# Object-Oriented Petri nets Based Architecture Description Language for Multi-agent Systems

***Zhenhua Yu and Yuanli Cai***

***School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, 710049, China***

**Summary**
To narrow the gap between multi-agent formal modeling and multi-agent practical systems, multi-agent systems (MAS) are studied from the point of view of software architecture. As the existing architecture description languages (ADLs) are not suitable for describing the semantics of MAS, a novel architecture description language for MAS (ADLMAS) rooted in BDI model is proposed, which adopts Object-Oriented Petri nets presented in this paper as a formal theory basis. ADLMAS is suitable for representing concurrent, distributed and synchronous MAS, and it is brought directly into the design phase and served as the high-level design for MAS implementation. ADLMAS can visually and intuitively depict a formal framework for MAS from the agent level and society level, describe the static and dynamic semantics, and analyze, simulate and validate MAS and interactions among agents with formal methods. To illustrate the favorable representation capability of ADLMAS, an example of multi-agent systems in electronic commerce is provided. Finally, the MAS model and its key behaviors properties are analyzed and verified.

***Key words:***
*Multi-agent systems, software architecture, architecture description language, Object-Oriented Petri nets, BDI model*

## 1. Introduction

Multi-agent systems (MAS) have been recognized as a main aspect of the distributed artificial intelligence and predicted to be the future mainstream computing paradigm. MAS are the most promising technology to develop complex software systems, and many attentions have been paid to MAS in complicated, large-scale and distributed industrial and commercial applications [1], [2].

MAS are adaptive and flexible systems in order to adapt to changes in their environment, in which agents may be added or deleted at run-time, and the agent behaviors and interactions among agents may vary dynamically [3], so there exist many difficulties in analyzing the structure and behaviors of MAS. There is a pressing need for a formal specification to support the design and implementation of MAS, and ensure the developed systems to be robust, reliable, verifiable, and efficient [5]. It has been recognized that the lack of rigor is one of the major factors that hamper the wide-scale adoption of multi-agent technology [4]. A rigorous approach toward MAS architecture level design can help to detect and eliminate design errors early in the development cycle, and thus to reduce overall development cost. In the past several years, some work has tended to investigate the formal modeling techniques of MAS. However, the effort in multi-agent systems modeling suffers from lack of systematic approach that is grounded in software development methodologies.

In this paper, to provide effective support for the development of correct, robust and dynamic MAS in a systematic way, a formal specification, called architecture description language for multi-agent systems (ADLMAS), is proposed. Our proposed formalism studies MAS from the point of view of software architecture. Architecture Description Language (ADL) [15] describes software architecture in a formal way, represents software design at the high level rather than the implementation details of any specific source modules. So far, many ADLs have been proposed for representing and analyzing software architecture, however the existing ADLs are difficult and not suitable for accurately describing the architecture, complex dynamic characteristics and reasoning of MAS. The ADLMAS adopts Object-Oriented Petri nets (OPN) as its formal theory bases. The OPN are graphical and mathematical modeling tool, which is simplicity and strong expressive power in depicting system structure and dynamic system behaviors. A notable benefit of using OPN is its modular and object-based approach for the specification and prototyping of complex software system. Most importantly, OPN supports formal analysis of MAS architecture in a variety of well-established techniques, such as simulation, deadlock detection, reachability analysis and model checking. As the Belief-Desire-Intention (BDI) model is well suited for describing an agent's mental state, ADLMAS roots in the BDI model.

ADLMAS is brought directly into the MAS design phase, and the formal MAS model is served as the high-level design for MAS implementation. ADLMAS is a graphical-intuitive language with formal and precise semantics to handle concurrency and synchronization, which can not only depict complex dynamic structure, but

Manuscript revised January 29, 2006.

also describe the static and dynamic semantics and provide a basis for verification and validation of the functionality of MAS. The ultimate goal of ADLMAS is to provide a tool that generates executable implementation skeletons from a formal model and enables software engineers to develop reliable and trustworthy MAS.

The rest of this paper is organized as follows: Section 2 overviews the related work; Section 3 presents Object-Oriented Petri nets. Section 4 proposes a novel architecture description language based on Object-Oriented Petri nets for multi-agent systems. Section 5 discusses MAS modeling process based on ADLMAS. Section 6 provides an agent society in electronic commerce to illustrate ADLMAS. Finally, Section 7 summarizes the results of this paper and discusses further research directions.

## 2. Related Work

So far, there have existed several typical formal specifications and agent-oriented methodologies for MAS, which can be classified into four main groups. The first group uses formal languages, such as Z, to provide a formal theory basis for representing MAS. dMARS [6] is an agent specification using the Z language as its formal theory basis. In dMARS, agent's beliefs, goals, intentions, plans, and actions are all described using Z. RIO framework [7] represents MAS based on Object-Z and state-charts, which uses Object-Z to specify the transformational aspects and state-charts to specify the reactive aspects. Although Z is precise and unambiguous, and facilitates the system description at different levels of abstraction, a key criticism using Z is that it cannot effectively model the interactions among agents and support the effective definition of concurrent and distributed MAS, and it is less expressive with regard to mental states of agents.

