

Transformation of UML Activity Diagrams into Analyzable Systems and Software Blueprints Construction

CHIEN-YUAN LAI^a, DONG-HER SHIH^{a*}, HSIU-SEN CHIANG^b, CHING-CHIANG CHEN^a

Department of Information Management

^aNational Yunlin University of Science and Technology

^bDepartment of Information Management, Da-Yeh University

^a123, Section 3, University Road, Douliu, Yunlin, Taiwan, R.O.C.

^b168, University Road, Dacun, Changhua, Taiwan, R.O.C.

g9723803@yuntech.edu.tw, shihdh@yuntech.edu.tw, chianghs@mail.dyu.edu.tw, g9723802@yuntech.edu.tw.

Abstract: - Unified Modeling Language (UML) is a standard language for software blueprints, UML can be used to visualize, specify, construct and document software-intensive system of heritage. In the UML, the activity diagrams often are widely used to workflow and system flow in system analysis. However, the activity diagram of UML now there are still many drawbacks to be overcome, such as lacks support for simulation, dynamic semantics limits and verifiability capabilities. Petri nets are a popular technique for modeling the control flow dimension of workflows. Associative Petri nets (APNs) not only take all the advantages of PNs but also has a complete semantics, simulation and verifiability capabilities. Therefore, in this paper, we propose a methodology to describe how UML Activity diagrams can be intuitively translated into an APN model. This work can improve the simulation and verifiability capabilities of activity diagram and provides the systematic procedure to reduce complexity of translating activity diagrams into associative Petri net.

Key-Words: - Unified Modeling Language, Activity Diagram, Visualize, System Description

1 Introduction

Unified Modeling Language (UML) is the standard notation of Object Management Group (OMG) for object-oriented modeling. It is easy, graphical and appealed, but still too imprecise in several cases. UML is considerably easy to be used but it does not support formal model analysis because it does not have a formal semantics. Although UML is strong as modeling means, supplies and several different diagrammatic notations for representing the different aspects of a system under development, it lacks simulation and verifiability capabilities.

UML 2.0 is composed of several diagram types activity, sequence, use case, class, timing and many others). Activity diagrams have been added to the UML quiet late. They have always been poorly integrated, expressive lacked, and did not have an adequate semantic. UML 2.0 replaces 'activity diagram' concepts of UML 1.5 based on stating machines with activity modeling that is supposedly based on Petri Net semantics (Borger et al., 2000). Activities are suitable to model web processing, web service composition (Artagna & Pernici, 2007), business process modeling, workflow management

systems, system integration and even basic software operation.

The following are some main properties of UML activity diagrams:

- Activity diagram nodes have flow-of-control constructs like synchronization, decision, concurrency and sequence. These are fundamentally similar to those of Associative Petri Nets.
- Activity diagram semantics are based on token flows. Tokens can contain objects, data, control information. Tokens are normally distinguishable through an individual time-stamp.
- UML activities try to deal with many different levels of activities: 1) fundamental 2) basic 3) intermediate 4) complete 5) structured, 6) complete structured and 7) extra structured. Each level adds its own constructs addressing a particular area. E.g. structured activities address traditional programming language modeling. Other activity classes like fundamental and basic are ideal for high level modeling and business process modeling.

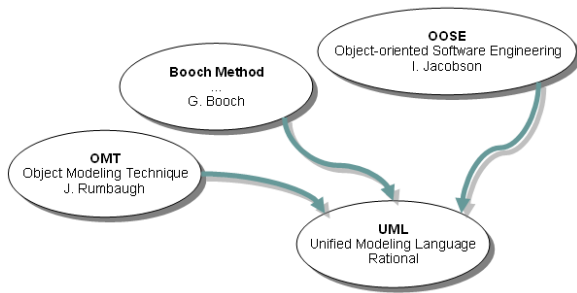


Fig. 1 Three Sources of UML-Standardized by
OMG

Associative Petri Nets (APNs) are a popular technique for modeling the control flow dimension of workflows (Shih, et al., 2007). When modeling workflows, people tend to draw nodes that represent tasks or activities, and arrows between the nodes that represent sequencing of activities. The resulting diagrams look like Associative Petri Nets. Thus, Associative Petri Nets seem a natural technique for modeling workflows (Ellis, & Nutt, 1993). The following arguments are often used to support this: Associative Petri Nets are graphical. They have a formal semantics and can express most of the desirable routing constructs. There is an abundance of analysis techniques for proving properties about them, and finally they are vendor-independent. Some of reasons following items are UML transform into Associative Petri Net (Petriu & Shen, 2002; Kristensen, et al., 2004; Baresi & Pezze, 2007).

In this paper, the study proposes an approach of translating activity diagram into Associative Petri Nets and improving the some drawbacks of UML2 activity diagrams. This work can improve the simulation and verifiability capabilities of activity diagram and provides the systematic procedure to reduce complexity of translating activity diagrams into associative Petri net.

2 Problem Formulation

In this section we will first introduced the definition of an activity diagram and various types of activity subnets. Next, the definition of Associative Petri Net and various types of associative production rules (APRs) will be introduced. Final, a mechanism of mapping activity subnets to APRs is provided.

2.1 Overview of the activity diagram

Unified Modeling Language (UML) is the standard notation of Object Management Group (OMG) for object-oriented modeling. It is easy, graphical and appealed, but it still too imprecise in several cases.

