An Agent-based Petri Net Model with Application to Seller/Buyer Design in Electronic Commerce

Haiping Xu and Sol M. Shatz
Department of Electrical Engineering and Computer Science
The University of Illinois at Chicago
Chicago, IL 60607
Email: {hxu1, shatz}@eecs.uic.edu

Abstract

Agents are becoming one of the most important topics in distributed and autonomous decentralized systems (ADS), and there are increasing attempts to use agent technologies to develop software systems in electronic commerce. Such systems are complex and there is a pressing need for system modeling techniques to support reliable, maintainable and extensible design. G-Nets are a type of Petri net defined to support modeling of a system as a set of independent and loosely-coupled modules. In this paper, we first introduce an extension of G-Net, agent-based G-Net, as a generic model for agent design. Then new communication mechanisms are introduced to support asynchronous message passing among agents. To illustrate our formal modeling technique is effective for agent modeling in electronic commerce, a price-negotiation protocol example between buyers and sellers is provided. Finally, by analyzing an ordinary Petri net reduced from our agent-based G-Net models, we conclude that our agent-based G-Net models are L3-live, concurrent and effective for agent communications.

1. Introduction

Agents are becoming one of the most important topics in distributed and autonomous decentralized systems (ADS). With the increasing importance of electronic commerce across the Internet, the need for agents to support both customers and suppliers in buying and selling goods or services is growing rapidly. Most of the technologies supporting today’s agent-based electronic commerce systems stem from distributed artificial intelligence (DAI) research [1][2]. Applications developed with multi-agent systems (MAS) in electronic commerce are examples of such efforts. A multi-agent system (MAS) is a concurrent system based on the notion of autonomous, reactive, and internally-motivated agents in a decentralized environment. The increasing interest in MAS research is due to the significant advantages inherent in such systems, including their ability to solve problems that may be too large for a centralized single agent, to provide enhanced speed and reliability, and to tolerate uncertain data and knowledge [2]. The notable systems developed with MAS in electronic commerce are Kasbah [3] and MAGMA [4]. Kasbah is meant to represent a marketplace where Kasbah agents, acting on behalf of their owners, can filter through ads and find those that their users might be interested in. The agents then proceed to negotiate to buy and sell items. MAGMA moves the marketplace metaphor to an open marketplace involving agents buying/selling physical goods, investments and forming competitive/cooperative alliances. These agents negotiate with each other through a global blackboard.

Although there are many efforts on developing multi-agent systems, there is a lack of research on formal specification and design of such systems [5][6]. As the multi-agent technology begins to emerge as a viable solution for large-scale industrial and commercial applications, there is an increasing need to ensure that the systems being developed are robust, reliable and fit for purpose. Previous work [7] on formal modeling agent systems includes: (1) using formal languages, such as Z, to provide a framework for describing a system at different levels of abstractions; (2) using temporal modal logic to allow the dynamic aspects of agents; and (3) designing formal languages, such as DESIRE, for multi-agent specification.

In this paper, we extend a formal model, called a G-Net (a form of Petri net [9]), to support modeling of agents in multi-agent systems. The advantage of our formal mechanism is that it provides a clean interface between agents with both asynchronous and synchronous communication abilities and supports formal reasoning.

---

1 This material is based upon work supported by the U.S. Army Research Office under grant number DAAD19-99-1-0350, and the U.S. National Science Foundation under grant number CCR-9988168.
for our agent design. Furthermore, our formal mechanism is based on Petri net formalism that is a mature formal model with existing theory and tool support. The rest of this paper is organized as follows. Section 2 briefly introduces the standard G-Net model, and discusses the general structure of the proposed agent-based G-Net model. Section 3 provides a seller/buyer example in electronic commerce. We show how a seller and buyer agent can be designed by using our agent-based G-Net model. Section 4 first reduces our seller and buyer agent-based G-Net models to an ordinary Petri net. Then our net model is proven to be L3-live and unbounded. In addition, we show by examples that agent communication protocols can be correctly traced in our net model. Section 5 provides a brief conclusion and a summary of our future work.

