Journal of Advances in Computer Research
Quarterly ISSN: 2008-6148
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 3, No. 2, May 2012), Pages: 75-84
www.jacr.iausari.ac.ir

A Formal Petri Net Based Model for Antivirus Update Agent System

Ali Poyan[✉] **, Zeynab Bahrami**

School of Computer and IT Engineering, Shahrood University of Technology, Shahrood, Iran

apouyan@shahroodut.ac.ir; zbahrami@gmail.com

Received: 2012/02/02; Accepted: 2012/05/20

Abstract

In this paper, a formal model for antivirus update agent system is presented based on mobile agent technology and predicate/transition Petri nets. The mobile agent system contains two mobile agents called DCA and UNA. It sends out agents to update antivirus on client computers in a network. Each agent takes on a specified responsibility. First, DCA roams through the network and check the last date of updating of antivirus on client computers. Then, by passing the list of unupdated client computers to UNA, next migration is started. The mobile agent system is modeled with logical agent mobility method (LAM) using Petri nets. Each agent is modeled with a predicate/transition Petri net. In this model, the antivirus updating system consists of a set of components to identify different locations and a set of connectors to specify the interactions among the components. Connectors and components are modeled with PrT Nets.

Keywords: *mobile agents, antivirus update agent system, logical agent mobility*

1. Introduction

Agent paradigm is considered as a promising option for system and service architecture of the next generation networks. It is because agents are conceptually and intrinsically communication and cooperation oriented [1]. Mobile agents are software abstractions of executing programs [2] that can migrate from machine to machine across a heterogeneous network representing users in various tasks [3]. The use of mobile agents can bring several advantages such as reducing network traffic, improving locality of reference, asynchronous and decentralized execution, load balancing, dynamic adaptation, allowing the user to disconnect from the network when agents are performing a task and flexible maintenance. There are plenty of applications that benefit from mobile agent paradigm such as E-commerce, distributed information retrieval, workflow management, monitoring, and automated software installation.

The major advantages of mobile agents are flexibility and high performance, which are mainly achieved by dynamic transformation from remote call (access) into local performing (accessing code and data). Generally speaking, the strengths of mobile agents can be summarized in: latency reduction and bandwidth conservation, dynamic load balancing, support for disconnected operation and support for dynamic deployment in mobile computing environments [4][5].

The rest of this paper is organized as follows: Section 2 describes the antivirus update agent system. Section 3 gives a brief introduction to Predicate/ Transition Petri nets. Section 4 describes the logical agent mobility method. Section 5 formally models the antivirus update agent system with logical agent mobility method. In section 6, we conclude this work and discuss the future work.

2. Antivirus Update Agent System

Antivirus update agent system (AUAS), which is proposed in this paper, is a kind of automated software installation that benefits from the application of mobile agents. Mobile agents have several advantages in automated software installation. By migrating to a client computer, an agent can invoke resource operations locally, eliminating the network transfer of intermediate data. By migrating to various client computers within a network, an agent can continue executing even if the network link goes down. The main tasks for mobile agents in AUAS are to update antivirus and detect new and unknown viruses on client computers in a networked environment. Basically, the act of updating antivirus contains two stages: update signature database and update executable code. Sending out only updated files from previously installed versions, instead of complete update files, will maintain network traffic and avoid congestion in the network routes. But updating executable code typically replaces complete modules of an antivirus scanner. Therefore the antivirus engine needs to have the functionality to register, remove, update and add modules of its own. However, Mobile agent system contains two mobile agents: Date-Check-Agent (DCA) and Update-Newsig-Agent (UNA). DCA and UNA reside in the server, as shown in Figure 1.

