

## FORMALIZING ACTIVITY DIAGRAM OF UML BY PETRI NETS

**Ivana Tričković**

University of Novi Sad, Faculty of Science, Institute of Mathematics  
Trg D. Obradovića 4, 21000 Novi Sad, Yugoslavia  
e-mail: ivanatr@uns.ns.ac.yu

### **Abstract**

This paper describes a possible application of the Petri nets to specify the dynamics of information systems. The Petri nets are a mathematical tool allowing formal specification of the system dynamics. A formal procedure is proposed for transforming the activity diagram of the Unified Modeling Language into a Petri nets model. On the basis of this transformation it is possible to accomplish verification of the dynamic model of the real system, i.e. to evaluate whether the activities and their order are well defined. It is also possible to solve the problem of concurrency and synchronization of the activities in the system, as well as to optimize the dynamic model.

*AMS Mathematics Subject Classification (1991):* 68Q90

*Key words and phrases:* information systems, object-oriented methods, activity diagram, Unified Modeling Language, dynamic model, formal specification, Petri nets

## **1. Introduction**

The process of the development of an information system encompasses specification of the static and the dynamic structure of the system. In the last

few years an object-oriented approach has been dominant in the development of information systems. The approach is based on the fact that the objects and their relationships represent the real characteristics of the system under development. The system is a set of mutually connected objects. Each state of the system is defined by the states of the objects. The system functions are accomplished as operations over the objects that can change the state of the objects.

There are different approaches to the object-oriented development of information systems [1, 2, 3], one of them being *Unified Modeling Language* [4]. This language proposes a set of modeling concepts that help building a semantically rich model of the real system. The set of concepts includes object, class, relationships between classes and objects (association, aggregation, specification/generalization), state, event, and function. A static model is presented by class diagrams and object diagrams. A dynamic model is presented by statechart diagrams, activity diagrams, sequence diagrams, and collaboration diagrams. These diagrams are informal software development techniques, and provide the user an easy way to describe the system, but they have many serious drawbacks pointed out in several papers [5, 6, 7, 8, 9]. The lack of formality in these diagrams prevents the evaluation of completeness, consistency, and content in the requirements and design specification. A dynamic model has many disadvantages. It lacks the formal semantics, which yields an ambiguous model and problems with the transition from the analysis to design and implementation, problems with modeling the process concurrency, synchronization, and mutual exclusion.

An approach to overcoming this problem is to specify the static and dynamic characteristics using formal methods. Hence, this paper is concerned with a formal specification of the system dynamics. The possible application of the theory of Petri nets to specify the system dynamics is described. A formal procedure for transforming the activity diagram of the Unified Modeling Language into a Petri nets model is proposed.

## 2. Notions of the theory of petri nets

Petri nets are a mathematical tool used for describing the system dynamics [10, 11, 12, 13]. Useful information about the structure and dynamic behavior of the modeled system can be obtained by analyzing the Petri nets

model. Based on this information, the system can be evaluated, and its improvements and changes proposed. *Place*, *transition* and *token* are basic concepts of the theory of Petri nets.

A Petri net structure is defined as four-tuple,  $C = (P, T, I, O)$ .  $P = \{p_1, p_2, \dots, p_n\}$  is a finite set of *places*,  $n \geq 0$ .  $T = \{t_1, t_2, \dots, t_m\}$  is a finite set of *transitions*,  $m \geq 0$ . The set of places and the set of transitions are disjoint,  $P \cap T = \emptyset$ .  $I : T \rightarrow P^\infty$  is the *input function*, a mapping from transitions to bags of places.  $O : T \rightarrow P^\infty$  is the *output function*, a mapping from transitions to bags of places.

An essential feature of the Petri nets is that they are dynamic systems. Tokens are used to reflect this dynamic behavior, and they are assigned to the places of a Petri net. The presence of a token in a place is interpreted as the condition associated with that place being fulfilled. In another interpretation,  $k$  tokens are put into a place to indicate that  $k$  data items or resource are available. The number and position of tokens may change during the execution of a Petri net. The *marking*  $\mu$  is a function that assigns tokens to the places of a Petri net. A marking  $\mu$  of a Petri net  $C = (P, T, I, O)$  is defined as a mapping from the set of places  $P$  to the non-negative set of integers  $N_0 = N \cup \{0\}$ ,  $\mu : P \rightarrow N_0$ .