The second group of researches uses temporal logics and multimodal logics to describe dynamic aspects of the agents that form a basis for specifying, implementing and verifying MAS. In the Concurrent METATEM [8], the temporal logic is applied to describe individual agent behaviors where the representation can be executed directly, verified with respect to a logical requirement, or transformed into a more refined representation. However MAS based on concurrent METATEM have no explicit architecture and interactions among agents are vague. DESIRE [4] based on temporal logic focuses on hierarchical task-based decomposition and provides a much clear and more readily comprehensible description of the application. Although these formal specifications are claimed to represent MAS, it is impractical to use a logic notation directly in the specification and reasoning about large-scale MAS, because such the specification will be a complicated logic formula that consists of mathematical notations and symbols. Such formalism has led to a sizable gap between these formal models and implemented MAS.

The third group consists of some new formal languages that support the formal specification and verification MAS, such as SLABS [9], agent-based G-net [5], etc. SLABS includes a modular structure suitable for the formal specification of multi-agent systems, a scenario description mechanism for defining agents behavior in the context of environment situations, and a notion of caste as a collection of agents that have same behavior and structural characteristics. Agent-based G-net, which is a type of Petri nets, is explicitly oriented for specifying and defining the design architecture of multi-agent software systems and illustrates a useful role for inheritance in the agent-oriented models. However, agent-based G-net does not provide adequate means to describe the BDI model which is a crucial characteristic of MAS. Furthermore the set of methods in agents based on G-net is fixed and may not adapt by changing their knowledge-base, goals and plans, not by reconfiguring, adapting or exchanging their methods [14].

The fourth group is agent-oriented development methodologies, such as Gaia [10], MaSE [11], AUML [12], Tropos [20], DECAF [21], and framework for MAS [22] development, etc. Gaia methodology emphasizes a few models that can be utilized to form the whole system. It describes what these models are, but the processes used to develop these models are vague [13]. Moreover Gaia requires that a single agent abilities and agent relationships remain static at run-time which makes the agent lack of autonomy. MaSE consists of seven phases to develop MAS. The goal of MaSE is to lead the designer from the initial system specification to the implemented agent system. MaSE requires that agent-interactions are one-to-one and not multicast [13]. Gaia supports MAS development in both the micro-level and macro-level, including analysis and design processes. AUML is an extension of UML to develop MAS. AUML addresses only the interactions among agents and does not facilitate the representation of reasoning and proactive nature of MAS, moreover it is only a semi-formal specification, and cannot verify and validate MAS. Tropos covers the very early phases of requirement analysis and the conceptual modeling is formalized in a metamodel described with a set of UML class diagrams. However, one criticism of this approach is that it does not provide strong support for protocols and modeling the dynamic aspects of the system.

Despite the important contribution of these four groups of formalisms and agent-oriented methodologies to a solid underlying foundation for MAS, most formal specifications are not oriented for software engineering in terms of providing a modeling notation that directly supports software development and how an implementation can be derived, and less expressive with regard to mental state of agents. These are challenges for formal modeling formalism. According to the above description, a preferable formal language for modeling MAS should satisfy the following requirement:

(i) The language should precisely and unambiguously describe the structure and behaviors of MAS in a readable and understandable manner;
(ii) The language should allow agents to be specified by using a combination of graphics and text;
(iii) The language should hide some details when necessary, and describe MAS at different levels so that the developers can effectively understand MAS;
(iv) The language should depict static and dynamic semantics, and provide tool support for modeling, analysis and verification;
(v) The language should be oriented for software engineering and easily implemented.

Our proposed ADLMAS can satisfy the above requirements. We use OPN to visualize the agent structure, agent behavior, and agent functionality for intelligent agents, and use well-established methods to analyze the model. ADLMAS is brought into the MAS design phase and served as the high-level design for agent implementation. ADLMAS is oriented for software engineering, therefore can effectively narrow the gap between MAS formal models and MAS implementation.

## 3. Object-Oriented Petri nets (OPN)

Petri nets are a graphical and mathematical modeling tool applicable to many systems that exhibit concurrency and synchronization [16]. The ordinary Petri nets, which highly depend on the system and lack the modularity and flexibility, easily lead to the so-called state-explosion. In order to solve the complexity and state-explosion, Petri nets are combined with Object-Oriented methods to set up the Object-Oriented Petri nets. OPN can tersely and independently represent all kinds of resources in a complex system, increase the flexibility of the model, discover the design mistakes in the earlier stage, and shorten the modeling cycle. In the OPN model, a system is composed of mutually objects and their interconnection relations; the formal definition is given as follows.

**Definition 1.** OPN is a 2-tuple, OPN=(O, MPR), where O is a finite set of physical object in the system, $O=\{O_1, O_2, \ldots, O_i\}$; MPR is a finite set of message passing relations among physical objects.

