A Formal Approach for Modeling Software Agents Coordination

Lily Chang[1], Junhua Ding[2], Xudong He[1] and Sol M. Shatz[3]

[1] School of Computing and Information Sciences, Florida International University
Miami, FL 33199, USA
Email: {lchan003, hex}@cis.fiu.edu
[2] Department of Computer Science, East Carolina University
Greenville, NC 27858, USA
Email: dingj@ecu.edu
[3] Department of Computer Science, University of Illinois at Chicago
Chicago, IL60607, USA
Email: shatz@uic.edu

Abstract: In this paper, we propose a nested Petri net approach to model the coordination of multi-agent systems. A higher level net (called host net) defines the movements and coordination mechanism of agents, while lower level nets (called agent nets that are tokens of the host net) model the behavior of individual agents. A dynamic channel concept and the notation are introduced for modeling the communications and coordination between the host net and agent nets. We demonstrate our modeling approach through an e-market example. Our approach promotes the extensibility and flexibility of multiple agent system design through dynamic channels.

Keywords: Modeling, Multi-Agent system, Coordination, Petri nets.

1. Introduction

Multiple agent systems have become a new computing paradigm in recent years [5]. An agent is an autonomous, reactive and pro-active entity with social ability. The above characteristics are typical in many software systems that are distributed, concurrent and expected to interact with other components or exploit services and resources dynamically on the Internet. A critical issue in these systems is the coordination of multiple agents to accomplish some global tasks.

To model individual agents and the coordination of agents, a well-defined approach is needed. An approach based on formal methods is especially useful since it help analyze properties and reveal system errors at an earlier stage of development process. In this paper, we propose a formal approach for modeling the coordination of multiple agent systems using a nested predicate transition net (PrT net) [2] framework. PrT nets are a class of high level Petri nets, well suited for describing data, control, functionality, and dynamic behaviors of concurrent, asynchronous, and distributed systems. In our nested PrT net framework, individual agents are defined by agent nets, which capture individual agents' local tasks. The environment where agents reside is defined as a host net, which represents global plan with predefined policies to coordinate agent nets. A two-level net-within-net [8] structure is formed consisting of a host net at the higher level and agent nets at the lower level, in which each agent net becomes a token of the host net. We adapt the channel commands in Hoare's process algebra CSP [4] and provide a new definition of channels to model the communications and coordination between the host net and agent nets. The channel communication consists of synchronous control and unidirectional information flow. Synchronous control addresses the issue of agent task synchronization. Unidirectional information flow models the reactiveness of agent through input command and pro-activeness through output command. By allowing resource tokens and agent tokens in the higher level net, the coordination mechanism and the movements of agents within a multi-agent system can be naturally modeled.

The remainder of this paper is structured as follows. Section 2 discusses the related works compare to ours. Section 3 provides the formal definitions of our framework. Section 4 presents an e-market example for the demonstration of modeling method and steps by applying our framework. Conclusion and discussion are drawn in Section 5.

2. Related Works

In this section, we discuss some of the research works that are related to ours.

2.1 Agent System Modeling

Agent-oriented paradigm has become an active research area in recent years [9]. Since object-oriented paradigm has gained popularity among developers, most of the agent-oriented methodologies were extended from object-oriented paradigm and used informal models that did not support formal analysis. In [1] and [7], formal specification languages (temporal logic and Z respectively) were used for specifying agent models, but no behavior models were provided. In [17], G-nets, an object-based Petri nets, were further extended for agent-based systems and used to model the message processing mechanism of agents.

2.2 Multi-Agent Coordination Modeling

Since resource coordination is one of the essential tasks of a multi-agent system, various algorithms were proposed to deal with conflict controls of competing resources based on static avoidance approach ([12], [13], [14]) or negotiation approach ([15], [16]). Static avoidance approach, however, is impractical for dynamic interacting software systems since a tightly coupled global plan limits the flexibility and extensibility of the system and it is difficult and sometimes redundant to name all the possible conflicts during design time. On the other hand, negotiation approach has the flexibility and extensibility of designing the agents independently by solving the conflicts at runtime using predefined protocols, although there is a trade off for temporal efficiency. In [6], the concept of potential arc was proposed in colored Petri nets as an avoidance approach for solving conflicts among agents Potential arcs were transformed into coordinators, which included all possible alternate paths to coordinate shared resources. The approach is still considered as static since all agent plans were eventually concatenated together with the coordinators to form a global plan; therefore the system and agents are tightly coupled. Another work on agent coordination was the moderator coordination model proposed in [18]. This work focused on the agent interaction protocol and the ontology of the conversation. The moderator was separated from agent models specifically for handling the conversation among agents in an organizational view.