2. Agent-based G-Net Model

2.1 The Standard G-Net Model

A widely accepted software engineering principle is that a system should be composed of a set of independent modules, where each module hides the internal details of its processing activities and modules communicate through well-defined interfaces. The G-Net model provides strong support for this principle [10]. G-Nets are an object-based extension of Petri nets. We assume that the reader has a basic understanding of Petri nets [9], so we begin with some introduction to the G-Net model. A G-Net system is composed of a number of G-Nets, each of them representing a self-contained module or object. A G-Net is composed of two parts: a special place called Generic Switch Place (GSP) and an Internal Structure (IS). The GSP provides the abstraction of the module, and serves as the only interface between the G-Net and other modules. The IS, a modified Petri net, represents the detailed design of the module. An example of G-Nets is shown in Figure 1. Here the G-Net models represent two objects – the Buyer and the Seller. The generic switch places are represented by GSP(Buyer) and GSP(Seller) enclosed by ellipses, and the internal structures of these models are represented by round-cornered rectangles that contain the detailed design of four methods: buyGoods(), askPrice(), returnPrice() and sellGoods(). The functionality of these methods are defined as follows: buyGoods() invokes the method sellGoods() defined in G-Net Seller to buy some goods; askPrice() invokes the method returnPrice() defined in G-Net Seller to get the price of some goods; returnPrice() is defined in G-Net Seller to calculate the latest price for some goods; and sellGoods() is defined in G-Net Seller to handle things like waiting for the payment, shipping the goods and generating the invoice. A GSP of a G-Net G contains a set of methods G.MS specifying the services or interfaces provided by the module, and a set of attributes G.AS as attributes or state variables (we do not show both of them in Figure 1). In G.IS, Petri net places represent primitives, while transitions, together with arcs, represent connections or relations among those primitives. These primitives may be actions or method calls, represented by special places called Instantiated Switch Places (ISP). A primitive becomes enabled if it receives a token, and an enabled primitive can be executed. Given a G-Net G, an ISP of G is a 2-tuple (G’.Nid, mtd), where G’ could be the same G-Net G or some other G-Net, Nid is a unique identifier of G-Net G’, and mtd ∈ G’.MS. Each ISP(G’.Nid, mtd) denotes a method call mtd() to G-Net G’. An example ISP (denoted as an ellipse in Figure 1) is shown in the method askPrice() defined in G-Net Buyer, where the method askPrice() makes a method call returnPrice() to the G-Net Seller to query about the price for some goods (we have omitted all parameters for simplicity).

![Figure 1 G-Net Model of Buyer and Seller Objects](image)

From the above description, we can see that a G-Net model essentially represents a module or an object rather than an abstraction of a set of similar objects. In a recent paper [11], we have extended the G-Net model to support class modeling. The idea of this extension is to generate a unique object identifier G.Oid and initialize the state variables in G.AS when a G-Net object is instantiated from a G-Net G. An ISP method invocation is no longer represented as the 2-tuple (G’.Nid, mtd), instead it is the 2-tuple (G’.Oid, mtd), where different object identifiers could be associated with the same G-Net class model.

The token movement in a G-Net object is similar to that of original G-Nets [10]. A token tkn is a triple (seq, sc, msg), where seq is the propagation sequence of the token, sc ∈ {before, after} is the status color of the token and msg is a triple (mtd_name, para_list, result). For ordinary places, tokens are removed from input places and deposited into output places by firing transitions. However, for the special place ISP, whenever a method...
call is made to a G-Net object, the token in the ISP place is processed (by attaching information for the method call) and removed, and an identical token is deposited into the GSP place of the called G-Net object. Through the GSP of the called G-Net object, the token is then dispatched into an entry place of the appropriate called method. After the method call, the token will reach a return place (denoted by double circles) with the result attached to the token. As soon as this happens, the token will return to the ISP place of the caller and the information related to this completed method call will be detached.