**Definition 2.** $O_i$ is a 9-tuple, $O_i$= (P, IP, OP, T, F, IIA, OIA, E, C), where P is a finite set of places, $P=\{p_1, p_2, \ldots, p_j\}$; IP (Input Place) is a set of input message places in OPN, $IP=\{ip_1, ip_2, \ldots, ip_l\}$; OP (Output Place) is a set of output message places, $OP=\{op_1, op_2, \ldots, op_m\}$; T is a finite set of physical object transitions in the system, $T=\{t_1, t_2, \ldots, t_k\}$; $F \subseteq (P\times T)\cup(T\times P)\cup(IP\times T)\cup(T\times IP)\cup(OP\times T)\cup(T\times OP)$ is the input and output relationships between transitions and places; IIA (Input Interface Arc) is a set of the input transition arc from outside to OPN, $IIA= \{iia_1, iia_2, \ldots, iia_n\}$ [17]; OIA (Output Interface Arc) is a set of output transition arc from OPN to outside, $OIA= \{oia_1, oia_2, \ldots, oia_o\}$ [17]; E: $F\rightarrow$ (ID, CDS) is expression functions in the arcs, ID is the identification of the arc and CDS is a complicated data structure; C(P) is a set of color associated with the places P, $C(P) = \{cp_1, cp_2, \ldots, cp_j\}$; C (IP) and C (OP) are sets of color associated with the input and output message places.

**Definition 3.** MPR is defined as MPR= (ILP, C), where ILP is the Intelligent Linking Place denoted by ellipse. The information obtained from the external is saved in the ILP. Each OPN dispatches the information by ILP. C (ILP) is a set of color associated with the ILP.

In the OPN model, some concepts of CPN are employed and some behavioral semantics does not violate the semantics of CPN formalism. In the places of OPN, data types of Token are defined, which can express complex data structures or objects. IP and OP are responsible for internal message passing, and message dispatching among objects.

## 4. Novel Architecture Description Language for Multi-agent Systems (ADLMAS)

ADLMAS is suitable for describing MAS architecture which possesses the advantages of semantics strictness and precision of the traditional program languages, and defines the abstract elements for MAS architecture. The main design object of ADLMAS provides a dynamic architecture modeling mechanism aiming at the complex dynamic characteristics of MAS, and lets the MAS architecture serve as the high-level design for MAS implementation.

In order to accurately describe MAS architecture, OPN presented in this paper are adopted as a formal theory basis of ADLMAS. OPN are a graphical and mathematical modeling tool, and are suitable for describing the large-scale, complicated and distributed MAS.

ADLMAS should provide with the essential characteristics of ADL. The traditional ADL mainly describes the components, connectors, in which components and connectors as modeling elements. A component is a unit of data or computation, loci of status store and computation with extended and integrated; a connector is used to model interactions among components and rules that govern those interactions. Agents are modeling elements in MAS. An agent is an encapsulated computer system that is situated in some environment and can act flexibly and autonomously in that environment to meet its design objectives [18]. There exists much difference between components and agents in semantics and function. In order to provide the uniform semantics, ADLMAS substitutes computing agents and connecting agents for components and connectors as the computation and interaction elements in MAS. ADLMAS studies MAS from the agent level and society level: The agent level to the structure of each agent, and the society level to a formal framework for MAS and interactions among agents.

**Definition 4.** ADLMAS is a 3-tuple, ADLMAS= (Computing agents, Connecting agents, Configurations).

## 4.1 Modeling computing agents

Computing agents are a finite set of the computing agent in MAS. Computing agents are responsible for interacting with users and environment to provide specific applications. A computing agent is based upon the Belief-Desire-Intention (BDI) model, which is used to describe its mental states.

A computing agent is a 2-tuple, Computing Agent = (ID, AS), where ID is the identifier of a computing agent; AS (Agent Structure) is the tuple $O_i$ in the OPN model which defines the interfaces and internal implementation of a computing agent. AS based on the BDI model is composed of the Knowledge-base module, Goal module, Plan module, and Interface module. The modules are described as follows: in practical terms, the Knowledge-base module corresponds to the agent's Beliefs, which describes the knowledge of the environment and other agents. The Beliefs of an agent may be represented as simple variables and data structures or, complex systems such as knowledge bases. The Goal module corresponds to the agent's Desires, which describes some desired final states and consists of a goal set. The Desires of an agent represent its motivation and are the main source for the agent's actions. The Desires may be associated with a value of a variable, a record structure, or a symbolic expression in some logic so that desires can be prioritized. The Plan module corresponds to the agent's Intentions, which is a list of plans and describes the actions achieving the Goal values of an agent [19]. The interface module allows a computing agent to interact with other agents and the environment, and is used to send and receive messages between agents. The interfaces specify the services (messages, operations, and variables) which a computing agent requires and provides, and are especially the channels for messages passing. A computing agent model is shown in Fig. 1, where Private Utilities represent private method and utilities, such as register and destroy information; the Knowledge-base, Goal and Plan are denoted by ellipses. As Fig. 1 only represents a template of a computing agent, interfaces and internal implementation are added according to the specific system requirements, and the BDI model can be refined.