2.3 Agent Modeling with Net-within-Net Approach

As for modeling the movements of agents, several works ([11], [19]) also used layered net structures. These works mainly focus on the mobility of agents with regard to location changes. In [11], nested colored Petri nets were used to define the synchronized communication through a fusion of two enabled transitions where the information exchange was bi-directional. In [19], a layered predicate transition net approach was used to model the information flow between an agent net and the system net through an internal connector. The internal connectors had to be constantly updated according to the changing number of agents to maintain consistency with the system net.

3. A Nested Petri Net Framework

We define a nested Petri net framework based on predicate transition nets for modeling various aspects of multi-agent systems. Other than data and functionality, our framework supports the modeling of communications, movements of agents and real-time requirements by introducing channel expression and time expression to transition constraints.

3.1 Predicate Transition Nets

We use the PrT net definition in [3]. A PrT net is a tuple (*N, Spec, ins*), where:

(1) *N= (P, T, F)* is a net structure. *P* and *T* are finite sets of places and transitions of *N*, where $P \cap T = \emptyset, P \cup T \neq \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, which define the flow relation,

(2) *Spec* is an algebraic specification used to define the sorts and tokens of *P*, the labels of *F* and the constraints of *T* ,

(3) *ins*= (φ, L, R, M_0) is an inscription that maps net elements to their denotations in the algebraic specification *Spec*. φ is a mapping from *P* to the set of sorts; *L* is a sort-respecting mapping from *F* to the set of labels; *R* is a mapping from *T* to the set of constraints; and M_0 is an initial marking – a mapping from *P* to the set of tokens.

The dynamic semantics of a PrT net can be defined as follows:

(1) A marking of a PrT net is a mapping from *P* to sorts defined in *Spec*;

(2) An occurrence mode of *N* is a substitution $\alpha = \{x_1 \leftarrow c_1, \ldots, x_n \leftarrow c_n\}$, which instantiates typed label variables. We use $e{:}\alpha$ to denote the result of instantiating an expression *e* with α, in which *e* can be either a label expression or a constraint;

(3) Given a marking *M*, a transition $t \in T$, and an occurrence mode α, *t* is α_enabled at *M* iff the following predicate is true: $\forall p: p \in P.(\bar{L}(p,t){:}\alpha) \subseteq M(p)) \wedge R(t){:}\alpha$, where

$$\bar{L}(x,y) = \begin{cases} L(x,y) & \text{if } (x,y) \in F \\ \emptyset & \text{otherwise} \end{cases}$$

(4) If *t* is α_enabled at *M*, *t* may fire in occurrence mode α. The firing of *t* with α returns the marking *M'* defined by $M'(p) = M(p) - \bar{L}(p,t){:}\alpha \cup \bar{L}(t,p){:}\alpha$ for $p \in P$. We use $M[t/\alpha\rangle M'$ to denote the firing of *t* with occurrence α under marking *M*. As in traditional Petri nets, two enabled transitions may fire at the same time as long as they are not in conflict;

(5) For a marking *M*, the set $[M\rangle$ of markings reachable from *M* is the smallest set of markings such that $M \in [M\rangle$ and if $M' \in [M\rangle$ and $M'[t/\alpha\rangle M''$ then $M'' \in [M\rangle$, for some $t \in T$ and occurrence mode α (note: concurrent transition firings do not produce additional new reachable markings);

(6) An execution sequence $M_0T_0M_1T_1\ldots$ of *N* is either finite when the last marking is terminal (no enabled transition in the last marking) or infinite, in which case each T_i is an execution step consisting of a set of non-conflict firing transitions;

(7) The behavior of *N* is the set of all execution sequences starting from the initial marking.