A Petri net executes by firing transitions, and is controlled by the number and distribution of tokens in the Petri net. A transition fires by removing tokens from its input places and creating new tokens which are distributed to its output places. A transition may fire if it is enabled. A transition is enabled if each of its input places has at least as many tokens in it as arcs from the place to the transition. Multiple tokens are needed for multiple input arcs. The tokens in the input places that enable a transition are its enabling tokens.

Firing a transition will in general change the marking  $\mu$  of the Petri net to a new marking  $\mu'$ . Since only the enabled transition may fire, the number of tokens in each place always remains non-negative when a transition is fired. If there are no enough tokens in any input place of a transition, the transition is not enabled and cannot fire. Transition firings can continue as long as there exists at least one enabled transition. When there are no enabled transitions, the execution halts.

Petri nets are a favorable mathematical tool for system modeling. The major feature of Petri nets is in modeling system that may exhibit concur-

rency. However, modeling by itself is of little use. It is necessary to analyze the modeled system.

The reachability problem is one of the most important problems for the Petri nets analysis. Many other problems in Petri net analysis can be defined by the reachability problem. The set of reachable markings for a given marked Petri net is defined as follows. The marking  $\mu$  is in the set of reachable markings,  $R(C, \mu)$ , if there is any sequence of transition firings which will change the marking  $\mu$  into  $\mu'$ .

### 3. Activity modeling with Petri nets

There are many papers [5, 6, 7, 8, 9] in which the authors emphasized the drawbacks of the methods and techniques currently used in object-oriented modeling. These drawbacks are related to the modeling dynamic and static aspects of a system. In the papers [5, 6, 7], the diagrams for describing dynamic aspects of a system are considered, especially statechart diagram of Unified Modeling Language. The transformation of the statechart diagram of UML into a Petri net model is given. A formalization of the use cases with Object Coloured Petri nets is proposed in the paper [8]. A method for deriving modular algebraic specifications directly from object model diagram of Object Modeling Technique is described in the paper [9]. In this paper the formalization of the activity diagram of UML by Petri net is considered.

#### 3.1. Introduction into activity diagram

Activity diagram is a special case of a statechart diagram of UML in which all states are action states and the transitions are triggered by completion of the actions in the source states. The activity diagram is attached to a class, to the implementation of an operation of the class, or a use case. The purpose of this diagram is to focus on the flows of control and data driven by internal processing.

An action state is a state with an internal action and at least one outgoing transition involving the implicit event of completing the internal action. If there are several outgoing transitions they must have guard conditions. An action state is used to model a step in the execution of an algorithm or procedure. Decisions express the situation when guard conditions are used

to indicate different possible transitions that depend on Boolean conditions. Transitions leaving an action states or decision are called output internal transitions, and other transitions are called input internal transitions. Such transitions are implicitly triggered by the completion of the action in the state. The transition may include guard conditions and actions.

In the following subsections transformation of the activity diagram of Unified Modeling Language into a Petri net model is considered. The transformation of the activity states, internal transitions and decisions is described in a formal way. A definition of the activity diagram and the accompanying Petri nets structure is given.

Furthermore, a complex transition may have multiple source activity states and target activity states. It represents synchronization and/or splitting of control into concurrent threads. A complex transition is enabled when all of the source states are occupied. After a complex transition fires all of its destination states are occupied. The complex transition that represents synchronization is called join, and the one that represents a splitting is called fork.