For simplicity and clarity of the diagrams, only names of places, transitions and arcs of all agents models are presented in this paper, and inscriptions, colors, guards and marking are left unspecified.

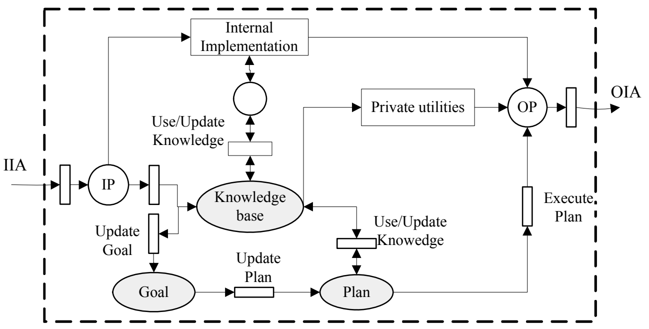

Fig. 1 A computing agent model

Agents communicate with other ones by message passing, which follows speech act theory and uses complex protocols to negotiate [5], e.g., the FIPA agent communication language(ACL) and KQML. Communication is the basis for interaction and organization without which agents would be unable to cooperate, coordinate, or sense changes in their environment. The agents proposed in this paper speak and understand FIPA ACL. In agent model, a message is described as a message token which is abbreviated to msgTkn. msgTkn is a 2-tuple msgTkn=(mID, body), where mID represents a message holds an unambiguous identification and body is a complicated data structure. . More specifically, the msgTkn body is defined as follows:

```
struct msgBody{
  int sndAgent;    // the identifier of the sending message
                      agent
  int recAgent;    // the identifier of the receiving message
                      agent
  int recAgentInterface;   // the identifier of the interface
                              of the receiving message agent
  string protocolType;   // the type of protocol
  string msgName;   // the name of the message
  string msgContent;   // the content of the message
}
```

When a computing agent first receives a message, a conversation is set up which is responsible for messages passing among agents; meanwhile, the message Token is dispatched into the "Internal Implementation" and further dealt with, and simultaneously updates the Knowledge base, Goal and Plan. The messages belonging to the conversation hold an unambiguous identification (mID). If an agent next receives a message carrying such a reference to an existing Token, the message will be directly dispatched into the knowledge base, and executed according to the former experience.

## 4.2 Modeling connecting agents

Connecting agents are a finite set of the connecting agent which is communication facilitator dealing with the interaction information among agents and defining the rules that govern those interactions. Fig. 2(a) describes a MAS consisted of a communication facilitator and some computing agents, and such MAS is called a group where agents are to achieve a certain goal. Fig. 2(b) represents several groups constitute a large-scale MAS, and these groups is connected by a communication facilitator.

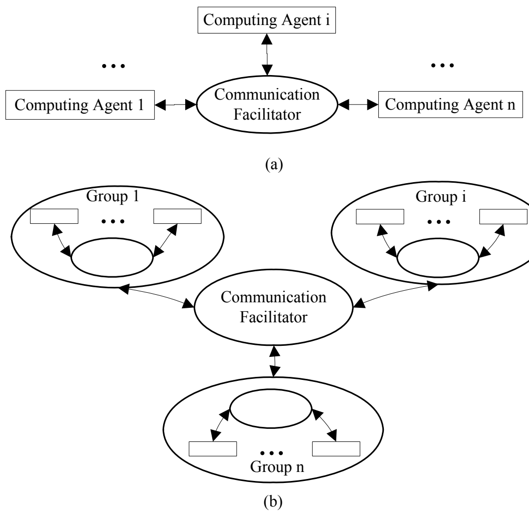

Fig. 2 The communication model of MAS. (a) the communicationmodel of a group, (b) the communication models of multi-group.

A connecting agent is defined as Connecting Agent= (MPR, KBP, T, F, Role), where MPR (Message Passing Relations) is the tuple in OPN model; KBP represents Knowledge Base Place which is defined to apperceive the external environment, acquire requisite knowledge, and describe services which computing agents provide via interfaces. Role is a set of interfaces in computing agents, which is defined as Role = {CID1… CIDn}, where CIDi is the identifier of the computing agent. The services provided by the role are stored in the KBP. There are two types of roles, static and dynamic role respectively. Dynamic role will change with the computing agent deleted or added.

Connecting agents are not only responsible for message passing of multi-agent systems, but also manage the common knowledge of the MAS. From the point of view of communication, connecting agents control and manage the communication and collaboration among agents; from the point of view of the system connection and conglutination, connecting agents play the role of glue conglutinating the MAS.

In MAS, computing agents first enroll their information (such as name, address, interface and capability) in connector agents. If a computing agent achieves its goal, it must delete its information, and then the information in connecting agents will not fall into confusion. If a computing agent requests a service, the connecting agent queries its knowledge base to search a corresponding computing agent providing the service. When the request computing agent receives the identifier of the service computing agent, it sends the message to the service computing agent by the connecting agent. If the service computing agent does not exist, the request computing agent can subscribe for this service. The connecting agent will inform the request computing agent as long as it becomes aware of the information that a corresponding computing agent registers.

