MODELLING INTELIGENT SYSTEMS WITH LEVEL PETRI NETS

Corina BOCĂNEALĂ

Department of Mathematics-Informatics,
University "Dunărea de Jos", Galați
Domnească 111, 800008, Galați
Email: cbocaneala@ugal.ro

Abstract: Level Petri Nets are formalism for modeling hierarchical multi-agent system. They are a Petri nets extension, allowing tokens to be nets themselves. This paper is inspired by two classes of level Petri nets: object Petri nets and nested Petri nets. We present some concepts from the artificial intelligence field and we use them to illustrate the modeling power of Petri nets with multi levels.

Keywords: Petri nets with multi levels, object Petri nets, artificial intelligence, daemons

1. INTRODUCTION

Petri nets are a very useful tool for modeling distributed and concurrent systems.

Many researchers extended this formalism using notions from object oriented programming. R. Valk (1998) introduced object Petri nets consisting in a system net with net tokens (object nets). An object marking of a place in the object system can be an object net with its marking or a natural number representing the number of black tokens. Object markings are adequate only for reference semantics (the case of Petri nets where the references to the dynamic token nets reside in the place, not the token nets themselves). To give the value semantics for the object Petri nets formalism, the notion of process markings for the object system was introduced. In this case the tokens are viewed as the finite process of the token net. Object systems can synchronize with the system net ore with another object system. The synchronization between object systems is not restricted to the case where they occupy the same place. The occurrence rule for object systems allows the distributed parallel execution of an object net in the presence of a strict fork-join structure (Farwer, 2001). Other properties of this class of nets are presented in (Kökler, 2003; Köhler and Rölke, 2004; Köhler and Rölke 2005).

Another model of nets within nets concept is represented by nested Petri nets introduced by I. A. Lomazova (Lomazova 1999; Lomazova 2000; Lomazova 2001). In a nested Petri net tokens may be nets themselves. An element net may have its own structure and behavior; it can appear and disappear during a system run. There is no limit concerning the number of the element nets. Lomazova presents the case of the two level Petri nets. The behavior of a two - level Petri nets consists in four types of steps. The transport step is a step in the system net that can generate, move or remove elements, but it can not change the inner state of the elements of the system net. An element autonomous step changes the inner state of some elements of the system net. The vertical synchronization step means simultaneous firings of two transitions: one from the system net and the other from an element net involved in the firing in the system net. Vertical synchronization step refers to simultaneous firings of two transitions from two element nets in the same place of the system net. Three level Petri nets are presented in (Jucan and Captarencu, 2002). We can generalize these steps to k-level Petri nets.

The first section presents some concepts regarding Petri nets with multi levels inspired by the Lomazova's nested Petri nets.

This paper was recommended for publication by Severin Bumbaru

In the second section we present some notions from the artificial intelligence field. A daemon is a procedure created for the purpose of handling periodic service requests that a computer system expects to receive. A program might include any number of daemons. A daemon has three states: asleep, awake and active.

The third part of this paper shows how we can use level Petri nets for modeling intelligent systems. Using level Petri nets processes can be manipulated by others. Level Petri nets are used for modeling issues by many researchers (Van Hee, *et al,*. 2006; Captarencu and Jucan, 2003; Bashkin and Lomazova, 2003a; Bashkin and Lomazova, 2003b; Lomazova, 2002).

2. TWO LEVEL PETRI NETS: DEFINITIONS

In this section we give a general definition of two level Petri Nets. These notions were presented by I.A. Lomazova (1999, 2000, and 2001). We consider that reader is familiar with the notions of Petri net theory (Jucan and Ţiplea, 1998).

Let $Var = \{v_1, ...\}$ a set of variable and $Con = \{c_1, ...\}$ a set of constants. We'll interpret the elements of *Con* as element nets with their markings. We consider $Atom = Var \cup Con$.

Definition 2.1. We define *Expr(Atom)* the expression language with the operations *"(,...,)$_n$"* and *"+"*. We have:

- An $atom \in Atom$ is an expression from *Expr(Atom)* with dimensionality *1*.
- If $atom_1, atom_2, ..., atom_n \in Atom$, then the tuple $(atom_1, atom_2, ..., atom_n)$ is an expression in *Expr(Atom)* with dimensionality *n*.
- IF $e_1, e_2 \in Expr(Atom)$ are expression with the same dimensionality *n*, then $(e_1 + e_2)$ is an expression in *Expr(Atom)* with the dimensionality *n*.