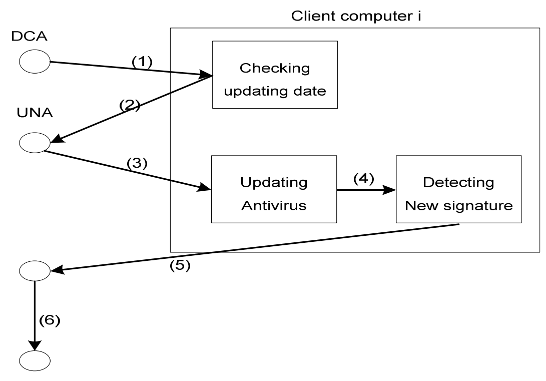

Figure 1. Tasks of mobile agents

Task of DCA is to roam through the network and check the last date of updating of antivirus (the most recent) on client computer. The UNA is responsible for updating signature database and executable code of antivirus engine. It also detects new or unknown viruses that their signatures don't exist in the signature database of antivirus, yet and send them to server for reporting to antivirus provider company. First DCA visits every client computer, and check the last date of updating of antivirus. If the antivirus has not been updated, it inserts the client name as a record into its Queue (1). Next, it migrates to another client computer within the computer network. Finally, if all client computers have been visited and checked, DCA will return to server with results as a list of unupdated client computers (2). Server uses these results to specify the traveling itinerary for UNA mobile agent. Then, server assigns mobile agent UNA to travel across specified client computers in order to update (3) and detect new and unknown viruses, respectively (4). After all specified client computers update and detect for new viruses, UNA returns to server with results as new signatures (5). Finally, server will report the new signatures that UNA has returned, to antivirus provider in order to examine the signatures and find the opposition algorithm, if needed. To present our proposed model of the mobile agent system we use Logical Agent Mobility (LAM) method based on a class of Petri nets called predicate transition (PrT) nets. In the rest of this paper, we first give a brief introduction to PrT nets and LAM method for modeling agent mobility. Then, we demonstrate the scenario of updating antivirus within a network using LAM model.

3. Predicate/Transition Petri Nets

Here for quick reference we give a brief definition of predicate/transition Petri nets, which will be referred to as PrT nets in this paper. PrT nets are a subclass of high level Petri nets [6]. The reader is referred to [7][8] for a more detailed version of PrT nets. A Predicate Transition Petri net is a seven tuple $\mathrm{Pr}\,T=(P,T,F,\Sigma,L,\varphi,M_0)$, Where

$P=\{p_1,p_2,...,p_m\}$ is a finite set of predicates (first order places),

$T=\{t_1,t_2,...,t_n\}$ is a finite set of transitions $(P\cap T=\phi, P\cup T\neq\phi)$,

$F\subseteq(P\times T)\cup(T\times P)$ is a set of arcs. (P,T,F) forms a directed net. No connections between two predicates or two transitions. Σ is a structure consisting of some sorts of individuals (constants) together with some operations and relations.

L is a labeling function on arcs. is a mapping from a set of inscription formulae to transitions. The inscription on transition $t\in T$, $\varphi(t)$, is a logical formula built from variables and the individuals, operations, and relations in structure Σ; Variables occurring free in a formula have to occur at an adjacent input arc of the transition. $M_0:P\rightarrow\{0,1,2,...\}$ is the initial or current marking, which defines number of tokens per predicate. Alternatively, the initial marking can be defined as $M_0=\bigcup_{p\in P}M_0(p)$.

A marking is an arrangement of tokens (elements subject to change) in predicate, representing state. An initial marking represent the initial state. The marking of a predicate under a certain state is a set of instead of a formal sum of tokens. These nets are more or less like first-order logic programs. From the perspective of formal verification, the reachability analysis of these PrT nets is made more efficient. It should be emphasized that transitions are the active components and predicates and tokens are passive. A transition is enabled if each of the input predicate contains tokens. An enabled transition may fire (an event happens) removing one token from each input predicate and depositing one token in each of its output predicates. And a firing sequence is a sequences <t_0, t_1, …, t_n> such that it can enabled and fired in m_0, t_1 is enabled and fires in m_1, etc.

Each agent is modeled with a PrT net, called agent net. The interface, the behavior, and the state of an agent are modeled by some input/output predicates for incoming/outgoing messages, the transitions, and the predicates of the agent net, respectively. Particularly, a concrete state of the agent is a marking of the agent net. More precisely, an agent net is a PrT net, to which an input predicate is associated for incoming messages and an output predicate is associated for outgoing messages. For a more detailed discussion the reader is referred to [9]. We use AN for agent nets and $AN.x$ to denote the element $x \in AN$ for a specific agent, throughout this paper. Tokens in Petri nets are passive, whereas agents are active. To bridge the gap between tokens and agents, a two layer approach has been proposed. Thus, we will allow an agent net to be packed up as part of a token in another PrT net (a system net in next section).