In ADLMAS, computing agents and connecting agents describe agent structure from the agent level, as well as the behaviors and interfaces of the individual agent.

## 4.3 Modeling Configurations

MAS configurations are connected graphs of computing agents and connecting agents that describe architectural structure. Explicit architectural configurations facilitate communication among a system's many stakeholders, who are likely to have various levels of technical expertise and familiarity with the problem at hand [15].

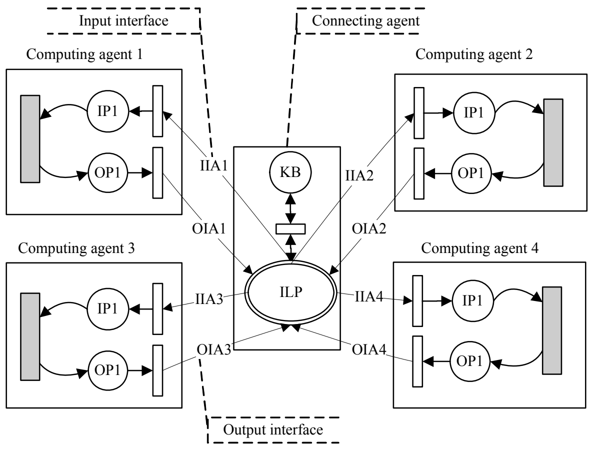

Fig. 3 MAS architectural configurations.

In MAS, existing agents cooperate towards some purposes beyond an agent's ability. The multi-agent systems architecture can not only describe individual agent, but also depict the whole system and interaction among agents. The multi-agent systems architectural configuration based on ADLMAS is shown in Fig. 3, and the MAS are studied from the society level, where MAS are conceived as a multitude of interacting agents. In the society level, the key point is the overall behaviors of the MAS, rather than the mere behaviors of individuals. For simplicity and clarity of the diagrams, this model is predigested. The computing agents are represented by IP, OP and abstract transitions denoted by shaded rectangles. The abstract transitions can be refined into subnets shown in Fig. 1. This architecture consists of four computing agents and one connecting agent.

The computing agent connects with the connecting agent by the interface; therefore an arborescent topology is formed. The static semantics of the multi-agent systems architecture is described in Fig. 3, and the dynamic semantics of the multi-agent systems architecture is controlled by the firing rule. The firing of the transition makes the Token dispatch, which expresses the message passing and well depicts interactions among agents. The purpose of modeling multi-agent systems in ADLMAS is to make full use of the well-established analysis methods proposed for Petri nets. These methods are commonly used to detect the deadlock, and boundedness properties of systems models. ADLMAS can systematically analyze, verify and validate the properties of the implemented system.

ADLMAS is a visual ADL, which can make users effectively understand and analyze MAS before MAS are implemented, and narrow the gap between agent formalism and practical systems.

## 5. MAS Modeling Process Based on ADLMAS

The purpose of the proposed ADLMAS is to ease the developer's effort to implement complicated applications of MAS. In order to design a MAS using ADLMAS, the requirement specification should be decomposed and described by formal methods, and then the computing agents and connecting agents are identified. In the early process of MAS modeling, the exact detailed information of the system is not known. Thus the detailed information and constraints can be temporarily ignored to simplify the modeling complexity of MAS, and a basic MAS model can be constructed to represent the static characteristics and dynamic behaviors. In this way, each agent model is reusable for future modeling. In the next step, these models are refined with the constraints and interrelations among agents, and analyzed (e.g. deadlock and boundness). Then, a complete MAS model based on ADLMAS may be constructed. The procedure for constructing a complete MAS model based on ADLMAS is summarized as follows.

(i) According to the system specification, the computing agents and connecting agents are distinguished, and the function of each agent is defined.

(ii) Define and initialize a set of goals $\Phi$ in the computing agent, where each goal is defined as a goal tree $\Gamma$, which means a goal may have a number of subgoals. The goal set is dynamic, which means the goals accomplished may be deleted from $\Phi$ and newly goals could be added into $\Phi$ at run time. Finally according to the template of the computing agent, the OPN model of the Goal module should be set up.

(iii) Define a set of plans P in the computing agent. Each plan has a priority and a set of conditions, and is associated with a particular goal or subgoal. Finally according to the template of the computing agent, the OPN model of the Plan module should be set up.

(iv) Define and initialize the knowledge base in the computing agent and connecting agent, and an interaction protocol among agents. The knowledge base is dynamic. Finally according to the template of the computing agent, the OPN model of the Knowledge module should be set up. As a result of the execution of a plan, the knowledge base may be updated at run time.

(v) Set up the MAS architecture, and simulate and analyze it with the supporting tools and analysis methods of Petri nets. If the model is not correct, we should return step 2 to redefine the MAS model until it is correct. Finally, we implement the MAS model.