3.2 Modeling Time Concepts

It is well known that a high-level Petri net model can handle time concepts adequately by representing time information as an additional element of tokens and adding time constraints as additional conjuncts to transitions [10]. Here we introduce a special variable τ, and use it exclusively as part of a transition constraint. A time expression has the following general form: $a \leq \tau \leq b$, where *a* and *b* indicate the lower and upper time bounds respectively. This time expression has the same meaning as a time interval [*a, b*] associated with a

transition in classical time Petri nets; such that an enabled transition t with time constraint $a \le \tau \le b$ is *fireable* within the relative time interval $[a, b]$ or the absolute time interval $[\theta + a, \theta + b]$. θ denotes the moment (absolute time) that transition t was enabled. Furthermore, we adopt the *strong fireability* rule, i.e., a fireable transition must fire by the time limit $\theta + b$. Transitions without timing constraints can be viewed to have fireable intervals of $[0, \infty]$.

To model timing concept, we need to modify the dynamic semantics of PrT nets. Since tokens now carry timing information, we do not explicitly add a time element into the definition of markings. Thus we only need to make the following changes to the definitions of dynamic semantics of PrT nets:

(3) Given a marking M, a transition $t \in T$, and an occurrence mode α, t is α_enabled at M iff the following predicate is true: $\forall p: p \in P.(\bar{L}(p,t):\alpha) \subseteq M(p)) \wedge R_u(t):\alpha$; where $R(t) = R_u(t) \wedge R_\tau(t)$. $R_u(t)$ is a non-timing constraint, and $R_\tau(t) = a \le \tau \le b$ is a timing constraint. We do not explicitly write the time interval $[0, \infty]$ for non-timed transitions;

(4a) An enabled transition t with timing constraint $R_\tau(t) = a \le \tau \le b$ under marking M with occurrence α is *fireable* within time interval $[\theta + a, \theta + b]$, and must fire at $\theta + b$ if it is continually enabled;

(4b) The firing of fireable transition t under M with α returns the marking M' defined by $M'(p) = M(p) - \bar{L}(p,t):\alpha \cup \bar{L}(t,p):\alpha$ for $p \in P$. We use $M[t/\alpha>M'$ to denote the firing of t with occurrence α under marking M. As in traditional Petri nets, two fireable transitions may fire at the same time as long as they are not in conflict.

3.3 Nets within Nets

Although we can define a general structure of deeply nested nets as in other works, here we elect to just define a two-level net structure that is adequate for our study and is much simpler. The lower level nets are used to model the behaviors of individual computation entity and thus PrT nets are appropriate. Since lower level nets serve as tokens of the higher level net, some care must be taken to ensure the higher level net are well defined. First, the higher level net has its own data abstraction and processing capabilities; thus we still need to have the full description power of PrT nets. Second, individual computation entities have their own behaviors and logics that cannot be described by static data. As a result, we have to treat net tokens as black boxes. Only their identities are visible and accessible in the higher level net. This treatment allows these net tokens to be grounded and thus well-defined. This treatment also respects the autonomy characteristic of the net tokens. Third, the creation and removal of a net token can be done through some boundary transitions without input places and transitions without output places respectively. We can leave the functionality (constraints) of these transitions open and view these as the responsibilities of an external environment. Fourth, we only model the logical mobility of an net token and thus do not consider the implementation details with regard to whether to use value semantics or reference semantics in passing an agent's information from one location to another.

To model synchronized communications between nets at the different levels, we extend the constraint definition to include channel expressions. We borrow the input and output commands in CSP [7] for representing channel expressions. Thus a channel expression is $n!e$ (output) or $n?x$ (input), where n is a channel name, e is an expression, and x is a variable. Channel names are the identifications of net tokens and the identification of the system net. A synchronized communication occurs when two fireable transitions at two different net levels have a *matching pair* of input and output channel expressions, i.e, a transition in the system net with identification *sys-id* contains *net-id* ! *exp* (or *net-id* ? *x*), and a fireable transition in a net token with identifier *net-id* contains *sys-id* ? *x* (or *sys-id* ! *exp*). To enforce well-definedness of communications, the channel names in agent nets must be constants; however, a channel name n in the system net can be a variable ranging over the net tokens' identifications, which is instantiated with an enabling net token identification. This allows great flexibility and concise representation of synchronized communications.

During the synchronization, a unidirectional information flow occurs such that the value in e of the output command is assigned to the variable x of the input command.