Activity diagrams are a technique to describe procedural logic, business process, and work flow. In many ways, they play a role similar to flowcharts, but the principal difference between them and flowchart notation is that they support parallel behavior (Fowler, 2004).

An activity diagram is a special form of state machine intended to model computations and workflows (Fowler, 2004). The states of the activity diagram represent the states of executing the computations, not the states of an ordinary object. Normally, an activity diagram assumes that computations are preceded without external event-based interruptions.

An activity diagram contains activity states. An activity state represents the execution of a statement in a procedure of the performance of an activity in a workflow. Instead of waiting for an event, as in a normal wait state, an activity state waits for the completion of its computation. When the activity completes, then execution proceeds to the next activity state within the graph. A completion transition in an activity diagram fires when the preceding activity is completed. Activity states usually do not have transitions with explicit events, but they may be aborted by transitions on enclosing states. (Rumbaugh, et al., 1999)

An activity diagram may also contain action states, which are similar to activity states. Except that they are atomic and do not permit transitions while being active. Action states should usually be used for short bookkeeping operations.

An activity diagram may contain branches, as well as forking of control into concurrent threads. Concurrent threads represent activities that can be performed concurrently by different objects or persons in an organization. Frequently concurrency arises from aggregation, in which each object has its own concurrent thread. Concurrent activities can be performed simultaneously or in any order. An activity diagram is like traditional flow chart except it permits concurrent control in addition to sequential control-a big difference. (Rumbaugh, et al., 1999)

The activity diagram contains seven types of state: 1) Action state, 2) Initial/Final action state, 3) Transition, 4) Branching, 5) Fork and join, 6) Object flow, and 7) Swimlanes.

(1)Action states: In the flow of control, action stated is modeled by an activity diagram. It might evaluate some expression that sets the value of an attribute or returns some value. Alternately, it might call an operation of an object, send a signal to an object, or even create or destroy an object. These executable, atomic computations are called action

states because they are states of the system. Each represents the execution of an action.

(2)Initial/Final action state: The initial action state can express the first action state in activity diagram and the start state is represented by a solid cycle. The final action state can express final action state in activity diagram and the final state is represented by a concentric cycle.

(3)Transitions: When the action or activity of a state completes, flow of control passes immediately to the next action or activity state. It's specify this low by using transitions to show the path from one action or activity state to the next action activity state (Booch, et al., 1999).

(4)Branching: Simple sequential transitions are common, but they aren't the only kind of path. You will need to model a flow of control. As in a flowchart, you can include a branch, which specifies alternate paths taken based on some Boolean expression (Booch, et al.). It represents a branch as a diamond. A branch may have one incoming transition and two or more outgoing ones. On each outgoing transition, you place a Boolean expression, which is evaluated only once on entering the branch. Across all these outgoing transitions, guards should not overlap (otherwise, the flow of control would be ambiguous), but they should cover all possibilities (otherwise, the flow of control would freeze).

(5) Forking and joining: Simple and branching sequential transitions are the most common paths you'll find in activity diagrams (Booch, et al., 1999). However, especially when you are modeling workflows of business processes, you might encounter flows that are concurrent. In the UML, you use a synchronization bar to specify the forking and joining of these parallel flows of control. A synchronization bar is rendered as a thick horizontal or vertical line.

For example, considering the concurrent flows involved in controlling an audio-animatronics device that mimics human speech and gestures. A fork represents the splitting of a single flow of control into two or more concurrent flows of control. A fork may have one incoming transition and two or more outgoing transitions. Each of which represents an independent flow of control. Below the fork, the activities associated with each of these paths continue in parallel. Conceptually, the activities of each in these flows are truly concurrent. In a running system, these flows may be either truly concurrent (in the case of a system deployed across multiple nodes) or sequential yet interleaved (in the case of a system deployed across one node). Thus they are only gave the illusion of true concurrency.

A join represents the synchronization of two or more concurrent flows of control. A join may have two or more incoming transitions and one outgoing transition. Above the join, the activities associated with each of these paths continue in parallel. At the join, the concurrent flows synchronize, meaning that each wait until all incoming flows has reached the join. At which point, one flows of control continues on below the join.

(6)Object flow: An activity diagram can show the flow of object values, as well as the flow of control. An object flow state represents an object that is the input or output of an activity (Fowler, 2004). For an output value, a dashed arrow is drawn from an activity to an object flow state. For an input value, a dashed arrow is drawn from an object flow state to an activity. If an activity has more than one output value or successor control flow, the arrows are drawn from a fork symbol. Similarly, multiple inputs are drawn to a join symbol.

(7)Swimlanes: The activities in an activity diagram can be partitioned into regions, which are called swimlanes from their visual appearance as regions on a diagram separated by dashed lines. A swimlane is an organizational unit for the contents of an activity diagram. It has no inherent semantics, but can be used as the modeler desires. Often, each swimlane represents an organizational unit within a real-world organization.

It is often useful to organize the activities in a model according to responsibility. For example, by grouping together all the activities handled by one business organization. This kind of assignment can be shown by organizing the activities into distinct regions separated by lines in the diagram. Because of their appearance, each region is called a swimlane.