There exists some feedback and adoptions of design information between steps. These steps can be performed in an iterative and incremental way. From the modeling process, this modeling approach based on ADLMAS follows the natural style of human thinking: Desire-Intention-Belief, rather than Intention-Desire-Belief.

The goal of ADLMAS is to lead the designer from the initial system specification to the implemented MAS, and further support for automatic code generation. ADLMAS has been successfully applied to Kunming Police Geographical Information System (KPGIS), which is a large-scale, multilevel, and distributed multi-agent system. The application of ADLMAS demonstrated that ADLMAS can help architecture designers to effectively analyze and design the complex, distributed and concurrent MAS. At present, a visual integrated development tool based on ADLMAS has been developed; the MAS architecture can be modeled and analyzed by this tool, and the development process will be discussed in detail in our future working paper.

## 6. An Example of Multi-agent Systems in Electronic Commerce

In this section, a multi-agent system in an electronic commerce is considered. The buyer agents and seller agents negotiate price, and finally the buyer agents determine whether to buy or not. The MAS architecture based on ADLMAS is set up, and then the model is analyzed by mathematical methods of Petri nets to ensure a correct design.

### 6.1 MAS modeling in electronic commerce

The architecture of the price negotiation MAS in electronic marketplace based on ADLMAS is shown in Fig.

4, which represents a pair wise negotiation process. The MAS is composed of three functional agents (one buyer agent and two seller agents) bargaining for goods. The seller agents are abstracted as abstract transitions, interface arcs, input places, and output places. This is feasible because agent models can only interact with each other through interfaces. For simplicity, the Goal, Plan, and Knowledge-base modules are not refined in more detailed units, but remain as abstract units; some constraints are omitted in this figure. The arc *OIA3* describes the interface that registers or destroys the agent information, the arc *OIA1* describes the output interface that sends a call for price proposals, the arc *IIA1* represents the interface that receives the proposals from other agents, and the arc *OIA2* represents the interface that executes the buying plan.

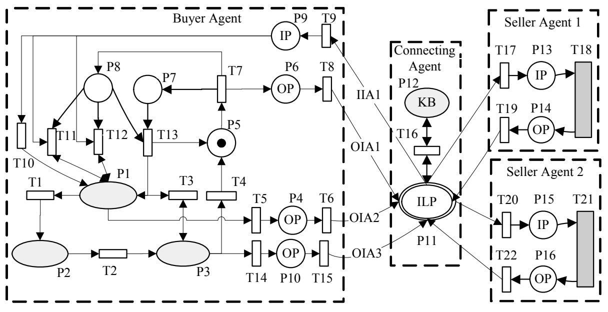

Fig. 4 MAS model in electronic commerce.

The legend provided in Table 1 and Table 2 describes the meaning of each place and transition in Fig. 4. The negotiation process can be described by ML. When a negotiation begins, each agent must register its basic information to the connecting agent. The connecting agent can accept or reject the registration based on the enrolled agent's reputation or function. The buyer agent starts the conversation by sending a call for price proposal to the connecting agent through the interface *OIA1*, then its state changes to *Waiting* (P8). An msgTkn token with an unambiguous identification will be deposited and dealt with in the connecting agent, and then the connecting agent sends it to the corresponding seller agents. The seller agents send the price proposals to the buyer agent by the interface *IIA1*. Upon the arrival of the price proposals, the buyer agent thinks whether the price proposals are acceptable or not with the help of its *knowledge-base* (P1). If the proposals are rejected, the transition *reject proposal* (T12) will fire and update the *knowledge-base* (P1), and the buyer agent will negotiate again. If the proposals are accepted, the transition *accept proposal* (T11) will fire and update the *knowledge-base* (P1). Finally the plan *plan_buy* (T14) will be generated and executed. To ensure the system is robust, the timeout mechanism is adopted, and this triggers the exception action (the transition *throw exception*) to stop the buyer agent from the waiting state, update *knowledge-base* (P1), and continue to send *call for price proposal* (T4). By then, the conversation of price negotiation is finished. When the buyer agent receives a message carrying identification the same as the existing message identification, the transition *deal with similar price proposal* (T10) will be enabled. The message is directly dispatched into the place *knowledge-base* (P1), dealt with according to the previous experience, and finally the corresponding plan will be executed.

Table 1: Legend of Fig. 4 (description of places).

| Place | Description |
|---|---|
| P1 | The abstract place for the knowledge-base module o the buyer agent. |
| P2 | The abstract place for the goal module of the buye agent. |
| P3 | The abstract place for the plan module of the buye agent. |
| P4/P6/P10/P14/ P16 | The places for dispatching outgoing messages. |
| P5 | The places for initial call for price. |
| P7 | The place for timeout mechanism. |
| P8 | The place for waiting price proposal. |
| P9/P13/P15 | The places for dispatching incoming messages. |
| P11 | The ILP place of the connecting agent. |
| P12 | The abstract place for the knowledge-base module of the connecting agent. |

Table 2: Legend of Fig. 4 (description of transitions).