We further revise the definitions of dynamic semantics of PrT nets to capture the synchronized communications as follows:

(3) Given a marking M, a transition $t \in T$, and an occurrence mode α, t is α_enabled at M iff the following predicate is true: $\forall p: p \in P.(\bar{L}(p,t):\alpha) \subseteq M(p)) \wedge R_u(t):\alpha$;
where $R(t) = R_u(t) \wedge R_\tau(t) \wedge R_c(t)$. $R_u(t)$ is a non-timing constraint, $R_\tau(t) = a \le \tau \le b$ is a timing constraint, and $R_c(t) = n!e \mid n?x$ is a channel expression. We do not explicitly write the time interval $[0, \infty]$ for non-timed transitions;

(4a-1) An enabled transition t with a channel expression $R_c(t)$ is *ready* if a transition with a matching channel expression is also ready;

(4a-2) A ready transition t with timing constraint $R_\tau(t) = a \le \tau \le b$ under marking M with occurrence α is *fireable* within time interval $[\theta + a, \theta + b]$, and must fire at $\theta + b$ if it is continuously enabled.

Note: The effect of information flow during a synchronized communication has been reflected in the arc label expressions and thus no change to the definition of (4b) from Section 2.2 is needed.

It is obvious that this modified semantics is only meaningful in the context of a net model that consists of multiple levels. The synchronized communications only happen vertically not horizontally (i.e., no direct communication between two agent nets.) The above one-to-one communications can be extended to one-to-many (broadcasting) communications. With the input and output commands, we can naturally define the reactivity and pro-activeness of an agent.

4. Multi-Agent System Coordination Modeling

4.1 An Agent Model

Distributed and heterogeneous software systems can be naturally modeled as multi-agent systems where distributed agents are autonomous and encapsulated computation entities. Agents rely on communications to perceive external states and participate in activities such as resource sharing and subtask execution. To build an agent model, a component that associates the communications with other members in the system is mandatory (Figure 1). As a receiver, agent receives message from the environment and performs actions based on its decision logic. As a sender, agent sends request or responds to the environment. A series of communication acts is called a *conversation*, which is defined as an *Interaction Protocol* [20] that helps designers to effectively implement agent models.

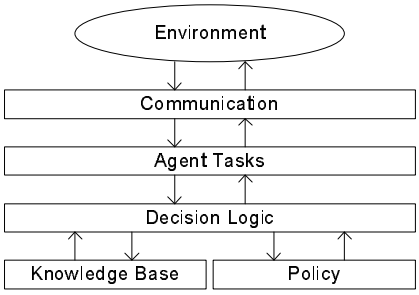

Fig. 1 Generic components of an agent model.

4.2 Coordination Model

Since agents can be geographically distributed and possibly running in heterogeneous systems, a coordination mechanism in multi-agent systems ensures over all system consistency. Although coordination strategies may vary with regard to different application domains and system objectives [21], some component responsible for coordination service is always needed. A coordination component usually has to provide up to date information about the system states and to coordinate agents based on predefined policy and system states. A communication component is also needed to facilitate the interactions between agents. (Figure 2)

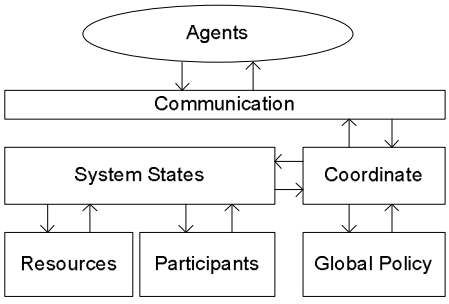

Fig. 2 Generic components of a coordination model.

Agents evolve constantly in a distributed and heterogeneous environment; separation of the coordination logic eases the complexity of agent models and allows the flexibility to modify a coordination mechanism. The coordination model also simplifies the interactions among agents. As long as agents obey the interaction protocol that complies with the coordination model, they can easily join the conversations.