### 3.2. Transformation of activity states and internal transitions into a Petri nets model

Transformation of the activity diagram into a Petri net model is based on the transformation rules shown in Figures 1-3. Figure 1 shows the rule related to transformation of an activity state and accompanying input and output internal transitions. The activity state  $s_i$  is transformed into the transition  $s_i$  of a Petri net. An input internal transition  $a_i$  of the activity state  $s_i$  is transformed into the input place  $a_i$  of the transition  $s_i$  of a Petri net. The output internal transition  $a_k$  of the activity state  $s_i$  is transformed into the output place  $a_k$  of transition  $s_i$  of a Petri net.

Transformation of a decision and accompanying input and output internal transitions of the activity diagram into places and transitions of a Petri net is shown in Figure 2. The input internal transition  $a_i$  is transformed into the input place  $a_i$  and output internal transitions  $a_j$  and  $a_k$  are transformed into the output places  $a_k$  and  $a_j$  of a Petri net. A decision is modeled with

two or more transitions. The number of transitions in a Petri net is the same as the number of output internal transitions of the decision.

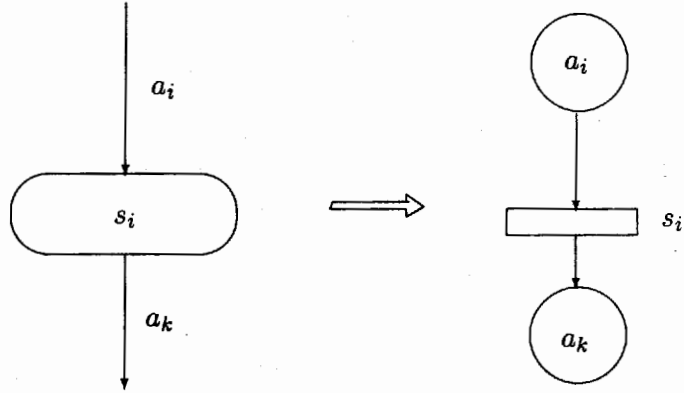


Figure 1: Transformation rule for an activity state

The activity diagram with more than one output internal transition can be modeled as shown in Figure 3. The activity state  $s_i$  is modeled with the activity state  $s'_i$  and decision  $d'_i$ , connected with the internal transition  $a'_i$ . The input internal transition  $a_i$  of activity state  $s_i$  becomes input internal transition of the activity state  $s'_i$ . The output internal transitions  $a_j$  and  $a_k$  become the output transitions of the decision  $d'_i$ . This structure can be transformed into a Petri net according the rules described above and shown in Figures 1 and 2.

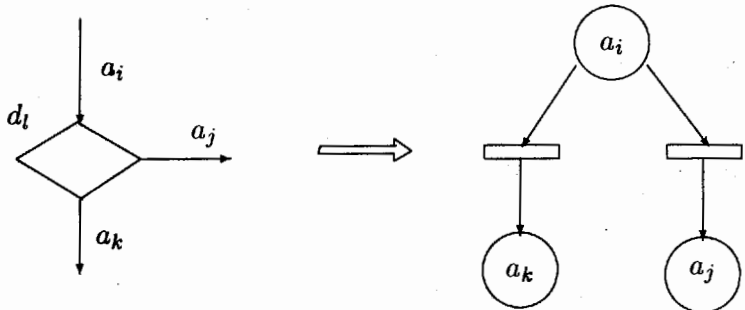


Figure 2: Transformation rule for a decision

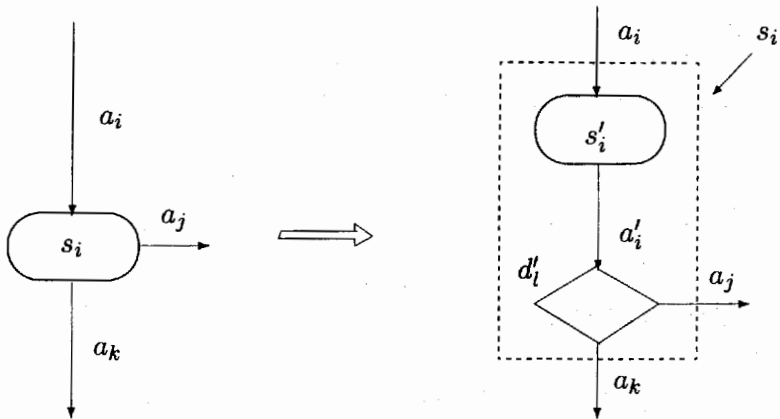


Figure 3: Transformation rule for the activity state with more than one leaving transition

### 3.3. Definition of activity diagram and related Petri nets structure

According to the description of the activity diagram and transformation rules given in Figures 1-3 the definitions of the activity diagram and related Petri net structure can be defined as follows.

