An Agent-Based Model for Virtual Tourism Using Object Petri Nets

Ali A. POUYAN, MEMBER IEEE
Faculty of Computer and IT Engineering
Shahrood University of Technology
IRAN

ALI HASSAN BEIGI
Department of computer engineering
University of Birjand
IRAN

MAHNAZ KADKHODA
Department of computer engineering
University of Birjand
IRAN

Abstract – This paper presents a Petri net based agent model for virtual tourism. This model is based on object oriented Petri nets. The object Petri net model of virtual tourism consists of three distinct layers. Petri net models of agents are proposed to specify the components of mobile agent systems as autonomous modules. Each layer of the system has been modeled by a Petri net sub-module. The upper layers of the model consist of transitions, which are the encapsulated versions of the Petri net modules in a lower level layer. Communications among agents are defined via interfaces that are specified by the agent.

Keywords: Modeling, object Petri nets, mobile agents, multi agent systems, virtual tourism.

1 Introduction

Web environment is undergoing a transformation into a platform for highly distributed applications such as web-based systems and electronic commerce. Specifically, mobile software agents are a generic network programming paradigm, where migrating software components (computer programs) carry out certain distributed tasks by roaming the heterogeneous network systems.

Mobile software agents [1], [2], or simply mobile agents, evolved from autonomous agents [3] introduced a decade ago as a powerful abstraction for conceptualizing large-scale distributed asynchronous computer network systems [4]. It supports a wide range of different types of computer applications such as electronic commerce, network management, distributed information retrieval, workflow management, real-time conferencing; wireless/cellular based mobile computing and the implementation of telecommunication services. In general, the mobile agent paradigm is considered as a solution to reducing network congestion due to heavy traffic load in the network and managing its complexity.

Mobile agents are executing programs that migrate from machine to machine in a heterogeneous network [5], [6]. They run within agent server programs as logical places referred to as agencies. When a mobile agent migrates to a specific node in the network, its execution is suspended at the original agency. The program code, control information, data and execution status are transferred to the host agency. The mobile agent resumes execution after being re-instantiated at the destination environment. Mobile agents have the ability to prevent or solve problems encountered in the network during their journey, and they have the ability to communicate with other. Mobile agent based software systems have gained wide acceptance as a conceptual framework that provides, among others, the following benefits [7]: more efficient use of communication resources by using much less bandwidth than a conventional correspondent RPC-based client; dynamic load balancing by partitioning a task into components that are distributed across multiple processors; flexible management for software deployment and maintenance; adequate support for interactions with environment and flexible support for disconnected operations.

A critical issue in the development of software systems based on mobile agents is the support of formal reasoning and analysis of designed systems [3]. For most real-world applications with a large number of communicating agents, it is fundamental that system behavior exhibits certain desired logical properties such as absence of deadlocks and reversibility or cyclic behavior. Although mobile agent software systems have been investigated by many researchers from different points of view and diverse orientations [1], work in formal analysis of design and communication behavior in distributed systems implemented with mobile agents is still needed. The aim of this paper is to develop a theoretical framework and modeling approach for communication behavior of mobile agents in multiagent systems.

The paper is organized as follows: Section 2 gives a brief introduction to mobile agents, Petri nets and Object Petri nets and introduces a formal notion of basic agent template. Section 3 describes the proposed approach for modeling agent communications. In section 4 communications of agents are modeled based on the proposed approach. Finally, a conclusion is presented and a sketch is discussed for the future work.

2 Communication Behavior

In this section, we first introduce mobile agents briefly. Then, the proposed modeling approach of mobile agent communications using Petri nets will be described.

In mobile agent software systems servers and agents are the most fundamental concepts [2]. A mobile agent system includes a number of servers, where various resources and services are provided and computation can take place. Mobile agent paradigm has evolved from two antecedents: client-server model and remote evaluation (REV) model [8]. In client-server model processes resided in the client and server communicate synchronously either through message passing or remote procedure call (RPC) mechanism. In RPC, data is transmitted in both directions between client and server. In REV model, client sends its own procedure code to a server rather than calling a remote procedure [9].