Constants in the expression in *Expr(Atom)* will be interpreted as ordinary Petri nets or as individual token without inner structure (atomic tokens). *Var(e)* means the set of variable occurring in $e \in Expr(Atom)$.

A_{net} is the set of net tokens and A_{atom} is the set of atomic tokens.

We consider $Lab_v = \{l_1, l_2, ...\}$ and $Lab_h = \{\lambda_1, \lambda_2, ...\}$ two disjoint set of labels. Labels from Lab_v are used for vertical synchronization and labels form Lab_h are used for horizontal synchronization.

We also define the adjacent labels $\bar{l} \in Lab_v$ and $\overline{\lambda} \in Lab_h$. $l_1, l_2 \in Lab_v$, $l_1 \neq l_2$ implies $\overline{l_1} \neq \overline{l_2}$. $\lambda_1, \lambda_2 \in Lab_v$, $\lambda_1 \neq \lambda_2$ implies $\overline{\lambda_1} \neq \overline{\lambda_2}$. $\bar{\bar{l}} =_{def} l$, $\overline{\overline{\lambda}} =_{def} \lambda$.

Definition 2.2. A two level Petri net is a tuple:

$LPN = (Atom, Lab, (PN_1, m_0^1), ..., (PN_k, m_0^k), SN, \Lambda)$, where:

- $Atom = Var \cup Con$ is a set of atoms;
- $Lab = Lab_v \cup Lab_h$ is the set of labels defined above;
- $(PN_1, m_0^1), ..., (PN_2, m_0^k)$, $k \geq 1$ are a finite number of ordinary Petri nets together with their initial markings;
- $SN = (N, L, U, W, M_0)$ is a high level Petri Net, called system net for *LPN* , where:
  - $N = (P, T, F)$ is a Petri net;
  - $L = Expr(Atom)$;
  - $U = (A, I)$, $A = A_{net} \cup A_{atom}$, $I : Con \rightarrow A$ is an interpretation function which gives the interpretation to constant names;
  - *W* is a function which maps an arc *(x, y)* to an expression *W(x, y)* with dimension *n*, where *n* is the arity of the place incident to arc *(x, y)*;

    For each transition its arc expressions must satisfy the following restrictions: there are not net constants (from A_{net}) in input arc expressions; every variable has at most one occurrence in each input arc expression; for every two expressions $W(p_1, t)$ and $W(p_2, t)$ ascribed to two input arcs for the same transition *t*, it is necessary that $Var(W(p_1, t)) \cap Var(W(p_2, t)) = \varnothing$.

    If for an arc an expression is not present we suppose it is $1 \in \mathbf{N}$.
  - M_0 is the initial marking of the net;
- Λ is a partial function of transition labeling.

Let us consider a transition *t* in *SN*. We denote by $^\bullet t = \{p_1, p_2, ..., p_i\}$ the set of its preelements and by $t^\bullet = \{q_1, q_2, ..., q_j\}$ the set of its post elements.

A binding of t is a function b which ascribes to each variable v occurring in some expression in $W(t)$ a value $b(v)$ from A.

Definition 2.3. A transition t in SN is enabled in a marking M w.r.t. a binding b if and only if

$$\forall p \in {}^{\bullet}t: W(p, t)(b) \subseteq M(p).$$

The enabled transition fires and results a new marking M′. We write $M \xrightarrow{t[b]} M'$.

For all places p,

$$M'(p) = (M(p) \setminus W(p, t)(b)) \cup W(t, p)(b).$$

If a net token appears as a variable in an input arc expression from $W(t)$, we say that it is involved in firing of t. It will be removed from input place, but it may be put to output places of t.

Definition 2.4. There are four kinds of steps in a two level Petri net LPN :

The transport step: Let t be an unlabeled transition ($\Lambda(t)$ is not defined) in SN. If t is enabled in a marking M w.r.t a binding b and $M \xrightarrow{t[b]} M'$, then this firing of t in the system net is called a transport step in LPN and we write $M[t[b]\rangle M'$ or just $M[\rangle M'$.

A transport step doesn't change the inner markings of net tokens, but it can remove or transfer some of net tokens. New net can evolve as a result of a transport step.

The element-autonomous step: Let M be a marking in LPN , $p \in P$ a place in SN and $\overline{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_n) \in M(p)$ a tuple of tokens in p. Let $\alpha_i = (PN, m)$ be a net token in this tuple with a transition t enabled in a marking m. We consider that $\Lambda(t)$ is not defined and $m \xrightarrow{t} m'$. Let M' be the marking in LPN obtained substituting a net token $\alpha_i = (PN, m)$ with $\alpha_i' = (PN, m')$ in M. So M' is the marking obtained form m as a result of local firing of t in a net token α_i, while PN remains in the same place of SN.