2.2 Associative Petri Nets (APN)

Petri Nets (PN) has its original in Carl Adam Petri's dissertation. It submitted in 1962 to the faculty of Mathematics and Physics at the Technical University of Darmstadt, West Germany. PN is a graphical and mathematical modeling tool applicable to many systems. (Murata, 1989) They are a promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. However, APN is different with current modeling tools. It belongs to dynamic modeling, except for providing with parallel and distributed system of dynamic behavior. On other hand, they support of graphical for simple hierarchical theory and support mathematics for qualitative and

quantitative analysis, further development on special flow.

We shall derive the definition of a generalized APN and augment the association production rules (APRs) of APN by adding association rule operators and equipping the nets with reasoning facilities. Based on the generalized associative Petri Net model, a systematic procedure of Associative Petri Net model construction methodology had been proposed.

2.2.1 Definition of APN

An APN is a directed graph, which contains three types of nodes: places, squares, and transitions where circles represent places, squares represent thresholds of association degree, and bars represent transitions. 3 each place may contain a token associated with a truth-value between zero and one. Each transition is associated with a CF value between zero and one. Directed arcs represent the relationships between places. A generalized APN structure is and it can be defined as a 13-tuple:

$$APN = (P, T, S, D, \Lambda, \Gamma, I, O, C, \alpha, \beta, W, Th)$$

where

$P = \{p_1, p_2, \dots, p_n\}$ is a finite set of places,

$T = \{t_1, t_2, \dots, t_i\}$ is a finite set of transitions,

$S = \{s_1, s_2, \dots, s_m\}$ is a finite set of supports,

$D = \{d_1, d_2, \dots, d_n\}$ is a finite set of propositions,

$\Lambda = \{\tau_1, \tau_2, \dots, \tau_m\}$ is a finite set of thresholds of the supports,

$\Gamma = \{\gamma_1, \gamma_2, \dots, \gamma_i\}$ is a finite set of thresholds of the confidences,

$P \cap T \cap D = \emptyset, |P| = |D|$

$I: T \rightarrow P^\infty$ is an input function, a mapping from transitions to bags of places,

$O: T \rightarrow P^\infty$ is an output function, a mapping from transitions to bags of places,

$C: T \rightarrow [0,1]$ is an association function that assigns a real value between zero to one to each transition,

$\alpha: P \rightarrow [0,1]$ is an association function, a mapping from places to real values between zero and one,

$\beta: P \rightarrow D$ is an association function, a bijective mapping from places to propositions,

$W: S \rightarrow [0,1]$ is an association function that assigns a real value between zero to one to each support,

$Th: S \rightarrow \Lambda, T \rightarrow \Gamma$ is an association function that defines a mapping from support and transition to thresholds of confidences.

Let A be a set of directed arcs. If $p_j \in I(t_i)$, then there exists a directed arc $a_{j_i}, a_{j_i} \in A$, from place p_j to transition t_i . If $p_k \in O(t_i)$, then there exists a directed arc $a_{i_k}, a_{i_k} \in A$, from transition t_i to place p_k . If $W(s_m) = w_m, w_m \in [0,1]$, then the support s_m is said to be associated with a real value w_m . If $C(t_i) = c_i, c_i \in [0,1]$, then transition t_i is said to be associated with a real value c_i . If $\beta(p_j) = d_j, d_j \in D$, then place p_j is said to be associated with proposition d_j .

An associative Petri net with some places containing tokens is called a marked associative Petri net. In a marked associative Petri net, the token in a place p_j is represented by a labeled dot $\overset{\alpha(p_j)}{\bullet}$.

The token value in a place $p_j, p_j \in P$, is denoted by $\alpha(p_j)$, where $\alpha(p_j) \in [0,1]$. If $\alpha(p_j) = y_i$ and $\beta(p_j) = d_j$, then it indicates that the degree of truth of proposition d_j is y_i .

2.2.2 Definition of APN

If the antecedent portion or consequence portion of an associative production rule (APR) contains "and" or "or" connectors, then it is called a composite APR. The composite APR can be distinguished into the following five rule-types:

Type 1: IF d_j THEN d_k (CF = c). The reasoning process of this type of association rule can be represented by (1)

$$\alpha(p_k) = \alpha(p_j) * c \quad \text{when } s \geq \tau, c \geq \gamma \quad (1)$$

Type 2: IF d_{j1} and d_{j2} and...and d_{jn} THEN d_k (CF = c). The reasoning process of this type of association rule can be represented by (2).

$$\alpha(p_k) = \text{Min}\{\alpha(p_{j1}), \alpha(p_{j2}), \dots, \alpha(p_{jn})\} * c$$

$$\text{when } s_i \geq \tau_i, c \geq \gamma, i = 1, 2, \dots, n \quad (2)$$

Type 3: IF d_j THEN d_{k1} and d_{k2} and...and d_{kn} (CF = c). The reasoning process of this type of association rule can be represented by (3).

$$\alpha(p_{k1}) = \alpha(p_j) * c, \alpha(p_{k2}) = \alpha(p_j) * c, \dots,$$

$$\alpha(p_{kn}) = \alpha(p_j) * c \quad \text{when } s \geq \tau, c \geq \gamma \quad (3)$$

Type 4: IF d_{j1} or d_{j2} or...or d_{jn} THEN d_k (CF = c). The reasoning process of this type of association rule can be represented by (4).

$$\alpha(p_k) = \max\{\alpha(p_{j1}) * c, \alpha(p_{j2}) * c, \dots, \alpha(p_{jn}) * c\}$$

$$\text{when } s_i \geq \tau_i, c_i \geq \gamma_i, i = 1, 2, \dots, n \quad (4)$$