Mobile agents can autonomously visit several hosts without the need for continuously interacting with originating host. Agents can have multiple hops and can be detached from the client without being permanently connected to the originating host. This distinguished characteristic makes mobile agent-based software systems ideal for handling temporary network connections in mobile computing. This makes mobile agents different from applets and from the servlets according to the movement pattern. An agent can visit a number of hosts and it does not need to know the complete itinerary in advance. Furthermore, the routing table of a mobile agent can be changed based on information gathered at intermediate hops during its journey in the network. Two patterns of mobility can be defined based on the state from which a mobile agent resumes execution after migration: weak migration and strong migration [10]. By weak migration, the code and part of the execution state (code and data but no control state) are moved. After migration, the execution resumes from the beginning or from a specific procedure. Strong migration allows the migration of both the code and the whole execution state (code, data and state). Mobile agent resumes execution from the point where it was stopped before migration. Other aspects of mobile agents relating to agent migration can also be investigated based on the scope of the study.

2.1 Petri Nets

We use Petri Nets to model communication behavior of software systems based on mobile agents. PNs; as a high level graphical specification language, have a sound and mature mathematical foundation. It allows a formal and direct investigation of factors such as resource conflicts, synchronization and concurrency in distributed systems. For quick reference, a brief overview of Petri nets is provided in this section, a more detailed coverage can be found in [11].

A Petri net consists of a structural part and a dynamic part. A PN structure, N, is a four-tuple, $N = (P, T, V, F)$ where $P = \{p1,p2,..., pn\}$ is a finite set of places, $n \geq 0$. $T = \{t1,t2,..., tm\}$ is a finite set of transitions, $m \geq 0$ ($T \cup P$ form the nodes of N) $V \subseteq \{(P\times T) \cup (T\times P)\}$ is a set of directed arcs (or a flow relation). F: $V \rightarrow \aleph$ is a multiplicity (incidence) function, $\aleph = \{0,1,2,3\ldots\}$. $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$

$(F\cap(T\times T) = (F\cap(P\times P) = \varnothing)$. A PN structure can be represented as a directed bipartite graph. In a Petri net graph, places are represented by circles and transitions by bars or boxes. Places and transitions are connected with directed arcs. Assignment of tokens to the places of a PN structure is called its marking and represents the state of the modeled system at each time instance. A marking μ of a Petri net $N = (P, T, V, F)$ is a mapping $\mu : P \rightarrow \aleph$. Tokens in a Petri net graph are represented by dots or positive numbers in places. The number of tokens in place p of a Petri net is formally denoted by $\mu(p)$. A place $p \in P$ is marked if $\mu(p)>0$, otherwise it is unmarked. A Petri net is marked if a marking function can be assigned to it. The state of a Petri net is defined by its marking. The dynamic part of a Petri net involves the change in markings over time.

2.2 Object Petri nets

Object Petri Nets (OPN) have been introduced as an extension of the class of colored Petri nets [12] made in a similar way to hierarchical colored Petri nets, but in an object-oriented perspective.

An OPN is a tuple OPN=(T, C, C0) where:

a) T is a set of basic types that determines the underlying component values, operators and functions.

b) C is an OPN class hierarchy with inheritance relations based on sub typing of data fields.

c) The class $C0 \in C$ is a designated root class.

OPNs support a complete integration of object-oriented concepts into Petri nets, including inheritance and the associated polymorphism and dynamic binding. A class is defined as a Petri net, which can be, as usual, instantiated. In addition to places and transitions, a class contains data fields and functions. Data fields have types that may be simple (integer, real Boolean), class, or multi-set, which generalizes classical Petri net, places. New functions can be defined assuming predefined types and functions.

2.3 Agent Model

In this section we give a definition of agent which is consistent with our approach. The environment of agents can be considered as a composite system made of agents of different kinds. Each agent is a flow of actions processing certain objects, is triggered by events, and changes the state of the system. Communicating agents have characteristics and behaviors that need to be taken into account to correctly model the behavior of the system.

An agent can be defined as an autonomous (having control over its own actions) software entity that is situated within an executing environment. It can also interact (communicate) with its environment and other agents while it is bond to certain predefined task on the user's behalf. From an object-oriented point of view, an agent is conceptualized as an encapsulated software entity that can send messages to and receive messages from other objects. It has a number of methods to process the messages and change its state as an encapsulated entity. An autonomous agent, as an active object has its own tasks that may be composed of several kinds of sequential or concurrent subtasks to be accomplished. An agent with the property of mobility (migration) between different servers is a mobile agent.