4. Logical Agent Mobility Model

A logical agent mobility (LAM) model specifies a mobile agent system as a set of components (COMP) and a set of (external) connectors (CONN). In deed, a LAM model addresses the behavior of the mobile agents and their interactions in a multi agent system. In this section we give a brief description about LAM. For a rigorous discussion about logical agent modeling please refer to [9] and [10], and for a discussion about agent-oriented software modeling the reader is referred to [11]. Different component identifies different locations for mobile agents. The connectors specify the interactions among the components. A component is a structure, which is made up of an environmental part and an internal connector, both represented with PrT nets. A connector is defined as a PrT net structure to model the connections among certain components. An agent can migrate from one component to another by transition firing at runtime because the whole agent net is used as part of a structured token in the PrT nets modeling components and connectors. Therefore, the migration results in the change of agent location. When an agent is being transferred, no transition in the agent net is enabled. The internal connector of a component is responsible for the dynamic connection of the environmental part with a changing number of mobile agents.

Definition 1. A LAM model $\Lambda = (COMP, CONN)$ is specified by a finite set of components $COMP = \{(CM_1, SN_1, ICN_1), (CM_2, SN_2, ICN_2), ...\}$ and a finite set of connectors $CONN = \{CN_1, CN_2, ...\}$, where CM_i is the location of component i, SN_i is the system net and ICN_i is the internal connector net.

The system net will be defined later in this section. Agents are distributed in components by means of packing agents up as parts of tokens in system nets and connector nets. The environmental part of a component provides facilities for agent mobility (e.g., execution place, activation, and deactivation) and an internal interface for dynamic connection with a group of agents through the internal connector. We model the environmental part of a component with a system net, as specified in the following.

Definition 2. A system net $SN=(P, T, F, \Sigma, L, \quad, p_{ex\text{-}in}, p_{ex\text{-}out}, p_{in\text{-}in}, p_{in\text{-}out}, M_0)$ for component CM is specified as a PrT net $(P, T, F, \Sigma, L, \quad, M_0)$, an external input predicate $p_{ex\text{-}in}$ and an external output predicate $p_{ex\text{-}out}$, an internal input predicate $p_{in\text{-}in}$, and an internal output predicate $p_{in\text{-}out}$.

The structure of a system net belongs to certain components with their associated connector nets can be composed into a PrT structure. Agent mobility (transfer) can be simulated in system nets by transition firings. This is because these net structures and their states can be packed up as parts of token in PrT nets. In a system net structure, if a transition is enabled (activated) an agent (modeled as part of a token), migrates from an input predicate to an output predicate. A sequence of transition firings will move the agent from one component to the other through the component net structure [9].

5. Formal Model for AUAS

In this section we describe the formal model for antivirus update agent system based on LAM model and predicate/transition Petri nets. An antivirus is installed on every client computer in the network, and every day, server assigns the mobile agents to update the Antivirus on the client computers and returns with the results. Specifically, there are three machines, namely, CM_1, CM_2, and CM_3, where CM_1 is the server and CM_2 and CM_3 are the client computers, which their antivirus software should be updated. We use LAM model to formally specify this scenario. The LAM model for this scenario can be defined as (COMP, CONN), where COMP ={(CM_1, SN_1, ICN_1) , (CM_2, SN_2,ICN_3) , (CM_3, SN_3, ICN_3)}, CONN={CN}, and CN is the connector among CM_1, CM_2, and CM_3. CM_2 and CM_3 provide a service agent for updating antivirus and detecting unknown viruses and checking the last date of updating of antivirus on the client computer. Figure 2, shows the system net SN for this scenario.