An activity diagram can be defined as the tuple  $(S, s_0, Z, A, C, D, In_s, Out_s, In_d, Out_d, In_c, Out_c)$ , where:

$S = \{s_1, s_2, \dots, s_k\}$  is a set of activity states,

$s_0$  is an initial pseudostate,

$Z = \{z_1, z_2, \dots, z_l\}$  is a set of final pseudostates,

$A = \{a_1, a_2, \dots, a_m\}$  a set of internal transitions,

$C = \{c_1, c_2, \dots, c_n\}$  a set of forks and joins,

$D = \{d_1, d_2, \dots, d_p\}$  a set of decisions,

$In_s : S \cup \{s_0\} \cup Z \rightarrow A$  and  $Out_s : S \cup \{s_0\} \cup Z \rightarrow A$  are mappings which define input and output transitions for states,

$In_d : D \rightarrow A$  and  $Out_d : D \rightarrow A$  are mappings which define input and output transitions for decisions, and

$In_c : C \rightarrow A$  and  $Out_c : C \rightarrow A$  are mappings which define input and output transitions for forks and joins.

Based on the definition of the activity diagram a Petri Net structure  $(P, T, I, O, m)$  can be defined as follows:

$P = A,$   
 $T = S \cup \{t_{d_i}^{a_i} \mid d_i \in D, Out_d(d_i) = a_i\} \cup C,$   
 $I(s_i) = In_s(s_i), O(s_i) = Out_s(s_i), s_i \in S,$   
 $I(t_{d_i}^{a_i}) = In_d(d_i), O(t_{d_i}^{a_i}) = \{a_i\},$   
 $I(c_i) = In_c(c_i), O(c_i) = Out_c(c_i),$  and  
 if  $Out(s_0) = a_i$  then  $\mu(a_i) = 1$  else  $\mu(a_i) = 0.$

#### 4. Illustrative example

In this section the transformation of the activity diagram of the UML into a Petri net model is described in an informal way. An automatic teller