| Transition | Description |
|---|---|
| T1 | The transition *update goal* that updates the goal set. |
| T2 | The transition *update plan* that updates the plan set. |
| T3 | The transition *update or use kb* that updates or uses the knowledge-base. The plan set can make use of the knowledge base. As a result of the execution of a plan, knowledge base may be updated |
| T4 | The transition *call for new price proposal* that sends a new price proposal for seller agents. |
| T5 | The transition *register or destroy* that registers the buyer agent, or deletes its information if it achieves its goal. |
| T6/T8/T15/T19/T22 | The transitions related to the *OP* places. |
| T7 | The transition *send call for price proposal*. |
| T9/T17/T20 | The transitions related to the *IP* places. |
| T10 | The transition *deal with similar price proposal*. |
| T11 | The transition *accept proposal* means the seller agent accepts the proposal which the seller agents propose. |
| T12 | The transition *reject proposal* means the seller agent rejects the proposal. |
| T13 | The transition *throw exception* is the exception process. |
| T14 | The transition *execute buy plan* that executes th acquired plan. |
| T16 | The transition *query or update knowledge-base*. |
| T18/T21 | The abstract transition represents the BDI module, internal implementation and private utilities of the seller agent |

The dynamic semantics is represented as follows. In ILP of the connecting agent, when the number of Tokens is greater than one, according to the message mID and body of a Token, the Token is sent to the IP in the corresponding agent and then dispatched. When the output result is produced in an agent, Tokens in the OP are sent to the ILP and then Tokens are dispatched. All the Tokens associated with an input interface in an agent are formed a message queue in ILP, and follow the rule of "First Come First Serve". If there are two Tokens in ILP, corresponding with two different interfaces in an agent respectively, the transitions meet the fire rule, then they can fire and execute concurrently.

ADLMAS visually describes the dynamic semantics of the multi-agent systems architecture. The firing of the transition makes the Token dispatch, which expresses the message passing and well depicts interactions among agents.

## 6.2 Analysis of MAS model in electronic commerce

A significant advantage provided by ADLMAS based on Petri nets is that the verification and validation of the model can be accomplished before implementation, and help ensure a correct design (such as liveness, deadlock freeness, boundness and concurrency) with respect to the original specification to enable software engineers to develop reliable and trustworthy MAS. In this section, the deadlock of the MAS model is analyzed. It is important that how to handle deadlock situations for development of electronic commerce systems and operating systems, where the communication plays a key role.

The theory of invariants [16] is employed as the deadlock detection method to analyze the simplified MAS model.

**Theorem 1.** Let $N$ is a Petri net model, an $n$-vector $I$ is a $P$-invariant (place invariant) of $N$ if and only if $\boldsymbol{I}^{\mathbf{T}} \bullet [\boldsymbol{N}] = \boldsymbol{0}^{\mathbf{T}}$. $\|\boldsymbol{I}\| = \{p \in P | \boldsymbol{I}(p) \neq 0\}$ is called the support of an invariant. If all $P$-invariants are marked in the initial marking and there are no empty siphons, the $N$ is live [16].

By analyzing, there are three $P$-invariants in the MAS model, and their supports are $\| \boldsymbol{I}_1 \| = \{P12\}$, $\|\boldsymbol{I}_2\| = \{P1, P2, P3, P4, P5, P6, P8, P9, P10, P11, P13, P14, P15, P16\}$, $\|\boldsymbol{I}_3\| = \{P1, P2, P3, P4, P5, P6, P7, P9, P10, P11, P13, P14, P15, P16\}$ respectively. All $P$-invariants are marked in the initial marking; moreover there are no empty siphons, so the model is live.

Deadlock analysis can help eliminate human errors in the design process, and verify some key behaviors for the MAS model to perform as expected, and increase confidence in the MAS design process.

## 7. Conclusions

Multi-agent systems are regarded as the most promising technology to develop complex software systems. Formal framework for MAS provides a base to design, verify and implement MAS, and ensures that robust, reliable, and efficient software is developed. In this paper, from the software architecture point of view, a novel architecture description language for MAS (ADLMAS) rooted in Petri nets is proposed to support the modeling and analysis of multi-agent systems. ADLMAS based on Belief-Desire-Intention (BDI) agent model stresses practical software design methods instead of reasoning theories, and analyze the static and dynamic semantics, and depict the overall and individual characteristics of MAS. ADLMAS can be applied to investigate MAS from the agent level and society level. From the agent level, the researchers can pay more attention to the implementation details of each agent; and from the society level, they can pay more attention to the overall design and interactions among agents. An example of an agent society in electronic marketplace is used to illustrate modeling capability of ADLMAS; and moreover, how to detect the deadlock in the MAS model by the theory of invariants is discussed. ADLMAS, as a visual ADL, can promote the intercourse and understand among clients, architecture designers and developers, and provide an effective modeling method for MAS modeling and verifying.

The tool kit based on ADLMAS will be considered in our future work, which can support automatic code generation. Also the learning ability of MAS will be further investigated.

**Acknowledgment**

This work is supported by the National High Technology Research and Development Program of China (863 Program), No. 2003AA721070. We thank all anonymous referees for the careful review of this paper and the many suggestions for improvements they provided.