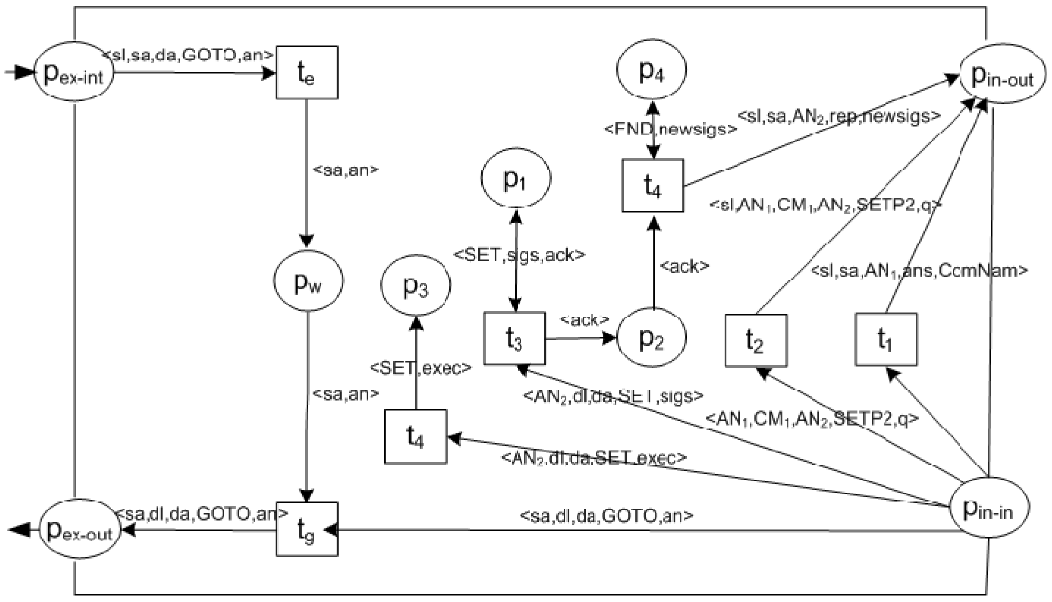

Figure 2. The system net (SN) for updating antivirus

The system nets for CM_2 and CM_3 are similar to SN but do not have transition t_2 and its associated arcs and labels. CM_2 and CM_3 provide a service agent for updating signature database and updating executable code of antivirus engine and detecting unknown viruses which are modeled by transitions t_3, t_4 and t_5, respectively. They also provide another service agent for checking the last date of updating of antivirus which is modeled by transition t_1. Because, t_4 has to start its task after updating signature database, it should wait until it receives an acknowledgement (Ack) from t_3 and then starts to detect unknown viruses. When the task of detecting virus finished, it employs structure <FND, *newsigs*> to represent replies for reporting, where *newsigs* contains new signature that t_3 has found. SN_1 is also similar to SN, but does not have t_1, t_3, t_4, t_5, p_1, p_2, p_3 and their associated arcs and arc labels. CM_1 also provides a service agent for passing list of the clients that their antivirus need to be updated, from agent DCA to agent UNA. Figure 3 shows the PrT net for connector CN, which builds the connections among CM_1, CM_2, and CM_3.

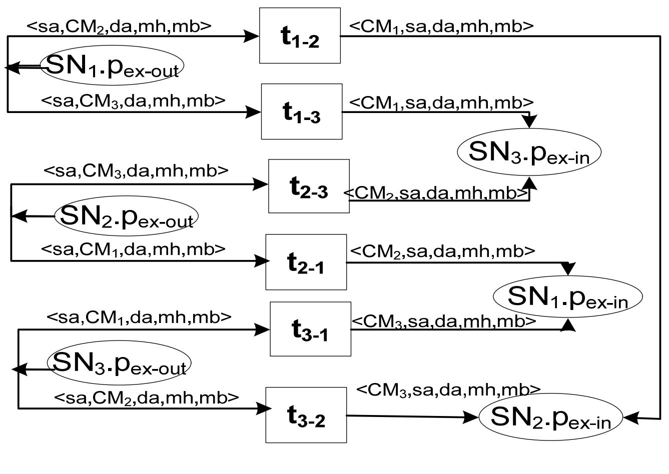

Figure 3. Prt net for connector CN

The internal connector nets ICN_i ($i = 1, 2, 3$) is shown in Figure 4.

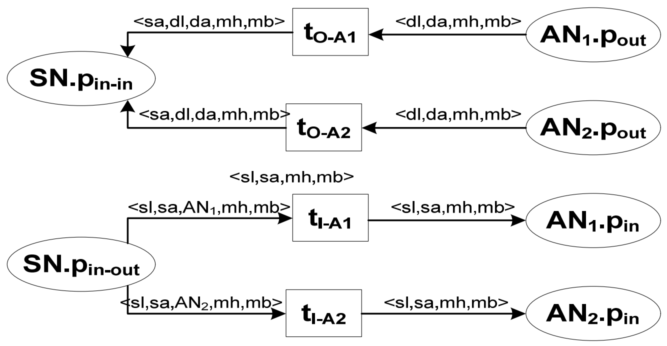

Figure 4. Prt net for internal connector (ICN)

Figure 5, shows the agent net AN for mobile agent DCA (AN_1), which may generate three types of messages as output: requests of migration (t_3), checking date (t_4) and passing unupdated client names (t_5). One of t3's predicate, p_2, specifies the traveling itinerary of agent DCA. Predicate p_3 specifies the system date.

