.* 2005. p. 216-232.
43. Russell, N., A. Ter Hofstede and W. van der Aalst, *Workflow Exception Patterns.* 2006.
44. Russell, N. and A.H.M. Ter Hofstede, *Exception Handling Patterns in Process-Aware Information Systems*.
45. Russell, N., A.H.M.T. Hofstede and N. Mulyar, *Workflow ControlFlow Patterns: A Revised View*. 2006.
46. Russell, N.N., V.D.W.W. Aalst and T.A.A. Hofstede, *New YAWL: designing a workflow system using coloured Petri Nets*. 2008, Xidian University.
47. Russell, N., A.H.M.T. Hofstede and N. Mulyar, *Workflow ControlFlow Patterns: A Revised View*. 2006.