NOVI SAD J. MATH.
VOL. 28, NO. 3, 1998, 187-195

TRANSFORMATION OF THE STATE DIAGRAM OF THE UNIFIED MODELING LANGUAGE INTO A PETRI NETS MODEL

Ivana Tričković
Institute of Mathematics, University of Novi Sad
Trg Dositeja 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 state diagram of the Unified Modeling Language into a Petri nets model. On the basis of this transformation a verification of the dynamic model of the real system can be done. This means the evaluation of the completeness of the dynamic model and its consistency with the system requirements. It is also possible to solve the problem of concurrency and synchronization of the process in the system, as well as to optimize the dynamic model.

Key words: information systems, object-oriented methods, Unified Modeling Language, dynamic model, formal methods, Petri nets

1. Introduction

The process of 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 objects and their

relationships present 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 above the objects that can change the state of the objects.

There are different approaches to the object-oriented development of information systems [1-3], 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 state 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. The lack of formality in these diagrams prevents the evaluation of completeness, consistency, and content in the requirements and design specification. 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 static and dynamic characteristics using formal methods. Formal specification of the system dynamics is examined in this paper. The possible application of the theory of Petri nets for specification of the system dynamic is described.

The paper is organized in two parts. The first part gives an informal introduction to the theory of Petri nets. In the second, a formal procedure is proposed for transforming the state diagram of the Unified Modeling Language into a Petri nets model.

2. Petri nets and system modeling

Petri nets are a mathematical tool used for describing the system dynamics [6-9]. 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.

Definition: A Petri net structure, C, is a four-tuple, $C=(P, T, I, O)$. $P=\{p_1, p_2, ..., p_n\}$ is a finite set of *places*, $n\geq 0$. $T=\{t_1, t_2, ..., t_m\}$ is a finite set of *transitions*, $m\geq 0$. The set of places and the set of transitions are disjoint, $P \cap T = \varnothing$. $I:T\rightarrow P^\infty$ 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 μ is a function that assigns tokens to the places of a Petri net.

Definition: A marking μ of a Petri net $C=(P, T, I, O)$ is a function 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 μ of the Petri net to a new marking μ'. 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 concurrency. 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 Petri nets analysis. Many other problems in Petri net analysis can be defined by the

reachability problem. For example, the set of reachable markings for a given marked Petri net is defined as follows. The marking μ is in the set of reachable markings, $R(C, \mu)$, if there is any sequence of transition firings which will change the marking μ into μ'.

Definition (Reachability problem): The Reachability problem for the Petri net (P, T, I, O, μ) and marking μ' is defined as question: is $\mu' \in R(C, \mu.)$?

3. System state modeling with Petri nets

The dynamic aspects of a system can be modeled with the state diagram of the Unified Modeling Language. The basic concepts shown in the diagram are the states and relationships between the states. A state diagram is described by the metamodel. A segment of the metamodel is shown in Figure 1. The state diagram is a directed graph. Nodes of the graph represent the states of the object, and arcs, ordered pairs of nodes, represent transitions from one state to another. The state can be an action state, a composite state, or a pseudostate. An action state represents the state with no substates and only one internal transition. The responsibility of the pseudostate is to specify all non-state nodes, including initial, final, and history nodes. A history pseudostate represents a history marker, whose presence affects the dynamic semantics of the state. The responsibility of the composite state is to specify a state containing one or more substates or concurrent substates. The state is described by a set of attributes representing the state variables. Each state can be the source and/or the target of one or more transitions. But a state can be the source and target of no transition. On the other hand, each transition must have at least one source and one target, although two states may be the same.

Each transition corresponds to at most one event and can have a condition for the trigger of the transition. The transition firing rule can be described as follows. Assume that the object is in a state that allows firing a transition t. If t corresponds to an event e, the transition will be fired if and only if e happened, and the guard condition is true. If a transition t does not correspond to any event, the transition will be fired if and only if the guard condition is true.

An event can be a signal event, call event, or time event. Signal is a named event sent to one or more objects. Call event is an event triggered by an operation. Time event is an event triggered by the passing of time.