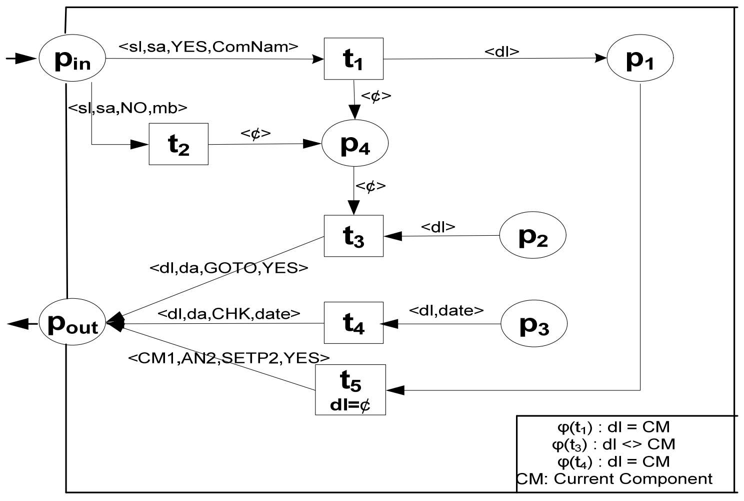

Figure 5. Agent net AN for mobile agent DCA

Transition t_1 receives an affirmative result of updating status from CM_2 (CM_3) and save the name of current component into predicate p_1. Transition t_2 handles negative answers (means current component has updated antivirus). Note that AN_1 has only one type of incoming messages: i.e., feedback if client antivirus is updated or not.

Figure 6, shows the agent net AN for mobile agent UNA (AN_2), which may output two types of messages: requests of migration (t_2) and updating Antivirus (t_3). And it has two types of incoming messages: receive new signatures (t_4) and unupdated client names (t_1). And Transition t_1 puts these names into predicate p_2, and in this manner, p_2 (like p_1 in DCA agent) specifies the traveling itinerary of agent UNA(AN_2). Predicate p_3 specifies which signatures and modules should be sent for updating. Transition t_4 receives new signatures and puts them into predicate p_1.

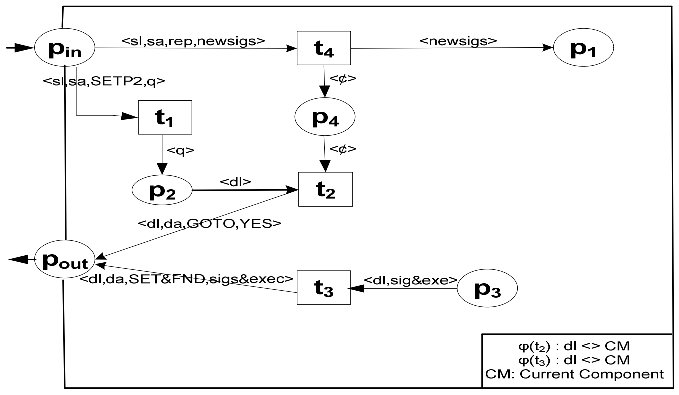

Figure 6. Agent net for mobile agent UNA

Now, we illustrate how the whole model works. Suppose in the initial state,

$AN_1.M_0(p_2) = \{< CM_2 >,< CM_3 >,< CM_1 >\}$

specifies the agent is going to visit CM_2, CM_3 and CM_1 (for return).

$AN_1.M0(p_3) = \{< CM_2, current\_date >, < CM3, current\_date >\}$

$AN_1.M0(p_4) = \{<\not{c}>\}$

$SN_1.M_0(p_w) = \{< AN_1, AN(P, T, F, \Sigma, L, \ , M_0) >, < AN_2, AN(P, T, F, \Sigma, L, \ , M_0) >\}$

Agents AN1 (DCA) and AN2 (UNA) are originally located at CM_1. $SN_2.M_0(p_1)$ and $SN_3.M_0(p_1)$ contain virus signatures and $SN_2.M_0(p_3)$ and $SN_3.M_0(p_3)$ contain executable code of AV engine.