Type 5: IF d_j THEN d_{k1} or d_{k2} or...or d_{kn} (CF = c_i). The reasoning process of this type of association rule can be represented by (5).

$$\alpha(p_{k1}) = \alpha(p_j) * c, \alpha(p_{k2}) = \alpha(p_j) * c, \dots, \alpha(p_{kn}) = \alpha(p_j) * c_n$$

$$\text{when } s_i \geq \tau_i, c_i \geq \gamma_i, i = 1, 2, \dots, n \quad (5)$$

Using this simple mechanism, all the APRs can be mathematically and graphically illustrated. By

carefully connecting related place and assigning reasonable values of CFs to transitions, it can come up with an APN that can make a decision based on the expertise during its construction.

3 Mapping activity diagram to APN

Activity diagrams are a technique to describe procedural logic, business process, and workflow. It is a graphical and appealing. However, in many ways, activity diagram lacks support for simulation, formal semantics and verifiability capabilities.

APN is a graphical and strong modeling tool. In many ways, they play a role similar to activity diagram, and APN has formal semantics, simulation and verifiability capabilities. Therefore, we purpose APN to improving activity diagram for system analysis and workflow in this study.

3.1 Mapping activities to APN notation

Activity diagram is lack of formal semantic, simulation, verifiability, and capability; thus, we propose an APN building through activities. The various characteristics of activities are as follows:

(1)Action node to conversion: Action node is a rounded rectangular, in rectangle marked with its activities, status and process. It is can express action state and process. The theory is similar to Place by APN. The Place is a cycle, its express activity is flowed and stated by token transition, and marked its activity properties on a nearby Place.

(2)Transition to conversion: Transition (AD) is a direction. It can be represented the transition between action flows and workflows. Transition (APN) is a rectangle. The Transition (APN) can be fired to next state by token. Through description of above the Transition (AD) function is similar to Transition (APN). Therefore, in this paper, it has been mapping Transition (AD) to Transition (APN).

(3)Initial/Final node to conversion: Initial node and Final node are represented the starting place and final place of activity diagram. The Initial node is a solid cycle, and the Final node is a concentric cycle. APN are circles starting node (p_s) and final node (p_g). They flow-through by token, and the node only presents as a node or a child node. It's similar with Initial node and Final node of activity diagram.

(4)Branching to conversion: Branching is a diamond, it can represent workflow for flow decision, e.g. loop structure and Boolean condition, etc. Therefore, it can be called a decision node. The decision node has a single incoming flow and several guarded outbound flows. Each outbound

flow has a guard: a Boolean condition placed inside square brackets. Each time it reaches a decision, it can take only one of the outbound flows, so the guards should be mutually exclusive. However, we deal with a conditional behavior, and are usually accompanied by a merge node, and the merge node has multiple input flows and a single output flows. A merge marks the end of conditional behavior started by a decision. Due to APN is no single notation for express decision state, therefore, we are going to show more than one notation into a graphics to illustrate the state of the decision node.

(5)Fork/Join node to conversion: Fork node and Join node deal with the parallel computing in action flow. When a parallel computing occurs, it will use Fork node to destroy several activities, and then all the parallel computing to Join node end of this activity computing. It is represented by a solid rectangle. Owing to APN has parallel computing capability, and it represents notation for "transition", and it represented by a rectangle. They flow-through by token and fire of transition to finish this work.

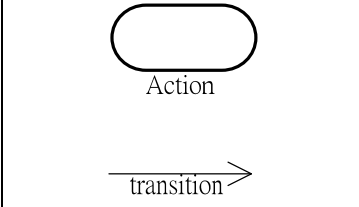
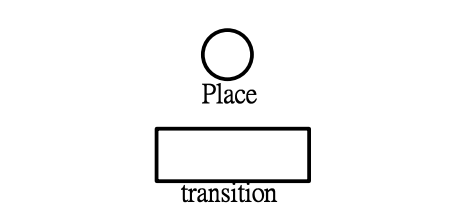