We call a G-Net model that supports class modeling as a standard G-Net model. Notice that the example we provide in Figure 1 follows the Client-Server paradigm, in which a Seller object works as a server and a Buyer object is a client. Although the standard G-Net model works well in object-based design, it is not sufficient in agent-based design for the following reasons. First, agents in multi-agent systems are usually developed by different vendors independently, and those agents will be widely distributed across large-scale networks such as the Internet. To make it possible for those agents to communicate with each other, it is essential for them to have a common communication language and to follow common protocols. However, the standard G-Net model does not directly support protocol-based language communication between agents. Second, the underlying agent communication model is usually asynchronous, and an agent may decide whether to perform actions requested by some other agents. The standard G-Net model does not directly support asynchronous message passing and decision-making, but only supports synchronous method invocations in the form of ISP places. Third, agents are commonly designed to determine their behavior based on individual goals and their knowledge. They may autonomously and spontaneously initiate internal or external behavior at any time. Standard G-Net models can only directly support a predefined flow of control.

2.2 Extending G-Nets for Agent Modeling

To support agent-based design, we need to extend a G-Net to support modeling an agent class. The idea is similar to extending a G-Net to support class modeling [11]. When we instantiate an agent-based G-Net (an agent class model), an agent identifier is generated and the mental state of the resulting agent object (an active object [7]) is initialized. In addition, at the class level, three special modules are introduced to make an agent autonomous and internally-motivated, namely the Goal module, the Knowledge-base module and the Planner module. The outline of an agent-based G-Net model is shown in Figure 2. A Goal module is an abstraction of a goal model [8], which describes the goals that an agent may possibly adopt. A Knowledge-base module is an abstraction of a belief model [8], which describes the information about the environment and internal state that an agent of that class may hold. A Planner module can be viewed as the heart of an agent that makes a plan to achieve some committed goals. For instance, in the Planner module, an agent may decide to ignore an incoming message, start a new conversation, or continue with a conversation, which may be initiated by some other agent or the agent itself.

The internal structure (IS) of an agent-based G-Net consists of four sections, namely the incoming messages, outgoing messages, public services and private utilities. Message Processing Units (MPU) defined in the incoming/outgoing messages section are used to process incoming/outgoing messages, and it may use ISP function calls to methods defined in its private utilities section. The public services section makes an agent able to work as a server. Other agents may use the ISP function call mechanism to invoke these services synchronously. We keep this synchronous communication mechanism for agents because we view an agent as an active object with further characteristics like being autonomous, reactive and internally-motivated. The private utilities section is similar to the public services section but with the difference that private utility functions can only be called by the agent itself.

![Figure 2 A General Agent-based G-Net Model](image)

Notes: G' = mThn.body.receiver as defined later in this section

Although both objects (passive object) and agents use message-passing to communicate with each other, message-passing for objects is a unique form of method invocation, while agents distinguish different types of messages and model these messages frequently as speech acts and use complex protocols to negotiate. In addition,

---

1 We view the abstract of a set of similar agents as an agent class, and we call an instance of an agent class an agent or an agent object.
agents analyze these messages and can decide whether to execute the requested action [7]. As we stated before, most of the agent communications are asynchronous message passing. Since asynchronous message passing is more fundamental than synchronous message passing, it is useful for us to introduce a new mechanism, called Message-passing Switch Place (MSP), to support asynchronous message passing directly. When a token reaches an MSP place (we represent it as an ellipse in Figure 2), the token is removed and deposited into the GSP place of the called agent. Unlike the ISP mechanism, the calling agent does not wait for the token to return before it can continue to execute its next step. Note that we have extended G-Nets to allow the use of the keyword self to refer to the agent object itself.

Figure 3 A Template of Planner Module

A template of the Planner module is shown in Figure 3. The modules Goal and Knowledge-base are represented as two special places, each of which contains a token that represents a set of goals or a set of beliefs. The Planner module is goal-driven because the transition start_a_conversion may fire whenever an attempt is made to achieve a committed goal. In addition, the Planner module is also message-triggered because certain actions may initiate whenever a message arrives (either from some other agent or the agent itself). If the message comes from some other agent, it will be dispatched to a MPU defined in the incoming messages section of the agent-based G-Net’s internal structure. After the message is processed, the MPU will transfer the processed message as a token to the GSP place of the agent itself. This is done by sending a message MSP(self) to the agent itself. Upon arrival of this internal message, the transition internal may fire, and the next action will be determined based on the agent’s current mental state. Alternatively, the next action could be to ignore the message or to continue with the current conversation. In either case, a token will be deposited in place update_goal/kb, and the transition update may fire. As a consequence, the agent’s mental state may change. If the next action is to continue the conversation, the tag of the token will be changed from internal to external, and the token will be deposited in place dispatch_outgoing_message. In this case, the corresponding MPU will be called before the message is sent to some other agent by using the MSP mechanism. In addition, an agent may also work as a server by providing a set of public services and allowing other agents to make synchronous method calls to it. Whenever there is a service request, the token deposited in the GSP place will be dispatched to a method in the public services section.