$AN_2.M_0(p_3)$ contains new signatures and new modules for updating Antivirus:

$AN_2.M0(p_4) = \{<\not{c}>\}$

Other predicates in AN_1, AN_2, SN_i (i =1, 2, 3), ICN_i (i =1, 2, 3), and CN do not have any token. Therefore, at the beginning, no transition at the system level is enabled. However, AN_1(DCA) is running on $CM_1(SN_1)$. The only enabled transition in AN_1 is t_3, by substitution $< dl = CM_2 >$. That is, AN_1 request to move to CM_2, token will pass through $AN_1.p_{out}$, ICN_1, $SN_1.p_{in\text{-}in}$, and reach $SN_1.p_{ex\text{-}out}$ (the token is $<AN_1, CM_2, AN_1,$ GOTO, AN > at this point). Thus, transition $CN.t_{1\text{-}2}$ in connector CN is enabled. The firing of $CN.t_{1\text{-}2}$ puts token $< CM_1, AN_1, AN_1,$GOTO; AN > into $SN_2.p_{ex\text{-}in}$. during migration, the agent is inactive. As soon as agent arrives at SN_2, $SN_2.t_e$ may activate the agent AN_1. Thus, the agent restarts. The only enabled transition is t_4 for checking the last date of updating. Through ICN_2, the message will reach $SN_2.p_{in\text{-}in}$, which enables $SN_2.t_1$ to check the date. Then, the result will go through $SN_2.p_{in\text{-}out}$, ICN_2 and, finally, reach $AN_1.p_{in}$, which enables exactly one of transitions t_1 (if reply is YES) or t_2 (if reply is NO) in AN_1.

Suppose the reply is YES (means CM_2 has unupdated antivirus) and only t_1 is enabled in AN_1, then the firing of t_1 puts a token $<\not{c}>$ in p_4, and enables t_3 for next round of migration and also save the current component name in p_1. Likewise, agent AN_1 will move to and check the last date of updating of CM_3 and, the result will reach $AN_1.p_{in}$, suppose reply is NO (means CM_3 has updated antivirus) and only t_2 is enabled in AN_1. Firing of t_2 deposits a token $<\not{c}>$ in p_4, and enables t_3 for next round of migration (going back to CM_1). Upon the arrival of token at $SN_1.p_{ex\text{-}in}$, the $SN_1.t_e$ may activate the agent AN_1. The only enabled transition in AN_1 is t_5. Firing of t_5 puts unupdated clients list in $AN_1.p_{out}$. Through ICN_1, the list will reach $SN_1.p_{in\text{-}in}$, which enables $SN_1.t_2$. Firing of $SN_1.t_2$ may activate the agent AN_2, and through ICN_1, pass the list to AN_2. The list specifies the traveling itinerary of agent AN_2 (UNA). Consequently, whereas only CM_2 has unupdated antivirus, the traveling itinerary is:

$AN_2.M_0(p_2) = \{< CM_2 >, < CM_1 >\}$

It means that AN_2 is going to visit CM_2 and CM_1 (for return). Likewise, agent AN2 move to CM2. Upon the arrival of token $< CM_1, AN_2, AN_2,$ GOTO, AN > at $SN_2.p_{ex\text{-}in}$, the $SN_2.t_e$ may activate the agent. Firing of $AN_2.t_3$ and through ICN_2, new signatures and modules will reach $SN_2.p_{in\text{-}in}$, which enables $SN_2.t_3$ and $SN_2.t_4$, which in turn update executable code of antivirus engine and signature database, respectively. After finishing the task of $SN_2.t_3$, the $SN_2.t_3$ puts token <ack> in $SN_2.p_4$, which enables $SN_2.t_4$ for detecting unknown viruses (for this task $SN_2.t_4$, use $SN_2.p_4$ that contains behavior signature database).

Consequently, the reply (new signatures) will go through $SN_2.p_{in\text{-}out}$, ICN_2 and, finally, it reaches $AN_2.p_{in}$, which enables $AN_2.t_4$ for saving new signatures in $AN_2.p_1$. It puts a token <¢> in p_4, and enables t_2 for next round of migration (go back to CM_1). The sequence of firings is as follows:

$AN_1.t_3$, $ICN_1.t_{O\text{-}A1}$, $SN_1.t_g$, $CN.t_{1\text{-}2}$, $SN_2.t_e$, $AN_1.t_4$, $ICN_2.t_{O\text{-}A1}$, $SN_2.t_1$, $ICN_2.t_{I\text{-}A1}$, $AN_1.t_1$, $AN_1.t_3$, $ICN_2.t_{O\text{-}A1}$, $SN_2.t_g$, $CN.t_{2\text{-}3}$, $SN_3.t_e$, $AN_1.t_4$, $ICN_3.t_{O\text{-}A1}$, $SN_3.t_1$, $ICN_3.t_{I\text{-}A1}$, $AN_1.t_2$, $AN_1.t_3$, $ICN_3.t_{O\text{-}A1}$, $SN_3.t_g$, $CN.t_{3\text{-}1}$, $SN_1.t_e$, $AN_1.t_5$, $ICN_1.t_{O\text{-}A1}$, $SN_1.t_2$, $ICN_1.t_{I\text{-}A2}$, $AN_2.t_1$, $AN_2.t_2$, $ICN_1.t_{O\text{-}A2}$, $SN_1.t_g$, $CN.t_{1\text{-}2}$, $SN_2.t_e$, $AN_2.t_3$, $ICN_2.t_{O\text{-}A2}$, $SN_2.t_5$, $SN_2.t_3$, $SN_2.t_4$, $ICN_2.t_{I\text{-}A2}$, $AN_2.t_4$, $AN_2.t_2$, $ICN_2.t_{O\text{-}A2}$, $SN_2.t_g$, $CN.t_{2\text{-}1}$, $SN_1.t_e$.

6. Conclusion

In this paper, we presented an antivirus update agent system based on mobile agent technology and PrT nets. We introduced two mobile agents; DCA and UNA. The tasks of agents are checking the update status of client computers, updating antivirus and finally detecting unknown signatures. These are for reporting to antivirus provider for applying different algorithm on new signatures to find the methods of opposition to new viruses. This ultimately leads to upgrade the antivirus program.

For modeling this agent system we benefit from logical agent method. A mobile agent system considered as a set of connectors and a set of components. Agent connectors and components are modeled with PrT nets which are called agent connector net and system net, respectively. The mobile agent modeling is an efficient way for analysis of automated software installing like antivirus update agent. Because resource operations can be locally implemented. This will reduce the network traffic, significantly. Moreover, mobile agents can operate asynchronously and less independent of the network. It is more stable than client-server based applications.

7. References

[1] S.N. Gujar, G.R. Bamnote, R.S. Apare, M.A. Pund and S.R. Gupta, "Mobile Agent Based Distribute System Computing in Network," *International Journal of Recent Trends in Engineering, vol. 2*, no. 4, November 2009.

[2] M.R. Genesereth and S.P. Ketchpel, "Software Agents," *Comm. ACM, vol. 37*, no. 7, pp. 48-53, 1994.

[3] A. Fuggetta, G. Picco and G. Vigna, "Understanding Code Mobility," *IEEE Trans. Software Eng., vol. 24*, no. 5, pp. 342-361, May 1998.

[4] R.S. Gray, G. Cybenko, D. Kotz and D. Rus, "Mobile Agents: Motivation and State of the Art," *Handbook of Agent Technology*, J. Bradshaw, ed., 2001.

[5] C.G. Harrison, D.M. Chess and A. Kershenbaum, "Mobile Agents: Are They a Good Idea?" *IBM Research Report*, 1995.

[6] H.J. Genrich, "Predicate/Transition Nets", In: High level Petri Nets. Theory *and Application. K. Jensen and G. Rozenberg*, eds., pp. 3-43, 1991.

[7] H.J. Genrich, "Predicate/Transition Nets", *Petri Nets: Central Models and Their Properties*, W. Reisig, and G. Rozenberg, eds., pp. 207-247, 1987.

[8] H.J. Genrich and K. Lautenbach, "System Modeling with High Level Petri Nets," *Theoretical Computer Science, vol. 13*, pp. 109-136, 1981.

[9] D. Xu, J. Yin, Y. Deng and J. Ding, “A Formal Architectural Model for Logical Agent Mobility,” *IEEE Trans. Software Eng., vol. 29*, no. 1, pp. 31-45, January 2003.

[10] M. Kohler, D. Moldt and H. Rolke, “Modeling Mobility and Mobile Agents Using Nets Within Nets”, Proc. Of *Int. Conf. on Application and Theory of Petri Nets, LNCS, vol. 2769*, pp. 121-139, June 2003.

[11] H. Xu and S.M. Shatz, “A Framework for Model Based Design of Agent-Oriented Software”, *IEEE Trans. Software Eng., vol. 29*, no. 1, pp. 15-30, Jan. 2003.