A multi-agent system is a set of communicating agents; each agent is situated in some environment and is able to interact with its environment and with other agents. This definition appears to be adequate for capturing the characteristics of the systems we are dealing with. But we need a formal definition of the concept of agent which is also consistent with our proposed approach. The definition of agent needs to be the one that can be used to compare or to combine different approaches. Moreover, the definition of agent can be used to describe relevant entities uniformly, independently of their physical nature. If a single system model has to represent different kinds of entities, a unique concept of agent provided by the definition is used to uniformly describe conceptual interfaces among them. Using mathematical notions enables us to reach a common interpretation of the concept of agent.

We model agent behavior as consisting of several concurrent actions. Each of these actions can execute in parallel to define the behavior of the agent. Actions are used to specify actual functions carried out by the agent and are performed inside the agent states. Each action may have a set of invariants that must hold during the entire life of the action. Actions are defined in the form of functions. Each function may return a result and may have a number of input parameters. While these actions execute concurrently and carry out high-level behavior, they can be coordinated using internal events. States encompass the processing that goes on internal to the agent. This processing is specified by a sequence of activities specified in a functional form. Transitions describe communications among agents. To communicate with other agents, external messages can be sent and received.

3 Hierarchical Multi Agent System

In this section we provide an OPN representation based on the definition of agent and other concepts provided in sections 2. In the proposed method the interaction between agents, or message passing, takes place through Petri net structures rather than an arc between two nodes (place or transition) as suggested in other methods in the literature. In other words, inter-agent communications happen as events via certain communicative acts containing the type and information of a message. These rules may be defined according to the structural relationships between Petri net modules that specify the entire model as a set of inter-related components or modules which hide their internal details. The advantage of this method is that the resultant net model of the system has already been extended as a correct Petri net system and there is no need for any posterior analysis while it grows in complexity. This reduces the modeling effort by a major amount. It should be noted that in Petri net modeling when the systems become large, the state-space explosion problem happens, so net system analysis becomes computationally difficult and in some cases impractical. Theoretically, the augmented PN models are guaranteed to be well-behaved regardless of the application domain and the design level. For instance, to host a mobile agent after migration, each host is supposed to provide the execution environment and the facilities for agent activation and deactivation. To accomplish its task, the mobile agent communicates with stationary environment, which consists of resources such as service agents. All these details can be modeled as PN structures, and the describing modules can then be composed and integrated to the PN model at system level. In OPN, similarly we have places and transitions but here tokens are instantiated from classes and may be changed from place to place. In fact tokens can be the system's objects or object oriented marking of system. We exemplify the proposed method in section 2 by constructing a multiagent system for virtual tourism. In this example, in addition to an object oriented approach a hierarchical architecture of analysis and design has also been observed.

After referring a tourist to system (internet site or office) a secretary fills the personal information forms and creates a Consulting Agent (CA) to assign to customer. The CA consult the tourist about his/her preferences (available places to visit, accommodation, how to travel, etc.) and costs, and help him/her to choose any of possible alternatives. Then the tourist makes a decision based on the information provided by the CA. The consulting agent then passes this information to the Envoy Agent (EA) to perform the relevant formalities (room and ticket booking and other necessary coordination). The envoy agent reports to the consulting agent after completing the assigned tasks by the CA. The tourist will be informed of the result by the consulting agent. Finally, in the end of the journey CA save all travel information in a system "log file". The consulting agent will also be removed (deleted) by the system. It should be mentioned that there does not exist a one-to-one mapping from the set of CA's to the set of EA's. This is because to each customer (tourist) a consulting agent is assigned which will be removed after finishing its mission (end of the journey. On the other hand, the number of envoy agents is usually constant and they accept various orders from different consulting agents in the system. They are able to perform concurrently. The relationship between secretary, CA and EA is depicted in Fig.1.

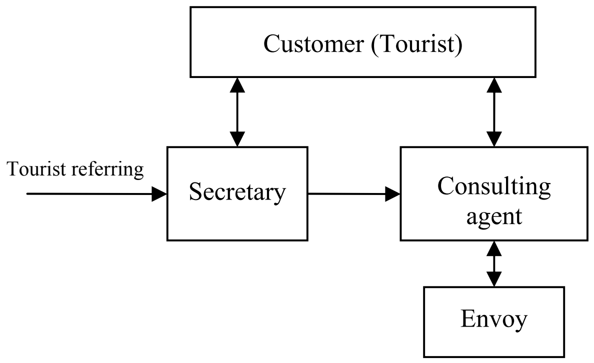

Fig.1. Interactions between types of agents

The components of the system are defined in four distinct classes as follows: the class of Secretary, the class of consulting agents, the class of envoy agents and the class of customers. These classes have been specified as follows:

Now, we can instantiate objects from these classes as tokens in the hierarchical model of object oriented Petri nets. The entire system can be represented in a 3-layer architecture. These layers are: layer 1 system (company), layer 2 Secretary and consulting agent, and layer 3 envoy agents. In OPNs each token in the PN model in a specific state is an object, which belongs to a class. Type of objects can vary as the system transforms from one state to another state.