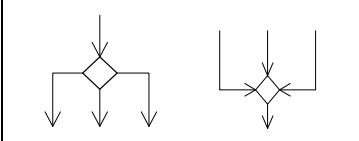
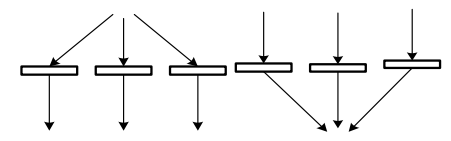
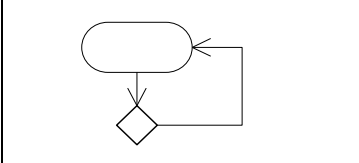
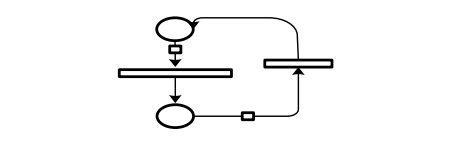
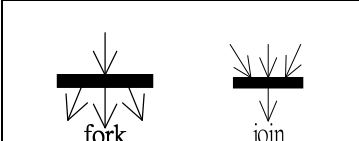
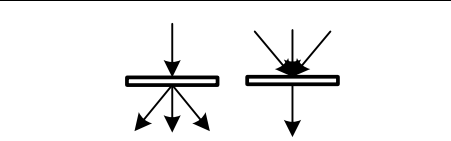
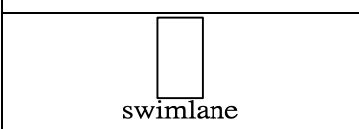
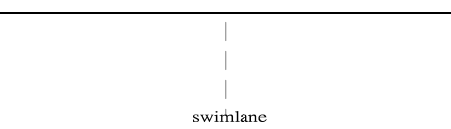
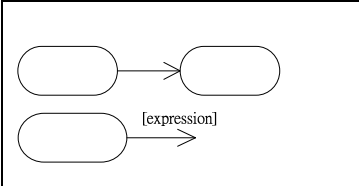
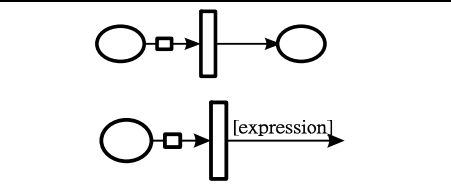
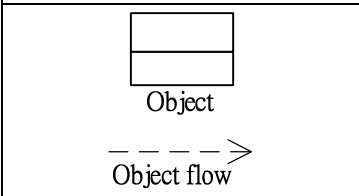
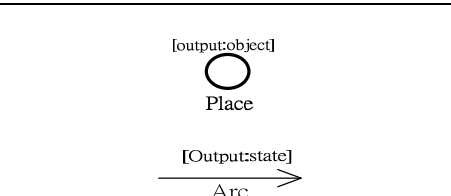
(6)Object node and object flow to conversion: An object flow state represents an object that is the input or output of an activity (Fowler, 2004). The object flow represents as a directed dashed line. The Object node is represented by a rectangle, and it divides into upper and lower level: The upper level is express object name, and lower level is express object state. In this study, we shall express object node by Place node of APN, and mark object properties on Place. The object flow represents by arc, and mark object state on is to express object flow.

3.2 The transformation activity diagrams into APNs

According to Table 1, the basic model contains five types of structures: 1) activity sequence structure, 2) activity fork and activity join structure, 3) activity branching structure, 4) activity iteration state structure, 5) activity object flow structure. We adopt the five kinds of types to translate activity diagram into associative Petri net.

(1)Activity sequence structure: The activity sequence is the most frequently used in activity diagram. Through a sequence structure can explicitly state the workflow processes and steps. The activity sequence structure, and starts by Initial node through the directed arcs to transform Action node. An example, the structure is an activity diagram for activity sequence basic patterns. If it will accomplish this procedure, it has to transit action 1, action 2 and action 3 to perform this procedure.

Table 1 The general notation mapping

Activity type	Activity Diagram notation	Associative Petri Nets notation
Action node/transition		
Initial/Final node		
Branching node		
		
Fork/join node		
Swimlane		
Activity edges		
Object node/flow		

According to data, it can convert Fig. 2a into Fig. 2b. The flow-through by token and fire by Transition; it can complete to express a simple process for APN.

(2)Activity fork and activity join: While dealing with a parallel computing, then fork will be used to process the partition of activity. If all states reach a join node is used to combine those actions and fire to next action. For example, Fig. 3a; it's a parallel computing state. In Fig. 3a, the action 1 is

forked to three actions A, B and C through fork node. The A, B and C are combined by join node, and the workflow is moved to action 2. The fork and join of activity diagram is shown in Fig. 3a. The mapping graph of APN is shown in Fig. 3b. According to Table 1, the fork node and join node convert into a "transition", they flow-through by token and fire of transition to finish this work. If the token meet a fork transition the token is forked and

transmitted to A, B and C places by enabling transition. Then three tokens moved to next action 2 through enabling joint transition, such as in Fig. 3.