This firing is called an element-autonomous step in LPN . We write $M \xrightarrow{t} M'$ or just $M[\rangle M'$.

The horizontal synchronization step: Let M be a marking in LPN , $p \in P$ a place in SN and $\overline{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_n) \in M(p)$ a tuple of tokens in p. Let $\alpha_i = (PN_1, m_1)$ and $\alpha_j = (PN_2, m_2)$ be a two net tokens in this tuple and t_1 an enabled transition in PN_1, with $\Lambda(t_1) = \lambda$, $\lambda \in Lab_h$ such that $m_1 \xrightarrow{t_1} m_1'$ (using the rules for ordinary Petri nets), t_2 an enabled transition in PN_2, with $\Lambda(t_2) = \overline{\lambda}$, $\overline{\lambda} \in Lab_h$ such that $m_2 \xrightarrow{t_2} m_2'$ (using the rules for ordinary Petri nets). We consider M' the marking in LPN obtained substituting a net token $\alpha_i = (PN_1, m_1)$ with $\alpha_i' = (PN_1, m_1')$ in M and $\alpha_j = (PN_2, m_2)$ in M with $\alpha_j' = (PN_2, m_2')$. M' is the marking obtained from M by simultaneous firings of t_1 in PN_1 and t_2 in PN_2 while both nets PN_1 and PN_2 remain on their position in the same tuple in a place in SN.

This firing of t_1 and t_2 is called a horizontal synchronization step in LPN . We write $M[t_1, t_2\rangle M'$ or just $M[\rangle M'$.

The vertical synchronization step: Let M be a marking in LPN and t a transition enabled in M w.r.t. a binding b and $M \xrightarrow{t[b]} M'$, $\Lambda(t) = l$, $l \in Lab_v$. Let $\alpha_1, \alpha_2, ..., \alpha_k \in A_{net}$ be the net tokens involved in the firing of t, where $\alpha_1 = (PN_1, m_1)$, $\alpha_2 = (PN_2, m_2)$, ..., $\alpha_k = (PN_k, m_k)$. We suppose that for each $i = \overline{1,k}$ there is a transition $t_i \in PN_i$, such that t_i is enabled in a marking m_i, $m_i \xrightarrow{t_i} m_i'$ (using the rules of ordinary Petri nets) and $\Lambda(t_i) = \overline{l}$, $\overline{l} \in Lab_v$. We consider $W'(t, p)(b)$ the multiset of token tuples, obtained from $W(t, p)(b)$ by replacing a net token $\alpha_i = (PN_i, m_i)$ by token $\alpha_i' = (PN_i, m_i')$, for all $i = \overline{1,k}$.

Synchronous firing of a transition t in a System net SN together with the transitions involved in this firing results a marking

$$M' = (M(p) \setminus W(p, t)(b) \cup W'(t, p)(b).$$

Such a synchronous firing of a transition t w.r.t. a binding b in a system net and transitions $t_1, t_2, ..., t_k$ in involved net tokens $\alpha_1, \alpha_2, ..., \alpha_k$ is called a vertical synchronization step.

We write $M[t[b]; t_1, ..., t_k\rangle M'$ or just $M[\rangle M'$.

We say that a marking M' is directly reachable from a marking M and we write $M[\rangle M'$ if there is a step in LPN leading from M to M'.

A run of a level Petri net LPN is a sequence of markings $M_0[\rangle M_1[\rangle ...$ successively reachable from the initial marking M_0.

A marking M in LPN is called reachable if there exists a run $M_0[\rangle M_1[\rangle ...[\rangle M_k$ with $M = M_k$.

If a level Petri net can generate its copy directly or as a grandchild in a run process, then it is called recursive level Petri net.