As a result of this extension, the structure of tokens in the agent-based G-Net model should be redefined. Essentially there are three types of tokens, namely the message token mTkn, the goal token gTkn and the knowledge token kTkn. One way to construct the gTkn and kTkn is to make them linked lists. In other words, a gTkn represents a list of goals and a kTkn represents a list of facts. Since these two tokens confine themselves in places in their corresponding modules of our agent-based G-Net model, we do not describe them further in this paper.

An mTkn is a 2-tuple (tag, body), where tag ∈ {internal, external, public, private} and body is a variant, which is determined by the tag. According to the tag, the token deposited in a GSP place will be dispatched into an entry place of a MPU or a method defined in the internal structure of the agent-based G-Net. Then the body of the token mTkn will be interpreted differently. More specifically, we define the mTkn body as follows:

```c
if (mTkn.tag ∈ {internal, external})
then mTkn.body = struct {
    int sender; // message sender identifier
    int receiver; // message receiver identifier
    string protocol_type; // protocol type
    string message_name; // message name
    string content; // message content
} ; // mTkn.body is as defined in Section 2.1
else mTkn.body = {seq, sc, msg};
```

We now provide a few key definitions for our agent-based G-Net models.

**Definition 2.1 Agent-based G-Net**

An agent-based G-Net is a 5-tuple AG = (GSP, GL, KB, PL, IS), where GSP is a Generic Switch Place providing an abstract for the agent-based G-Net, GL is a Goal module, KB is a Knowledge-base module, PL is a Planner module and IS is an internal structure of AG.

**Definition 2.2 Planner Module**

A Planner module of an agent-based G-Net AG is a colored sub-net defined as a 5-tuple (IGS, IGO, IKB, IIS,
DMU), where IGS, IGO, IKB and IIS are interfaces with GSP place, Goal module, Knowledge-base module and internal structure of AG, respectively. DMU is a decision-making unit with the functionality of dispatching messages, determining the next action, starting a new conversation and updating the Goal and Knowledge-base module of AG.

Definition 2.3 Internal Structure (IS)
An internal structure (IS) of an agent-based G-Net AG is a 4-tuple \((IM, OM, PS, PU)\), where IM/OM is the incoming/outgoing messages section, which defines a set of Message Processing Units (MPUs); PS/PU is the public services/private utilities section, which defines a set of Methods.

Definition 2.4 Message Processing Unit (MPU)
A message processing unit (MPU) is a triple \((P, T, A)\), where \(P\) is a set of places with three special places called entry place, ISP place and MSP place. Each MPU can have only one entry place and one MSP place, but it may contain multiple ISP places. \(T\) is a set of transitions, and each transition can be associated with a set of guards. \(A\) is a set of arcs defined as: \(((P\{-MSP\}) \times T) \cup ((T \times (P\{-entry\}))).

Definition 2.5 Method
A method is a triple \((P, T, A)\), where \(P\) is a set of places with three special places called entry place, ISP place and return place. Each method can have only one entry place and one return place, but it may contain multiple ISP places. \(T\) is a set of transitions, and each transition can be associated with a set of guards. \(A\) is a set of arcs defined as: \(((P\{-return\}) \times T) \cup ((T \times (P\{-entry\}))).

3. Seller and Buyer Design

To illustrate how to design a seller/buyer agent by using our agent-based G-Net model, we use an example derived from [12]. Figure 4 (a) is a modified example of an FIPA contract net protocol adapted from [12], which depicts a template of protocol expressed as a UML sequence diagram for a price-negotiation protocol between a buyer and a seller. To correctly draw the sequence diagram for this template, we need to introduce two new notations, i.e., the end of protocol operation “*” and the iteration of communicative acts operation “+”. Examples of using these two notations are as follows. In Figure 4 (a), we put a mark of “*” in front of the message name “refuse” to indicate that this message ends the protocol. In the same figure, a mark “*” is put on the right corner of the narrow rectangle for the message “propose” to indicate that the communicative actions in this section can be repeated zero or more times.