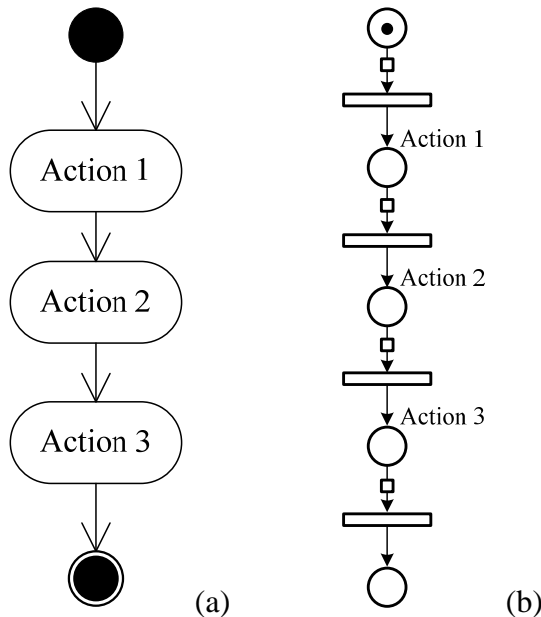


Fig. 2 Activity sequence Structure (a)AD (b)APN

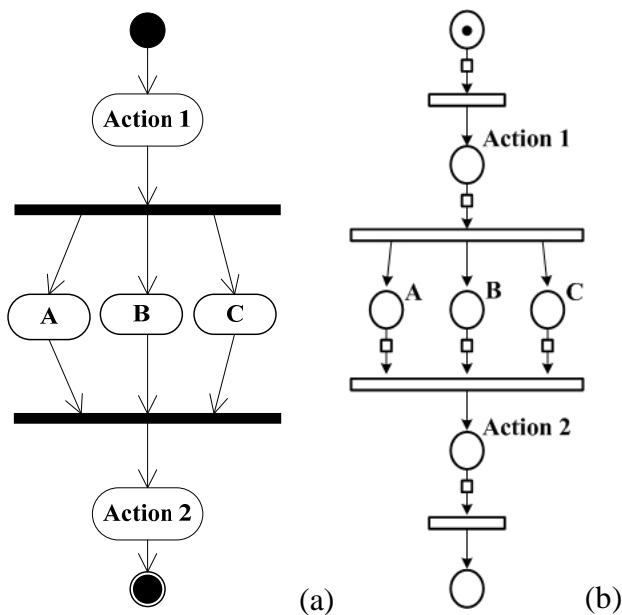


Fig. 3 Activity fork/join Structure (a)AD (b)APN

(3)Activity Branching: Branching is using in workflow decision state. It can decide the workflow direction of the change process. The Fig. 4a indicates a branching state by activity diagram, and the action1 transmitted to action4 that through a branching. Through judge of branching, it can represent the flow of action state in activity diagram, like Fig. 4a. If “N” is rejected, then it transfer to action 2, and other else will transfer to action 3. Finally, it combines this flow by branching, and transmits to action 4. According to Table1, the Fig.

4b is activity diagram transform APN’s graph. It can express decision of multi flow in APN, and flow-through by token and fire of transition to represent one of branching state.

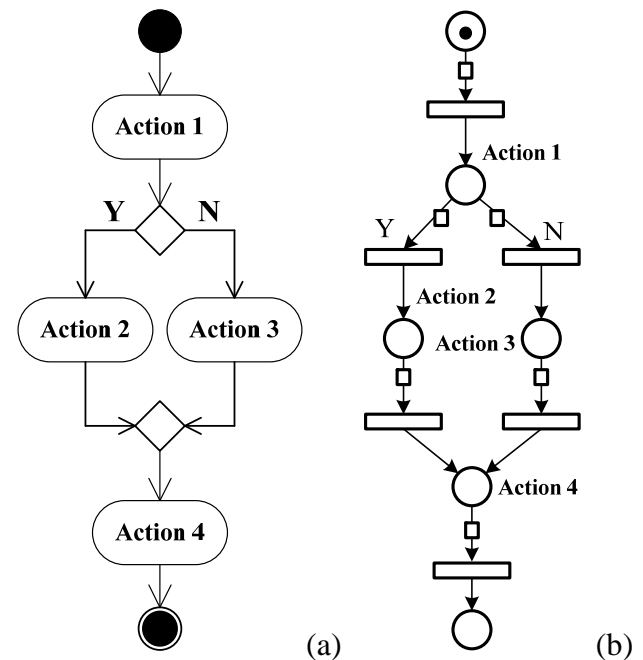


Fig. 4 Activity Branching Structure (a)AD (b)APN

(4)Activity branching iteration state: branching is using in workflow decision state. However, the workflow contains two types of decision state: one is “or” state, another state for “loop”. In workflow or program language, the structure will be used to system development and workflow planning. Through the decision of Branching express “loop” state. The Fig. 5a shows an activity state. The action 2 is transmitted to action 3, that need one of decision by branching. If flow is “y”, it then transmits to action 3 and other else transmits to action 1. In the Table1, APN combines with several notations to express a “loop” structure. Therefore, through Table1, the result of notation mapping transforms activity diagram into APN graph is show in Fig. 5b.

(5)Activity object flow: An object flow state represents an object that is the input or output of an activity. In the Fig. 6a, it represents an action 1 to output an object. The object name is “object”, and its state is “state”. In this example, illustrating the action1 converts to action 2, and then output an object. According to Table 1, the result of mapping, it transforms all notations, and shows in Fig. 6b. In Fig. 6b, it represents the object output and object input by APN. The object name is place and the state is arc.