Company or system tasks are represented in layer 1, Fig.2. Type of tokens is shown as place labels. Transitions t2 and t3 contain a series of encapsulated operations. Therefore, layer 2 can be constructed by expanding transitions t2 and t3.

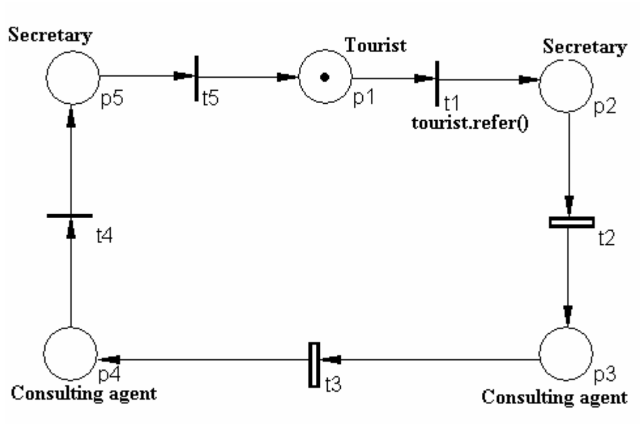

Fig.2. Layer 1 system layer

In this layer modeled as PN in Fig.2, transitions are interpreted as follows: t1 customer referring, t2 registration and assigning a consultant to a consumer, t3 performing the consulting task, t4 saving task information in system log file, and t5 removing (deleting) the consulting agent.

In OPNs an arc can be labeled (can contain) an object function. This function can be executed and its outcome will be returned to transition as input. In Fig.3, when function refer() from tourist object is invoked (in fact when a tourist is referred to the agency) transition t1 will fire.

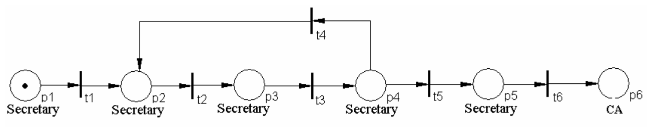

Fig4. Secretary work flow in layer 2

As shown in Fig.4, except place p4 (that is of customer type) and place p6 (that is of consulting agent type), all other places have tokens of secretary type because at this stage the main task is conducted by the secretary. Transitions are labeled as follow: t1 filling registration forms, t2 acquiring brief information about trip and estimating trip costs, t3 request for basic credit creation, t4 customer disagreement with declared costs, t5 customer agreement and creating basic credit, t6 creating a consulting agent and assign it to the customer (initialization CA). Note that places p1 and p6 in Fig.2 are the same as places p2 and p3 in Fig.3, respectively with the same token type.

The second part of layer 2 is the PN model of the consulting agent task, which corresponds to transition t3 in layer 1 (Fig.2). This part is shown in Fig.3.

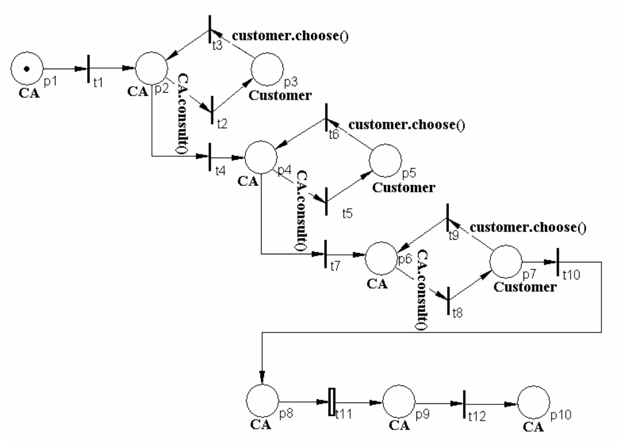

Fig.3. Consulting agent work flow in layer 2

Transition labels in PN model of Fig.3 are as follows: t1 initializing and start running, t2 consulting with customer about tourism attractions and region of trip, t3 deciding region of trip by customer, t4 confirming the region of trip, t5 consulting with customer about accommodation and duration of trip, t6 choosing accommodation and duration of trip by customer, t7 confirming the accommodation and trip duration, t8 consulting about transportation, t9 choosing the vehicle type by customer, t10 confirming the means of transportation and passing trip information to the envoy agent, t11 booking and coordination (by envoy), and t12 online consulting during the trip.