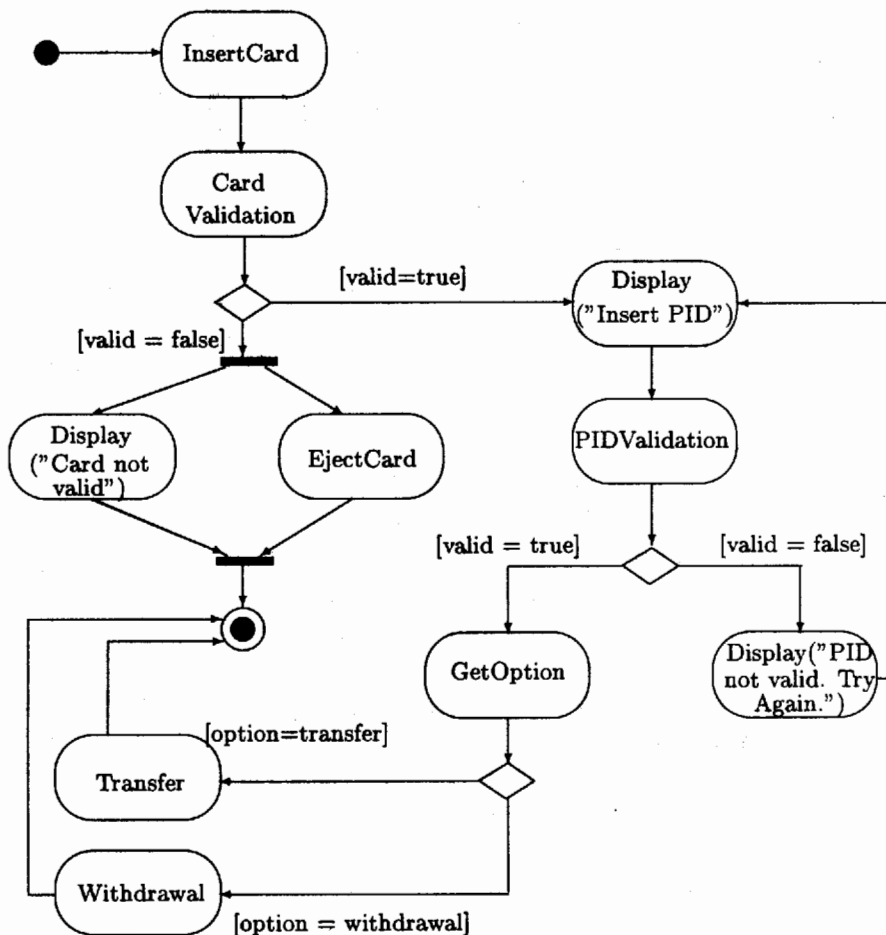


Figure 4. Activity diagram for the ATM machine

machine (ATM) is used to illustrate the procedure formally introduced in the previous section. Activities of the ATM machine are presented by activity states in the activity diagram shown in Figure 4. The initial activity *InsertCard* models situation when the user inserts bankcard, and after that it is checked whether the card is valid. If the card is not valid a message is sent to the user, the card is ejected and the session is ended. If the card is valid, the user is asked to insert *PID*, which is validated.

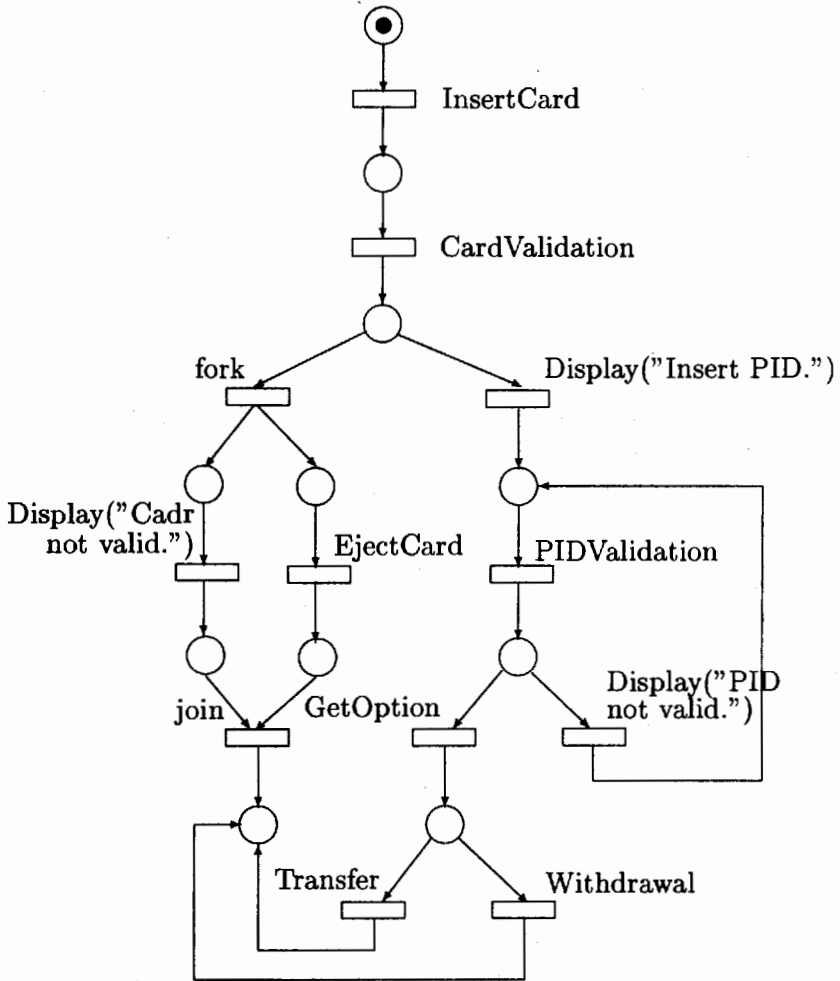


Figure 5: Petri net model of the activity diagram shown in Figure 4

If the *PID* is not valid, an appropriate message is sent to the user. The user is asked to insert *PID* again. If the *PID* is valid the user is asked to select an option. After this an appropriate service is called. When the service is completed the section is ended.