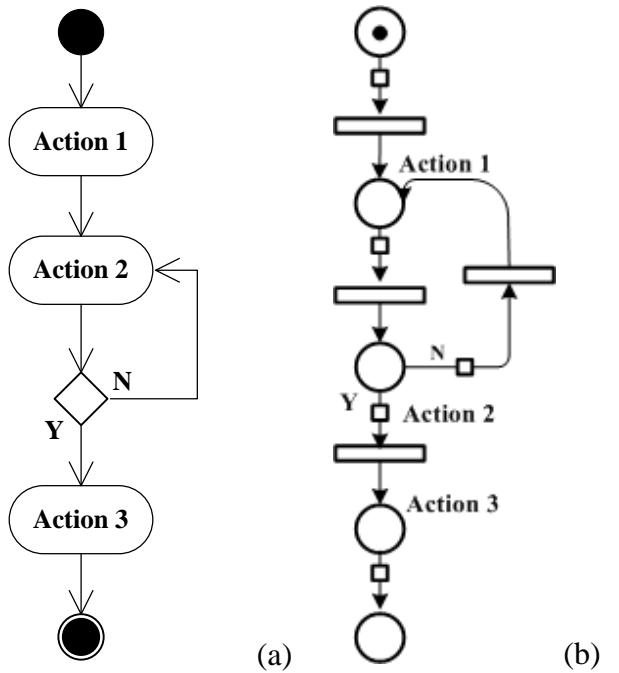


Fig. 5 Activity branching iteration state (a)AD (b)APN

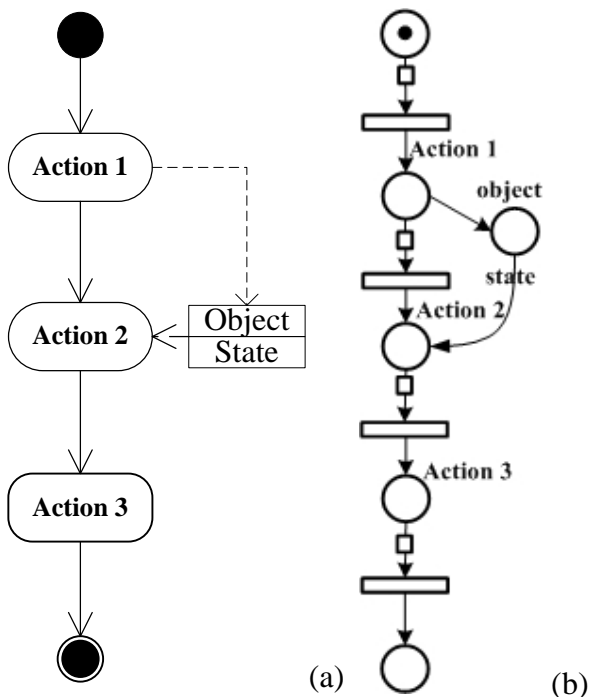


Fig. 6 Activity object flow (a)AD (B)APN

4 A Methodology of Translating Activity diagrams to APN

A methodology based on a five-stage schema was developed for activity diagrams transfer to APN. The five steps correspond to 1) Initial node determination, 2) Finding basic structure models, 3) Final node determination, 4) AD-to APN model mapping, 5) Models combination, and 6) APN

model accomplishment were shown in Fig. 7 and described as follows:

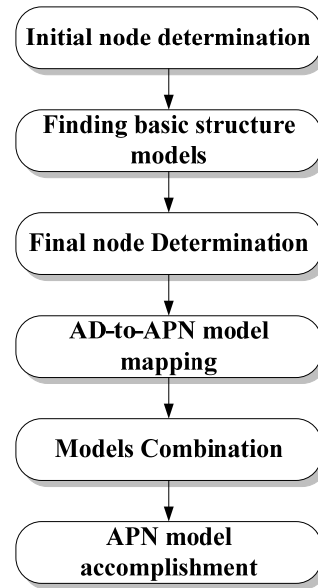


Fig. 7 Transform steps of an AD-to-APN model

Step1 Initial node determination: In this step, it incises the starting place p_i . If a node is no parent node, it calls this node as a start node p_i . It is determined as a start node. Next step, an incision place are decided for model combination, therefore, next node of p_i facilitates follow-up action. Finally, p_i will be mapped and next node of p_i .

Step2 Finding basic structure models: According to section 3.2 description, the basic structure models contains five types: 1) activity sequence structure, 2) activity fork and join structure, 3) activity branching structure, 4) activity iteration structure, 5) activity object flow structure. On the other hand, we facilitate for model combination, therefore, we shall incision the basic model for parent node and child node.

Step3 Final node determination: In this step, the main purpose of deciding the final node of activity diagram, determining the model range, and mapping the Final node. If a node is no child node then called this node is a final node p_f . Through the p_f , the action of activity diagram can be judged between p_i and p_f . Due to p_f is no child node, therefore, it will incise p_f and present node of p_f for combining point. Finally, it will map p_f and present node of p_i .

Step4 AD-to-APN model mapping: As describing in section 3.1 and 3.2, the various types basic structure models of activity diagrams are transformed into corresponding APN models.

Step5 Models Combination: Due to activity diagram is a directed graph, and workflow transmitted by transition, it's the same as the

required permutations and combinations. Otherwise this graph through the conversion would be an error graph. Therefore, for each place state to match, if the place of the same states, it combines this model and just leaves one place, and repeats above then can output a goal graph.

Step6 APN model accomplishment: According to above steps, we can obtain an APN model.

5 Conclusion

Unified Modeling Language (UML) is the standard notation of Object Management Group (OMG) for object-oriented modeling. UML activity diagram (AD) is widely used to workflow and system flow in system analysis. However, activity diagram lacks

support for simulation, dynamic semantics limits and verifiability capabilities. APN are a popular technique for modeling the control flow dimension of workflows and APN. It has a complete semantics, simulation and verifiability capabilities. The activity diagrams are widely used to workflow and system flow in system analysis. However, activity diagram lacks support for simulation, dynamic semantics limits and verifiability capabilities. Therefore, in this paper, we propose a transform methodology with AD-to-APN model. This work can improve of the simulation and verifiability capabilities for activity diagram. In the further, a case is used for explaining and illustrating this concept.