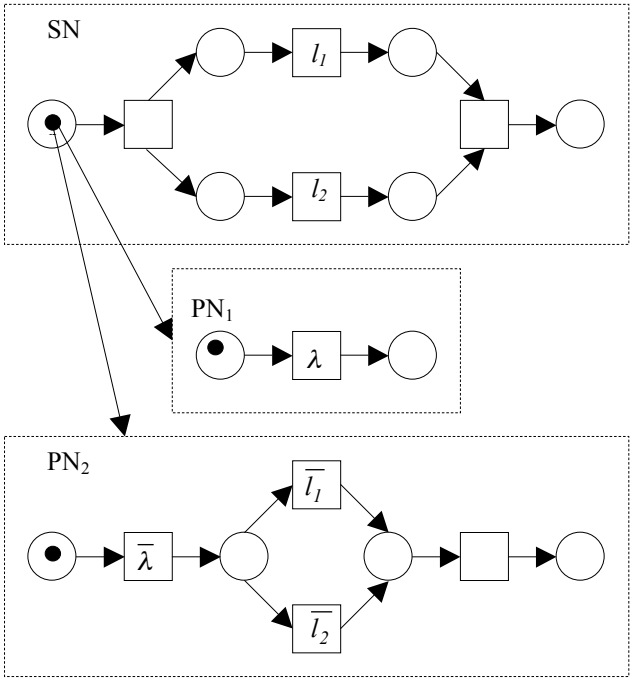

Fig.1. A two level Petri net

In a three level Petri net (Jucan and Captarencu, 2002), tokens are two level Petri nets. The behavior of a three level Petri net is more complex. The transport step is similar to the one in the case of two-level Petri nets. An element autonomous step consists in two cases depending if the transition fires at level two or at level three. The horizontal synchronization considers the synchronization between elements on level two or between elements on level three. We can have a horizontal synchronization between all three levels, between level one and level two or between level two and level three.

For practical reasons it is useful to suppose that the element nets in a k-level Petri net can have *1*, *2*, … or *k-1* levels. If a k-level Petri net is considering, the four classes of steps are maintained, but generalized. The transport step is the same like in the two or there level net. The element autonomous step is divided in two cases: the element net is an ordinary Petri net or is a level Petri net. Horizontal synchronization is divided in horizontal synchronization of some elements (two ore more) in the system net or horizontal synchronization of some elements in the system net of an element net. Vertical synchronization means synchronization between transitions from two ore more adjacent levels. First level of vertical synchronization is also important.

We can also generalize the coverability structures for level Petri nets (Bocăneală, 2008). We construct the coverability trees for the system net and for the element net and we synchronize them using the labels for vertical or horizontal synchronization. The coverability tree for the system net of *LPN* is finite and it can be effectively constructed.

The coverability tree helps us to solve some decidability problems such as: the termination problem, the transitions activity maintainability problem, the maintainability problem.

3. DAEMONS

An expert system to act intelligent must validate and manipulate knowledge, not only store it. A method is a procedure that is executed whenever is needed. A daemon represents a procedure which is activated whenever the intelligent system satisfies some conditions. A daemon always offers its services if the system needs that. We can consider a daemon an IF-THEN structure. The two notions of method and daemon are not synonyms.

A daemon has three states: asleep, awake and active. When it is asleep the daemon doesn't pay attention to the changes in his universe. If the daemon is awake it traces the changes in his environment and decides if it is necessary to offer its services. If the daemon is active it resolves its job and after finishing it becomes awake. Only another process can decide if an awaken daemon must be asleep or if an asleep daemon must be awake. Most demons will operate without user actions.

An example of a demon may be found in personal computer help systems when a program activated by the state of the user applications, offers help, or an idea. Another example is the alarm clock from the mobile phone. Mail daemons will let us know that an email has been unsuccessful and returned. Antivirus programs are demons.

An AI program might include a number of demons. One or more daemons might become active when new information (knowledge) was acquired by the program. If the new knowledge affects demon knowledge, it would spring into action and create new piece of knowledge based on its particular inference rules. Each of these new pieces of knowledge might activate additional demons that would continue to filter through and refine the entire AI knowledge base.

4. MODELLING INTELIGENT SYSTEMS WITH LEVEL PETRI NETS

Daemons are one of the main notions of artificial intelligence and software development. A daemon is

intelligent. It must complete some actions and must interact with agents. We can say that a daemon:

- *has initiative:* its actions are determined by the state of its universe, but it decides if, when and where must intervene;
- *interacts:* it interacts with other daemons or with others processes (e.g. the processes which changes its state: awake or asleep);
- *is reactive*: if its universe attained a certain state if can perceive this information and react on it.

We can model intelligent system with object Petri nets. In this case daemons can be dynamically created and destroyed as instances of a daemon classes. A daemon class is a Petri net.

Level Petri nets can be used for modeling daemons and their behavior. The universe can be modeled as the system net. Elements in a level Petri net may have their own structure and behavior may evolve or disappear during the system run and their number is unlimited.

It is obvious that we can have any number of levels in the net witch models an intelligent system. A daemon is a procedure witch calls other daemons. Net daemons may have net tokens corresponding with some procedures.