4.3 A Modeling Approach

While many agent-oriented engineering methodologies have been proposed [9], few of them dealt with the coordination modeling using formal methods. In [22], the construction of a skeleton for individual autonomous agent was studied. Agents' externally visible events relevant to coordination were first identified. Based on the events, the skeleton was defined using finite state automata. The approach started from building a Dooley graph based on the conversations among agents. Then, the histories of agents in a conversation were analyzed to induce the agent skeleton. The resulting meta-model represented by finite state automata was then used to validate the specified coordination requirements, which was represented by temporal relationships. In [18], agent interaction protocols and the ontology of the conversation were investigated. The protocols were isolated from agent models and considered as resources and predefined processes that agents had to follow. A moderator encapsulated with a well-identified process was generated for each conversation between agents and a Conversation Server was defined to keep the information of all active conversations. The moderators were used as the coordination model to grant roles to agents and to control the ongoing conversation. The behavior model was specified using CoOperative Objects, a Petri net based formalism integrated with object-oriented features.

Our approach is to use the generic models described in Section 4.1 and to provide steps to build interaction models for multi-agent systems. The coordination model is conceived as a broker that matches up agents that exhibit the same interests (resources or services) based on predefined matching mechanism and published information of agents. The agents are therefore categorized into two types, one is the requestor, and the other is the provider. The requestors and providers register their interests in a public directory provided by the broker, who deals with the coordination logic that includes public directory service, collecting requests and available resources, and matching requestors and providers. Based on these assumptions, a two layered multi-agent system's hierarchy is depicted in Figure 3.

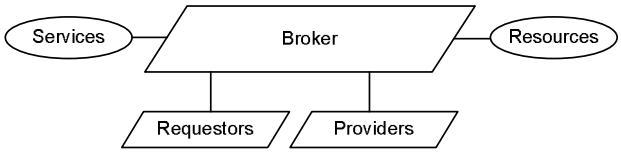

Fig. 3 Hierarchy of multi-agent coordination.

To model the coordination of a multi-agent system with layered hierarchy, we use the nested Petri net paradigm defined in Section 3. A higher level net describes the coordination behaviors with the participation of both active and passive tokens. In this case, requestors and providers are active tokens, resources and services are passive tokens. Active tokens are modeled as agent nets with their own interaction models. The communications and information flows between nets at different levels are through the channel concepts defined in Section 3.

4.4 Constructing Interaction Models

We define an *Interaction Model* to handle the coordination behaviors of a set of possible agent conversations. A C*onversation* is an execution sequence, which is initiated by a requestor and ended with a successful commitment or terminated by failure resulted from any participant that engaged in the conversation. It is obvious that an ongoing conversation will affect an agent's local behavior, thus an interaction model must comply with certain predefined interaction protocols or the conversation will not be meaningful.

We use an e-market example to demonstrate the application of the net-within-net paradigm to model the coordination of multi-agent system. Let us consider a simple conversation scenario at an e-market where seller and buyer auctioning goods. The conversation is in a format that includes *sender*: *communicative act*, *message content* and *receiver*.

Seller: request, 'sell book 30', broker
Broker: agree, 'posted book 30', seller
Buyer: request,' buy book 25', broker
Broker: inform, 'sell book 25', seller
Seller: commit, 'commit book 25', broker
Broker: inform, 'buy book 25', buyer
Buyer: commit, 'commit book 25', broker

We use higher level of abstraction to represent the message for demonstration purpose and abstract away the negotiation process about the payment transaction and shipping detail between seller and buyer, since it is not relevant to the coordination behavior. The conversation starts from a request of a seller who wants to sell book for 30 dollars, the broker agree the request and posts the information. A buyer sends a request to broker for buying book. The broker informs the seller that there is someone wants to buy book for 25 dollars and seller agrees the price. The broker informs the buyer and the deal is committed by the buyer.

First of all, there are three entities engaged in the conversation: broker, seller and buyer. The broker is served as the coordinator thus modeled as the higher level host net. The host net provides the information service of auctioning goods. The buyer and seller are participants in the activity of auctioning goods, therefore modeled as agent nets at lower level. Follow the generic model in Figure 1 and 2; each communication component is specified with transitions and places as in Figure 4.

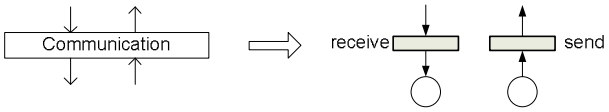

Fig. 4 Transformation of communication net.