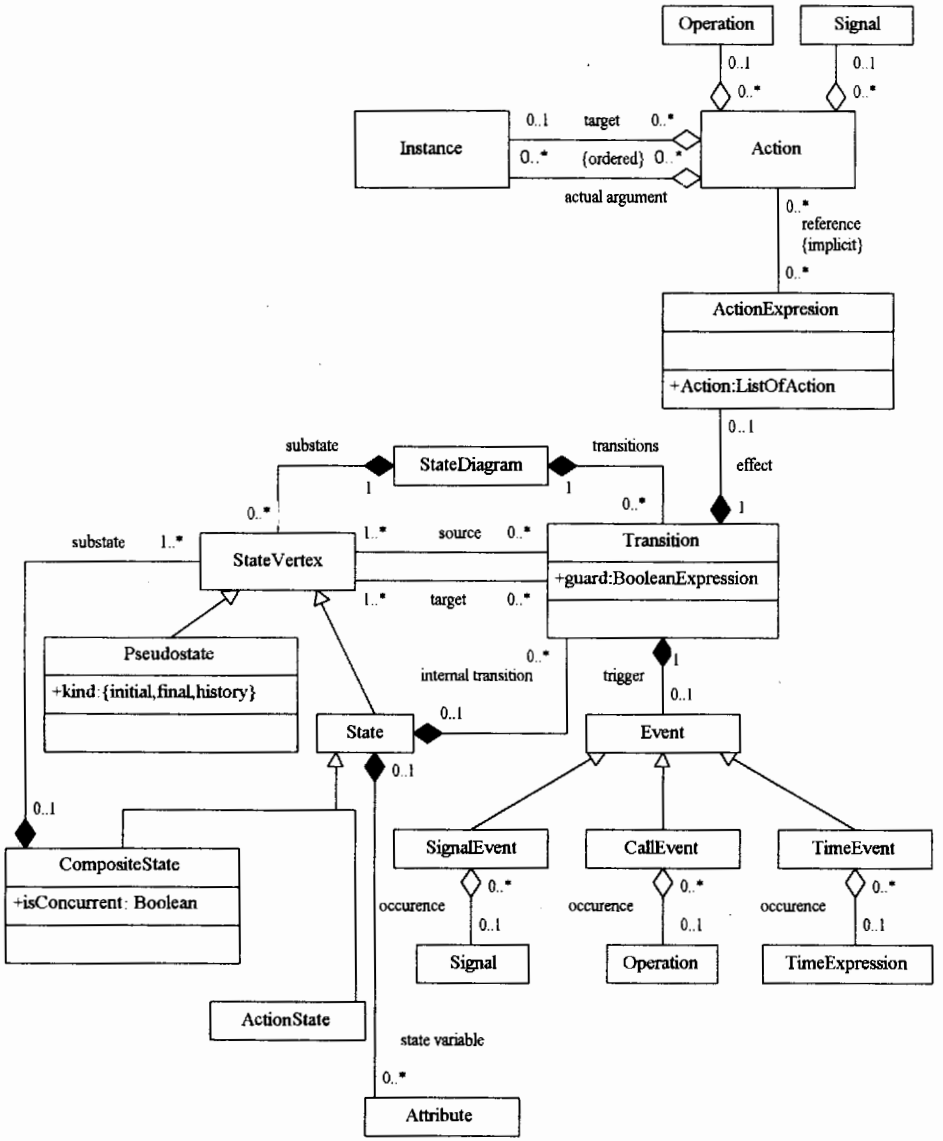

Figure 1: Segment of the state diagram

The association between states and transitions in the metamodel allows one transition to have multiple input states and multiple output states. The transition that has multiple input states and one output state is called join. It represents a

synchronization among all of the sources, each of which must be a concurrent state. The transition that has multiple output states and one input state is called fork. The target of such a transition must refer to concurrent state.

A definition of the state diagram can be formalized as follows. Let us define *input event* as an event received from some external, or some other object in the system. *Output event* is defined as an event generated while transaction is fired and sent to one or more externals, or some other objects in the system. The given definition of the state diagram is based on the mentioned metamodel.

Definition of the state diagram

State diagram is tuple $(S, s_0, Z, A, In, Out, E, F, \Sigma, \Delta, A_\Delta)$.
$S = \{s_1, s_2, \ldots, s_k\}$ is a finite set of states;
s_0 is initial pseudostate;
$Z = \{z_1, z_2, \ldots, z_l\}$ is finite set of final pseudostate;
$A = \{a_1, a_2, \ldots, a_m\}$ is a finite set of transitions;
$In: A \rightarrow S \cup \{s_0\}$,
$Out: A \rightarrow S \cup Z$ are functions that define a set of source and target states of a transition;
$E = \{e_1, e_2, \ldots, e_n\}$ is a finite set of events;
$F: A \rightarrow E \cup \{\varepsilon\}$ is a function which mapps transition to the event that triggers the transition (simbol ε represents dummy event meaning that the transition is not triggered by any event);
$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_p\}$ is a finite set of input events, $\Sigma \subseteq E$;
$\Delta = \{\delta_1, \delta_2, \ldots, \delta_q\}$ is a finite set of output events;
$A_\Delta = \{(a_{\delta_1}, \delta_1), (a_{\delta_2}, \delta_2), \ldots, (a_{\delta_q}, \delta_q)\}$, $a_{\delta_i} \in A$, $i = 1, \ldots, q$, is a finite set of ordered pairs, the first member of pair is a transition, and the second member is an output event generated when the transition is fired.