Each place can reside a token of a specific type, which depends on the agent that performs the major task. As shown in Fig. 4, except places p3, p5 and p7, which are of customer type other places will reside tokens of type consulting agent. In every cycle of choice, transition t2 (consulting) will fire if function consult() is called by the consulting object. And transition choice will fire if function choose() is invoked by the customer object.

After completing decision process about trip parameters, official tasks and formalities assigned to an envoy object, which is represented by transition t11. Thus, the envoy task constitutes third layer of the proposed architecture of the model. Note that places p1 and p10 in Fig.4 are the same as places p3 and p4 in Fig.3 with the same type.

Layer 3 represented in Fig.5; include envoy agent workflow.

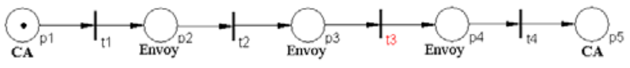

Fig.5. Envoy task makes layer 3 of architecture

Transition labels in Fig. 5 are: t1 assigning task to envoy by consulting agent, t2 acquiring the necessary information from related databases, t3 booking according to received orders, and t4 reporting the result of mission to consulting agent. Places p1 and p5 are the same as places p8 and p9 in PN model in Fig.4. This layer should contain more details than upper layers in our proposed model.

4 Conclusion

In this paper, we have presented an agent-based model using object oriented Petri nets. Our proposed model consists of three layers. Each layer has been modeled by a Petri net module. An agent has been defined to support formal reasoning for agent communications in multi agent systems. The proposed Petri nets model consists of transitions, which are the encapsulated versions of the Petri net modules in the lower levels of the system. The interfaces of each layer (modeled by Petri net modules) are defined as transitions that are unidirectional interfaces with simple data transfer capacity.

References:

[1] R. Guttman, A. Moukas, and P. Maes, "Agent-mediated Electronic Commerce: A Survey Knowledge Engineering Review," June 1998.

[2] S. Green, L. Hurst, B. Nangle, P. Cunningham, F. Somers, and R. Evans, "Software Agents: A Review," *Technical report TCD-CS-1997-06, Trinity College, Dublin*, May 1997.

[3] T.J. Rogers, R. Ross, and V.S. Subramanian, "IMPACT: A System for Building Agent Applications," J. Intelligent Information Systems, Vol. 14, 2000, pp. 95-113.

[4] F.M.T. Brazier, Dunin, B. Keplicz, N. Jennings, and J. Truer, " DESIRE: Modeling *Multi-Agent Systems in a Compositional Formal Framework," in Int'l J. Cooperative Information Systems,* Vol. 6, *Special Issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems,* 1997, pp. 67-94.

[5] M. Iglesias, Garrijo, and J. Centeno-González, "A Survey of Agent-Oriented Methodologies," in *Proc. Fifth Int'l Workshop (ATAL-98)*, 1998, pp. 317-330.

[6] M. Wooldridge, and N.R. Jennings, "Special Issue on Intelligent Agents and Multi-Agent Systems," *J. Applied Artificial Intelligence,* 1996.

[7] M. Petit, P. Heymans, and P.Y. Schobbens, "Agents as a key concept for Information Systems Engineering Requirements," *Position paper, Dept. Comp. Science*., Namur Univ, Belgium, 1999.

[8] M. Kolp, P. Giorgini, and J. Mylopoulos, "Organizational multi-agent architectures: A mobile robot example," *Proc. AAMAS*, 2002, Bologna, Italy, 2002, pp. 94-95.

[9] G. Di. Marzo Seregeundo, et al, "Survey of theories for mobile agents," *Technical report,* No. 106, Geneva Univ, 1996.

[10] H. Xu, and S.M. Shatz, "An Agent-Based Petri Net Model with Application to Seller/Buyer Design in Electronic Commerce," in *Proc. Fifth Int. Symposium on Autonomous Decentralized Systems (ISADS 2001*), March 26-28, Dallas, Texas, USA 2001, pp. 11-18.

[11] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE,* Vol.77, No.4, April 1989, pp. 541-580.

[12] T. Miyamoto, "A Survey of Object-Oriented Petri Nets and Analysis Methods," *IEICE Trans. Fundamentals*, VOL.E88–A, NO.11 November 2005.