Next step is the transformation of the actions. The agent tasks component in Figure 2 is the set of actions that agent possibly perform. Similarly, the coordinate component in Figure 3 is the set of actions that the broker possibly used to coordinate agents. In the conversation scenario, the communicative acts in *verb* represent *actions*. These actions are transformed into transitions. For example, seller has actions '*request*' and '*commit*', which imply proactive and reactive behavior respectively and should be linked to a message outgoing place. Upon messages received, seller's decision logic decides further action to be taken according to local knowledge and policy. Here, we abstract the decision logic into one transition, knowledge and policy into two places. As a result, Figure 5 shows the action net of the seller.

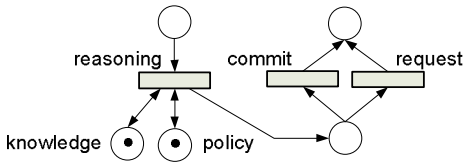

Fig. 5 Seller's action net.

Finally, let us reconsider the communication part of the model. Since the agent communication is to the upper level host net, the information is sent to external entities and not to local. Thus, the transitions for the communications must be differentiated from regular transitions to represent external communications. From Figure 4, the transition '*send*' is for output messages; we add the output channel notation to represent information flows toward outside of the model. On the other hand, transition '*receive*' is augmented with input channel notation to represent information flows from outside of the model. We concatenate the nets in Figure 4 and 5; a resulting interaction model for seller is shown in Figure 6. Note that we use dash line to represent a transition augmented with channel notations. The interaction models for broker and buyer can be built in the same manner.

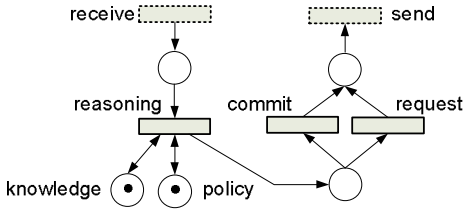

Fig. 6 Seller's abstract interaction model.

The interaction model in Figure 6 is incomplete in the sense of deriving from a simple conversation, while a set of

conversations is possible. Designer would want to list as many scenarios as possible and extract the *verb* imbedded in a conversation as the *actions* to build a more complete model. It is preferable that a set of standard *interaction protocols* is predefined for agent model designers to follow. In the following section, we give a more detail model by adding exception handling and semantics definitions.

4.5 Interaction Models of Simple Conversation Scenario

There is only one goods in this example, thus we abstract away the knowledge component, which is not relevant. The preset price of the goods is the policy of agent. In this example, seller and buyer have the same interaction model (Figure 7) and the broker's interaction model is shown in Figure 8.

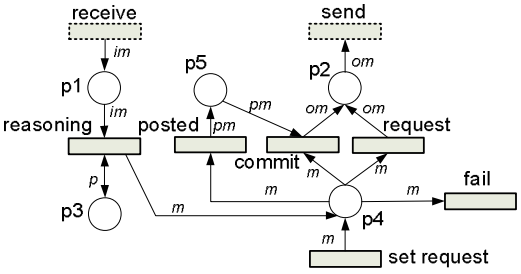

Fig. 7 Seller's and buyer's interaction model.

Token Types:

$MESSAGE = MID \times SENDER \times ACT \times PRICE \times RECEIVER$

$\varphi(p1) = \varphi(p2) = \varphi(p4) = \varphi(5) = MESSAGE$

$\varphi(p3) = PRICE$

// Type *MESSAGE* is a Cartesian product of predefined type which can represent message id, sender id, action, price and receiver id. *PRICE* is a preset price defined by an integer.

Transition Constraints:

$R(receive) = S\,?\,im$

$R(send) = S!om$

$R(reasoning) = ((im[2] = 'buy' \wedge im[3] \geq p) \vee (im[2] = 'sell' \wedge im[3] \leq p) \wedge m = im \wedge m[2] = 'commit') \vee m = im$

$R(commit) = pm[1] = m[1] \wedge m[3] = 'commit' \wedge om[1] = m[1] \wedge om[2] = m[5] \wedge om[3] = 'sell'' \wedge om[4] = m[4] \wedge om[5] = m[2]$

$R(fail) = m[2] \notin \{'request', 'commit', 'posted'\}$

$R(request) = m[2] = 'request' \wedge om = m$

$R(post) = m[2] = 'posted' \vee pm = m$

$M_0(p4) = \{\{1, a1, sell, 30, s1\}\}$

$M_0(p1) = M_0(p2) = M_3(p5) = \varnothing$

$M_0(p3) = 25$

//Transition '*receive*' and '*send*' are used to input messages and output messages through channels respectively. Transition *commit* is enabled when there is a previous request exists in place *p5* and a response message for that request is also available. If the message content can not be identified, the message is discarded through transition '*fail*'. Transition '*set request*' inputs message token from outside of the model. Transition '*reasoning*' decides what actions to be taken next based on received message. The initial markings M_0 assuming that an agent id *a1* request to sell book for 30 dollars and the message with id #1 goes to broker id *s1*, the minimum acceptance price is set to 25 dollars.