The Petri net model is based on the above definition of the state diagram.

The Petri net (P, T, I, O, μ) of the state diagram $(S, s_0, Z, A, In, Out, E, F, \Sigma, \Delta, A_\Delta)$

$P = S \cup \Sigma \cup \Delta$, $P = \{p_1, \ldots, p_u\}$, $u = k + p + q$, $k = |S|$, $p = |\Sigma|$, $q = |\Delta|$;
$T = \{t_{a_1}, \ldots, t_{a_v}\}$, $a_1, \ldots, a_v \in A_s$,
where $A_s = A \,/\, \{a_i \mid In(a_i) = s_0\} \cup \{a_i \mid Out(a_i) = z_j, j = 1, \ldots, l\}$;
$I(t_{a_i}) = \{In(a_i) \mid a_i \in A_s\} \cup \{\sigma_i \mid \sigma_i \in \Sigma, F(a_i) = \sigma_i\}$;

$O(t_{a_i}) = \{Out(a_i) \mid a_i \in A_s\} \cup \{\delta_i \mid \delta_i \in \Delta, (a_i, \delta_i) \in A_\Delta\}$;
initial marking is defined as follows:

if exists $a_i \in A$ such that $In(a_i) = s_0$ i $Out(a_i) = s_i$,

$$\mu(p_{s_i}) = 1,$$

else

$$\mu(p_{s_i}) = 0.$$

In the state diagram a transition might have multiple input states and multiple output states, but there are no multiple parallel arcs between two states. States can be interpreted as conditions that are satisfied. The object is in the state and waits for the even that triggers a transition which fires and changes the state of the object, or executes an activity and than changes the state of the object.

A Petri net model of the state diagram is an ordinary Petri net because there are no multiple arcs between two states. Inputs and outputs of the Petri net are modeled as places. It is possible to model input and output as transitions. The set of transitions T will be expanded with the transitions that correspond to the input and output places defined as follows:

$$T_u: t_{\sigma_i} \in T_u,\ I(t_{\sigma_i}) = \{\},\ O(t_{\sigma_i}) = \{\sigma_i \mid \sigma_i \in \Sigma\}$$
$$T_i: t_{\delta_j} \in T_i,\ I(t_{\delta_j}) = \{\delta_j \mid \delta_j \in \Delta\},\ O(t_{\delta_j}) = \{\}$$

This transformation of the state diagram into a Petri net model must encompass composite states. The transformation of the composite states into a Petri net model is given in [5]. The transformation of the activity diagram, the collaboration diagram, and the sequence diagram into the Petri net model has also been proposed in [5].

4. Conclusion

Petri nets are a graphical and mathematical tool well suited to model the dynamic aspects of a system. The practical application of the Petri nets can be accomplished in two ways. One approach considers Petri nets as an auxiliary analysis tool. For this approach, conventional design techniques are used to specify a system. The system specification is then modeled as a Petri net that is analyzed. Any problem encountered in the analysis point to flaws in the design. The design must be modified to correct the flaws. This modified design can be modeled and analyzed again. This cycle is repeated while the analysis reveals

unacceptable problems. This approach requires modeling techniques to transform the model of a system into a Petri net model. The other approach is more radical. The entire design and specification process is carried out in terms of Petri nets.

In this paper, we proposed the first approach. In this way, the designer can use, for example the widely accepted Unified Modeling Language, as a software development technique, for creating a model of the system. The complexity of the Petri net structure restricts the number of designers that use the second approach. We described the transformation of a state diagram of the UML into a Petri nets model.

There are different software tools and packages supporting drawing, analysis, and simulation of Petri nets. Software tools for the analysis of information system design based on the proposed transformation can be implemented. The main problem in the proposed formal transformation is presentation of the history state. Next step in the research can be to further simplify the Petri nets model, or to use some special classes of Petri nets (e.g. object-oriented Petri nets).

References

[1]Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, 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 1.0*, Rational Software Corporation, January 1997, http://www.rational.com

[5]Tričković, I., *The Application of the Petri Nets for Specification of the Dynamics of Information Systems*, M.Sc. thesis. University of Novi Sad, 1997.

[6]Peterson, J.L., *Petri Net Theory and the Modeling of the Systems*, Englewood Cliffs, New York, Prentice-Hall, 1981.

[7]Murata, T., *Petri Nets: Properties, Analysis and Applications*, Proc. IEEE, vol.77, no.44, pp.541-580, Apr.1989.

[8]David, R., Alla, H., *Petri Nets for Modeling of Dynamic Systems-a Survey*, Automatica, vol.30, no.2, pp.175-202, Feb.1994.

[9]Baccelli, F.L., Cohen, G., Olsder, G.J., Quadrat, J.-P., *Synchronization and Linearity: An Algebra for Discrete Event Systems*, John Wiley & Sons, 1996.