Figure 5 shows the Petri net model of the activity diagram presented in Figure 4. The transformation of the activity diagram into the Petri net model is based on the transformation rules shown in Figures 1-3. For example, the initial pseudostate of the UML is transformed into a place of the Petri nets with one token in the initial marking. The activity state InsertCard is modeled as transition InsertCard in the Petri net model. And, it is similar is for other activity states and decision points. The Petri net model gives a precise description of the flow of control and shows concurrent activities, synchronization of the activities and conflict situations.

## 5. Conclusion

Petri nets are proposed as an auxiliary tool to model activities of a system under the development. Activities and their order are described in an activity diagram of the UML (Unified Modeling Language). An activity diagram shows the interaction between objects, but in terms of activities. Activities are represented as action states and the transitions between states are implicitly triggered by completion of the actions in the source states.

A formal procedure for transformation of the activity diagram of the UML into a Petri net model is given. On the basis of this transformation it is possible to accomplish verification of the dynamic model of the real system, i.e. to evaluate whether the activities and their order are well defined. It is also possible to solve the problem of concurrency and synchronization of the activities in the system, as well as to optimize the dynamic model.

## References

- [1] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenzen, W., Object-Oriented Modeling and Design, Prentice Hall, 1991.
- [2] Booch, G., Object-Oriented Analysis and Design with Applications, 2nd edition, Benjamin/Cummings, 1994.

- [3] Jacobson, I., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1993.
- [4] Booch, G., Rumbaugh, J., Jacobson, I., *Unified Modeling Language, Version 1.1*, Rational Software Corporation, September 1997, <http://www.rational.com>
- [5] Tričković, I., *Application of Petri Nets in the Specification of the Information Systems Dynamic*, master's thesis, Institute of Mathematics, Novi Sad 1997. (in Serbian)
- [6] Tričković, I., *The transformation of the state diagram of the UML into the Petri Net model*, Novi Sad Journal of Mathematics, Vol. 28, Number 3, 1998, Department of Mathematics, University of Novi Sad, 1998, 187-195.
- [7] Tričković, I., Surla, D., *An application of Petri nets to the formal specification of the system dynamics*, Proceedings of the IASTED Conference on Software Engineering, Las Vegas, 1998., ACTA Press, pp. 250-253.
- [8] Maier, C., Moldt, D., *Object Coloured Petri Nets: A Formal Technique for Object-Oriented Modeling*, Proceedings on Petri Nets in System Engineering - Modeling, Verification, and Validation, Hamburg, 1997, 11-19.
- [9] Bourdeau, R.H., Cheng, B.H.C., *A Formal Semantics for Object Model Diagrams*, IEEE Transactions on Software Engineering, Vol. 21, No. 10, (1995), 799-821.
- [10] Peterson, J.L., *Petri Net Theory and the Modeling of the Systems*, Englewood Cliffs, New York, Prentice-Hall, 1981.
- [11] Murata, T., *Petri Nets: Properties, Analysis and Applications*, Proc. IEEE, vol.77, no.44, pp.541-580, Apr.1989.
- [12] David, R., Alla, H., *Petri Nets for Modeling of Dynamic Systems-a Survey*, Automatica, vol.30, no.2, pp.175-202, Feb.1994.
- [13] Baccelli, F.L., Cohen, G., Olsder, G.J., *Quadrat, Synchronization and Linearity - An Algebra for Discrete Event Systems*, John Wiley & Sons, 1996.