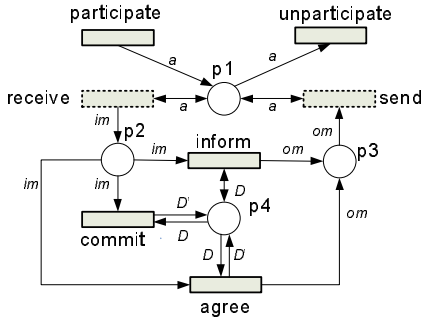

Fig. 8 Broker's interaction model.

Token Types:

$MESSAGE = MID \times SENDER \times ACT \times PRICE \times RECEIVER$

$DIRECTORY = MID \times SENDER \times ACT \times PRICE \times RECEIVER$

$\varphi(p1) = AGENTNET$

$\varphi(p2) = \varphi(p3) = MESSAGE$

$\varphi(p4) = DIRECTORY$

//Type *MESSAGE* and *DIRECTORY* are of the same type that is defined in the agent model. Type *AGENTNET* defines the active token, which is also a net.

Transition Constraints:

$R(receive) = aid\,?\,im$

$R(send) = aid\ = om[5] \wedge aid!om$

$R(agree) = (\forall d \in D.(d[1] \neq im[1] \wedge D' = D \cup im) \wedge om[1] = im[1] \wedge om[2] = im[5] \wedge om[3] = 'posted' \wedge om[4] = im[4] \wedge om[5] = im[2])$

$R(\text{inf}\ orm) = \exists d \in D.(d[1] = im[1] \wedge om[1] = d[1] \wedge om[2] = d[5] \wedge om[3] = d[3] \wedge om[4] = d[4] \wedge om[5] = d[2]$

$R(commit) = \exists d \in D.(im[1] = d[1] \wedge im[3] = 'commit' \wedge D' = D - im)$

// Transition '*receive*' and '*send*' are used to input messages and output messages through channels respectively. Transitions '*participate*' and '*unparticipate*' allow agent nets enter and out of the system. Transition '*agree*' sends a successful posted information message back to agents. Transition '*inform*' notify agent that there is a match deal. When the deal is committed, the information is deleted from the directory through transition '*commit*'.

5. Concluding Remarks

We have provided a formal net-within-net paradigm and demonstrated how to apply it to model the coordination of multi-agent systems. A formal model enables us to better understand system requirements and critical design issues and facilitate formal analysis to detect potential problems in system design at an earlier stage.

To extend our approach to model a complete multi-agent system, there are several major research issues to be solved. First, individual agent's decision logic decides the degree of autonomous and the behaviors of how an agent should react. The decision logic largely depends on the knowledge base of the agent. Thus, knowledge representation in a Petri net model is a challenge issue. Second, all entities have to speak

the same language in order to understand each other and the content of the exchanged information, which is usually domain specific. Third, message exchanges in a multi-agent system are often asynchronous, i.e. agents may not need to respond immediately or wait for responses. On the other hand, there may still be some temporal dependency among tasks. Fourth, some of the methodologies for multi-agent systems using the organization view, for instance in [23]. It is possible for an agent to be assigned different roles based on the tasks required.

Acknowledgements

We thank the anonymous reviewers for their comments to improve the presentation of the paper. Lily Chang and Xudong He's research was partially supported by NSF grants HRD-0317692 and IIP-0534428. Sol M. Shatz's research was partially supported by U.S. Army Research Office under grant number W911NF-05-1-0573.

References

[1] H. Barringer, M. Fisher, D. Gabbay, G. Gough and R. Owens, METATEM: A Framework for Programming in Temporal Logic, Proceedings on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness. LNCS, Vol. 430, pp.94-129.

[2] H. J. Genrich, Predicate/Transition nets. Advances in Petri Nets 1986, pp. 207–247.