## References

[1] F. Zambonelli and A. Omicini, "Challenges and research directions in agent-oriented software engineering", Autonomous Agents and Multi-Agent Sytems, vol. 9, no. 3, pp. 253-283, 2004.
[2] M. Luck, P. Mcburney, and C. Preist, "A manifesto for agent technology: towards next generation computing", Autonomous Agents and Multi-Agent Sytems, vol. 9, no. 3, pp. 203-252, 2004.
[3] W. Jiao, M. Zhou, and Q. Wang, "Formal framework for adaptive multi-agent systems", Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology, pp. 442-446, 2003.
[4] F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur, "DESIRE: modelling multi-agent systems in a compositional formal framework", International Journal of Cooperative Information Systems, vol. 6, no. 1, pp. 67-94,

1997.
[5] H. Xu and S. M. Shatz, "A framework for model-based design of agent-oriented software", IEEE Transactions on Software Engineering, vol. 29, no.1, pp. 15-30, 2003.
[6] M. Luck and M. d'Inverno, "A formal framework for agency and autonomy", Proceedings of First Int'l Conf. Multi-Agent Systems, pp. 254-260, 1995.
[7] P. Gruera, V. Hilairea, A. Koukama, and K. Cetnarowicz, "A formal framework for multi-agent systems analysis and design", Expert Systems with Applications, vol. 23, no. 4, pp. 349-355, 2002.
[8] M. Fisher and M. Wooldridge, "On the formal specification and verification of multi-agent systems", International Journal of Cooperative Information Systems, vol. 1, no. 6, pp. 37–65, 1997.
[9] H. Zhu, "SLABS: a formal specification language for agent-based systems", International Journal Software Engineering and Knowledge Engineering, vol. 11, no. 5, pp. 529-558, 2002.
[10] M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia methodology for agent-oriented analysis and design", International Journal of Autonomous Agents and Multi-Agent Systems, vol. 3, no. 3, pp. 285– 312, 2000.
[11] S. DeLoach, "Multiagent Systems Engineering", Proceedings of Agent Oriented Information Systems, pp. 45-57, 2000.
[12] J. Odell, H. V. D. Parunak, and B. Bauer, „Representing agent interaction protocols in UML", Proceedings of 1st International Workshop on Agent Oriented Software Engineering, pp. 121-140, 2000.
[13] T. I. Zhang, E. Kendall, and H. Jiang, "A software engineering process for BDI agent-based systems", Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology, pp. 392-399, 2003.
[14] M. Köhler, D. Moldt, and H. Rölke, "Modelling the structure and behaviour of Petri net agents", Lecture Notes in Computer Science, vol. 2075, pp. 224-241, 2001.
[15] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages", IEEE Transactions on Software Engineering, vol. 26, no. 1, pp. 70-93, 2000.
[16] T. Murata, "Petri nets: properties, analysis, and application", Proceedings of the IEEE, vol. 77, no. 4, pp. 541-580, 1989.
[17] J. A. Saldhana and S. M. Shatz, "Formalization of object behavior and interactions from UML models", International Journal of Software Engineering and Knowledge Engineering, vol. 11, no. 6, pp. 643-673, 2001.
[18] N. R. Jennings and S. Bussmann, "Agent-based control systems: Why are they suited to engineering complex systems", IEEE Control Systems Magazine, vol. 23, no. 3, pp. 61-73, 2003.
[19] K. M. Kavi, M. Aborizka and D. Kung, "A framework for designing, modeling and analyzing agent based software systems", Proceedings of 5th International Conference Algorithms and Architectures for Parallel Processing, pp.196 – 200, 2002,.
[20] P. Bresciani, A. Perini, etc, "Tropos: an agent-oriented software development methodology", Autonomous Agents and Multi-Agent Sytems, vol. 8, no. 3, pp. 203-236, 2004.
[21] J. Graham, K. Decker, and M. Mersic, "DECAF-A flexible multi agent system architecture", Autonomous Agents and Multi-Agent Systems, vol. 7, no. 1, pp. 7-27, 2004.
[22] S. Park and V. Sugumaran, "Designing multi-agent systems a framework and application", Expert Systems with Applications, vol. 28, no. 2, pp. 259-271, 2005.

**Zhenhua Yu** received the B.S. and M.S. degrees in Control Theory and Control Engineering from Xidian University, Xi'an, China, in 1999 and 2003, respectively. He is currently pursuing the Ph.D. degree at Xi'an Jiaotong University, Xi'an, China. His research interests include Modeling and Analyzing Multi-agent Systems, Reinforcement Learning, Petri nets, etc.

**Yuanli Cai** received the B.S., M.S. and Ph.D. degrees in Aerospace Engineering from Northwestern Polytechnical University, Xi'an, China, in 1984, 1987 and 1991, respectively. He is currently Professor at Xi'an Jiaotong University, Xi'an, China. His research interests include Multi-agent Systems, Intelligent Transport Systems, and Intelligent Guide, etc.