References:

- [1] W.M.P. van der Aalst, "The application of Petri Nets to workflow management," *The Journal of Circuits, Systems and Computers*, vol. 8, no. 6, pp. 21-66, 1998.
- [2] Fan Yang, Zhi-Mei Wang, A Mobile Location-based Information Recommendation System Based on GPS and WEB2.0 Services, WSEAS TRANSACTIONS on COMPUTERS, Issue 4, Volume 8, April 2009, pp. 725-734.
- [3] Chang-Chun Tsai, Cheng-Jung Lee, Shung-Ming Tang, The Web 2.0 Movement: Mashups Driven and Web Services, WSEAS TRANSACTIONS on COMPUTERS, Issue 8, Volume 8, August 2009, pp.1235-1244.
- [4] Zhi-mei Wang, Ling-Ning Li, Enable collaborative learning: an improved e-learning social network exploiting approach, Proceedings of the 6th Conference on WSEAS International Conference on Applied Computer Science, Vol.6, 2007, pp.311-314.
- [5] S. S. Alhir, Learning UML. Sebastopol, CA: O'Reilly & Associates, Inc., 2003.
- [6] R. Agrawal, T. Imielinski, and A. Swami, "Database Mining: A Performance Perspective," *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 6, pp. 914-925, Dec. 1993.
- [7] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," *Proc. ACM SIGMOD Conf. Management of Data*, pp. 207-216, May 1993.
- [8] D. Artagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Transactions on Software Engineering*, vol. 33 no. 6, pp. 369-384. Jun. 2007.
- [9] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. Int'l Conf. Very Large Databases (VLDB '94)*, pp. 487-499, Sept. 1994.
- [10] E. Borger, A. Cavara and E. Riccobene, "An ASM Semantics for UML activity diagrams," *Proceedings of the 8th International Conference on Algebraic Methodology and Software Technology*, Iowa City, Iowa, USA, pp. 293-308. May 2000.
- [11] L. Baresi and M. Pezze, "Improving UML with Petri Nets," *Electronic notes in Theoretical Computer Science*, vol.44, no. 2, pp. 107-119, Jul. 2007.
- [12] S. Chen, J.S. Ke, and J. Chang, "Knowledge Representation Using Fuzzy Petri Nets," *IEEE Trans. Knowledge and Data Eng.*, vol. 2, no. 3, pp. 311-319, Sept. 1990.
- [13] C.A. Ellis and G. J. Nutt. "Modelling and Enactment of Workflow Systems," *Lecture Notes in Computer Science (LNCS)*, vol. 691, pp. 1-16, Springer, 1993.
- [14] R. Eshuis and R. Wieringa, "Comparing Petri Net and activity diagram variants for workflow modeling - a quest for reactive Petri Nets," *Lecture Notes in Computer Science (LNCS)*, vol. 2472, pp. 321-351, Springer-Verlag 2003.

- [15] M. Fowler, UML distilled: a brief guide to the standard object modeling language (3rd ed.). Boston, MA: Addison-Wesley Publishing, 2004.
- [16] J. L. Garrido and M. Gea, "A Colored Petri Net Formalization for a UML-based Notation Applied to Cooperative System Modeling, Interactive Systems: Design, Specification and Verification," Lecture Notes in Computer Science (LNCS), vol. 2545, pp.16-28, Springer, 2002.
- [17] K.M. van Hee, "Information Systems Architecture A Practical and Mathematical Approach," Technische Universiteit Eindhoven, <http://www.wis.win.tue.nl/~wsinhee/sm1/>, 2005.
- [18] J. Han and M. Kamber, Data Mining Concepts and Techniques, pp. 226-230. Morgan Kaufmann, 2001.
- [19] Z. Hu and S. M. Shatz, "Mapping UML diagrams into a Petri Net notation for system simulation," Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004), pp. 213-219, Jun. 2004.
- [20] L.M. Kristensen, J.B. Jorgensen, K. Jensen, "Application of Coloured Petri Nets in System Development," Lecture Notes in Computer Science, vol. 3098, pp. 626-685, Springer, 2004.
- [21] J. LI, X. Dai, Z. Meng, J. Dou and X. Guan, "Rapid design and reconfiguration of Petri net models for reconfigurable manufacturing cells with improved net rewriting systems and activity diagram," Computers & Industrial Engineering, Vol. 57, no. 2, pp. 1431-1451, 2009.
- [22] T. Murata, "Petri Nets: Properties, Analysis and Applications," Proc. IEEE, vol. 77, no. 4, pp. 541-580, 1989.
- [23] D. C. Petriu and H. Shen, "Applying the UML Performance Profile: Graph Grammar based derivation of LQN models from UML Specifications," Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools, pp. 159-177, London UK, 2002.
- [24] J. Rumbaugh, I. Jacobson, and G. Booch, The unified modeling language user guide. Boston, MA: Addison-Wesley Publishing, 1999.
- [25] J. Rumbaugh, I. Jacobson, and G. Booch, The unified modeling language reference manual. Boston, MA: Addison-Wesley Publishing, 1999.
- [26] T. S. Staines, "Intuitive mapping of UML 2 activity diagrams into fundamental modeling concept Petri Net diagrams and Colored Petri Nets," 15th Annual IEEE *International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, pp. 191-200, Apr. 2008.
- [27] D. H. Shin, H. S. Chiang and B. Lin, "A generalized associative Petri Net for reasoning," IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 9, pp. 356-366, 2007.
- [28] UML 2 Superstructure Specification. V2.11, Object Management Group (OMG), <http://www.omg.org/technology/documents/formal/uml.htm/>, 2009.
- [29] D.S. Yeung and E.C.C. Tsang, "A Multilevel Weighted Fuzzy Reasoning Algorithm for Expert Systems," IEEE Trans. Systems, Man, and Cybernetics, vol. 28, no. 2, pp. 149-158, 1998.