[3] X. He and Y. Deng, A Framework for Developing and Analyzing Software Architecture Specifications in SAM. The Computer Journal, Vol. 45, No. 1, 2002, pp. 111–128.

[4] C. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.

[5] N. Jennings and M. Wooldridge, Agent-Oriented Software Engineering. Proceedings of the 9th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, 2000.

[6] J. Lian and S. M Shatz, Potential arc: A Modeling Mechanism for Conflict Control in Multi-agent Systems. Proceedings of the 4th Symposium on Design, Analysis, and Simulation of Distributed Systems (DASD-06) 2006, pp. 467–474.

[7] M. Luck, N. Griffiths and M d'Inverno, From Agent Theory to Agent Construction: A Case Study. Proceedings of The ECAI'96 Workshop on Agent Theories, Architectures, and Languages: Intelligent Agents III 1997, Vol. 1193, Springer-Verlag: Heidelberg, Germany, pp. 49–64.

[8] R. Valk, Petri nets As Token Objects: An Introduction to Elementary Object Nets. Application and Theory of Petri Nets, 1998, pp. 1–25.

[9] M. Wooldridge and P. Ciancarini, Agent-oriented Software Engineering, The State of The Art. Lecture Notes in Computer Science, Vol. 1957, 2001, pp. 1–28.

[10] C. Ghezzi, D. Madrioli, S. Morasca, M. Pezze, A Unified High Level Petri Net Formalism for Time Critical Systems, IEEE Transactions on Software Engineering, Vol. 17, No. 2,(1991) 160-172.

[11] M. Kohler,. D. Moldt, H. Rolke, Modeling Mobility and Mobile Agents Using Nets Within Nets, Proc. of International Conf. on Application and Theory of Petri Nets, LNCS vol. 2679 (2003), 121-139.

[12] K. S. Barber, T. H. Liu and S. Ramaswamy, Conflict Detection during Plan Integration for Multi-agent Systems. IEEE Transactions on Systems, Man, and Cybernetics, Part B, Vol. 31, No. 4, 2001, pp. 616–628.

[13] S. Resmerita and M. Heymann, Conflict Resolution in Multi-agent Systems. Proceeding of 42nd IEEE Conference on Decision and Control, Vol. 3, 2003, pp. 2537–2542.

[14] P. Moraitis and A. Tsoukias, A Multi-criteria Approach for Distributed Planning and Conflict Resolution for Multi-agent Systems. Proceeding of International Conference on Multi-Agent Systems, 1996, pp. 212 – 219.

[15] I. Loutchko and F. Teuteberg, An Agent-based Electronic Job Marketplace: Conceptual Foundations and Fuzzyman Prototype. International Journal of Computer Systems Science and Engineering, Vol. 20, 2005, pp. 95–109.

[16] J. Sillince, Multi-agent Conflict Resolution: A Computational Framework for an Intelligent Argumentation Program. Knowledge Based Systems, Vol. 7, 1994.

[17] H. Xu and S. M. Shatz, A Framework for Model-based Design of Agent-oriented Software. IEEE Transactions on Software Engineering, 2003, pp. 15–30.

[18] C. Hanachi and C. Blanc, "Protocol Moderators as Active Middle-Agents in Multi-Agent Systems," Autonomous Agents and Multi-Agent Systems, Vol. 8, No. 2, pp. 131-164, 2004.

[19] D. Xu, J. Yin, Y. Deng and J. Ding: A Formal Architecture Model for Logical Agent Mobility. IEEE Transactions on Software Engineering. Vol. 29, No. 1, pp. 31-45, Jan. 2003

[20] FIPA, Interaction Protocol Library Specification, 2000.

[21] D. Deugo, M. Weiss and E. Kendall: Reusable Patterns for Agent Coordination, Coordination of Internet agents: models, technologies, and applications. 2001, pp. 347-368.

[22] M. P. Singh, Synthesizing Coordination Requirements for Heterogeneous Autonomous Agents, Autonomous Agents and Multi-Agent Systems, Vol.3, No. 2. pp. 107-132, 2000.

[23] F. Zambonelli, N. R. Jennings and M. Wooldridge: Developing Multiagent Systems: The Gaia Methodology. ACM Transactions on Software Engineering and Methodology, Vol. 12, No.3, pp.417 – 470, 2003.