Level Petri nets are adequate for modeling daemons features. Initiative is modeled by the element autonomous step. An element autonomous step only changes the inner state of the net daemon. Interaction is modeled by horizontal synchronization. A daemon must act in the same time with other ones or with others procedures. The transitions in the daemon net witch must fire at the same time with other from other nets must be labeled for horizontal synchronization. We can show that a daemon is reactive using vertical synchronization. Its transitions fire with others of his parent net and it must be labeled for horizontal synchronization.

For modeling daemon learning this formalism must be extended with operations that can change arc inscriptions or even net structure. These operations should maintain some properties such as decidability results. This is a subject for further research.

5. CONCLUSIONS

Level Petri nets are a visual and convenient tool for modeling distributed and intelligent systems. They maintain some important properties of classical Petri net model, but they give a dynamic representation of the hierarchical and modular structure of a system. There are many kind of synchronization between elements of Petri nets with multi levels witch increase their expressivity.

In this paper we discuss some concrete aspects of the applicability of level Petri nets. Our intension was to combine some AI concepts with the modeling power of level Petri nets.

For the moment, there is no software tool who implements these ideas. We intend to create a program to take advantage of the expressivity of the nets within nets concept.

REFERENCES

Bashkin, V. and I.A. Lomazova (2003a). Resource Similarities in Petri Net Models of Distributed Systems, *Lecture Notes in Computer Science*, Vol. 2673, Springer-Verlag, pp. 35-48.

Bashkin, V. and I.A. Lomazova (2003b). Petri nets and resource bisimulation, *Fundamenta Informaticae*, Vol. 55, 2003, pp. 101-114.

Bocăneală, C. (2008). On Coverability Structures for Nested Petri Nets, *Proceeding of the Third International Conference on Mathematical Sciences – ICM2008*, United Arab Emirates, Volume 1, pp. 234-243.

Captarencu, O. and T. Jucan (2003). Interorganizational Workflows – an approach based on level Petri nets, *Scientific Annals of the "Alexandru Ioan Cuza" of Iaşi, Romania, Computer Science Section*, Tome XIII, pp. 17-37.

Farwer, B. (2001), Comparing Concepts of Object Petri Net Formalisms, *Fundamenta Informaticae*, Vol. 47, pp.247-258.

Jucan, T. and O. Captarencu (2002). Three Level Petri Nets, *Scientific Annals of the "Alexandru Ioan Cuza" of Iaşi, Romania, Computer Science Section*, Tome XII, pp. 29-52.

Jucan, T. and F. Țiplea (1998). *Rețele Petri – teorie şi practică*, Romanian Academy, Bucharest.

Köhler, M. (2003). Object Petri Nets: Definitions, Properties and Related Models, *Technical Report FBI-HH-M-329/03*, Universität Hamburg, Fachbereich Informatik.

Köhler, M. and H. Rölke (2004). Properties of Object Petri Nets, *Lecture Notes in Computer Science*, Vol. 3099, Springer-Verlag, pp. 278-297.

Köhler, M. and H. Rölke (2005). Reference and Value Semantics Are Equivalent for Ordinary Object Petri Nets, *Lecture Notes in Computer Science*, Vol. 3536, Springer-Verlag, pp. 309-328.

Lomazova, I.A. and P. Schnoebelen (1999). Some Decidability Results for Nested Petri Nets, *Lecture Notes in Computer Science*, Vol. 1755, Springer-Verlag, pp. 208-220.

Lomazova, I.A. (2000). Nested Petri nets – a formalism for Specification and Verification of Multi-Agent Distributed Systems, *Fundamenta Informaticae*, Vol. 43, pp.195-214.

Lomazova, I.A. (2001). Nested Petri nets: Multi-level and Recursive Systems, *Fundamenta Informaticae*, Vol. 47, pp. 283-293.

Lomazova, I.A. (2002). Modeling Dynamic Objects in distributes Systems with Nested Petri Nets, *Fundamenta Informaticae*, Vol. 51, pp. 121-133.

Valk, R (1998). Petri nets as Token Objects: An Introduction to Elementary Object Nets, Lecture *Notes in Computer Science*, Vol. 1420, Springer-Verlag, pp1-25.

Van Hee, K., I.A. Lomazova, O. Oanea, A. Serebrenik, N. Sidorova and M. Voorhoeve (2006). Nested Nets for Adaptive Systems, *Lecture Notes in Computer Science*, Vol. 4024, Springer-Verlag, pp. 241-260.

http://whatis.techtarget.com/definition/0,,sid9_gci211932,00.html.