When a conversation based on this contract net protocol begins, the buyer agent sends a request for price to a seller agent. The seller agent can then choose to respond to the buyer agent by refusing to provide price or submitting a proposal. Here the “x” in the decision diamond indicates an exclusive-or decision. If a proposal is offered, the buyer agent has a choice of either accepting or rejecting the proposal. If a seller agent receives a reject-proposal message, it may send the buyer agent a new proposal or replies to the buyer agent with a confirmation message. If the seller agent receives an accept-proposal message, it will simply send a confirmation message to the buyer agent. Whenever a confirmation message is sent, the protocol ends. Figure 4 (b) and 4 (c) shows two actual cases of this protocol template. In Figure 4 (b), the seller agent’s proposal is accepted by the buyer agent in one round; while Figure 4 (c) shows the case that the proposal is accepted by the buyer agent in the second round.

Based on the communicative acts (e.g., request-price, propose etc.) needed for this contract net protocol, we may design the buyer agent as in Figure 5. In Figure 5, the Goal and Knowledge-base modules remain as abstract units and can be refined in further detailed design. The
Verifying Agent-based G-Net models

One of the advantages of building a formal model for agents in agent-based design is to ensure a correct design that meets certain specifications. A correct design of agents at least has the following properties:

- **L3-live:** any communicative act can be performed as many times as needed.
- **Concurrent:** a number of conversations among agents can happen at the same time.
- **Effective:** an agent communication protocol can be correctly traced in the agent models.

To verify the correctness of agent-based G-Net models for seller/buyer agents with respect to the above properties, we first reduce our agent-based G-Net models to an ordinary Petri net as follows: (1) simplify the Goal module and Knowledge-base module as ordinary places with ordinary tokens; (2) omit the public services and private utilities sections; (3) simplify mtkn tokens as ordinary tokens; (4) use net reduction to simplify the Petri net corresponding to an MPU/Method as a single place; and (5) use the close world assumption and make our system only contains two agents, i.e., a buyer agent and a seller agent.

The resulting ordinary Petri net is illustrated in Figure 6. To verify the correctness of our agent-based G-Net model for agent communication, we utilize some key definitions and theorems as adapted from [9].

**Definition 4.1 Incidence Matrix**
For a Petri net N with n transitions and m places, the incidence matrix A = [aij] is an n x m matrix of integers and its typical entry is given by

\[
 a_{ij} = a^{+}_{ij} - a^{-}_{ij}
\]

where \( a^{+}_{ij} = w(i,j) \) is the weight of the arc from transition \( i \) to output place \( j \) and \( a^{-}_{ij} = w(j,i) \) is the weight of the arc from input place \( j \) to transition \( i \).

**Definition 4.2 Firing Count Vector**
For some sequence of transition firings in a Petri net N, a firing count vector \( x \) is defined as an n-vector of nonnegative integers, where the \( i \)-th entry of \( x \) denotes the number of times that transition \( i \) must fire in that firing sequence.

**Definition 4.3 T-invariant**
For a Petri net N, an n-vector \( x \) of integers (\( x \neq 0 \)) is called a T-invariant if \( x \) is an integer solution of homogeneous equation \( A x = 0 \), where A is the incidence matrix of Petri net N.

**Definition 4.4 Support and minimal-support T-invariant**
The set of transitions corresponding to non-zero entries in a T-invariant \( x \geq 0 \) is called the support of a T-invariant and is denoted as \( \|x\| \). A support is said to be minimal if no proper non-empty subset of the support is also a support. Given a minimal support of a T-invariant, there is a unique minimal T-invariant corresponding to the minimal support. Such a T-invariant is called the minimal-support T-invariant.

**Definition 4.5 L3-live Petri net**
A Petri net N with initial marking \( M_0 \), denoted as \( (N, M_0) \), is said to be L3-live if for every transition \( t \) in the net, \( t \) appears infinitely often in some firing sequence \( L(N, M_0) \), where \( L(N, M_0) \) is the set of all possible firing sequences from \( M_0 \) in the net \( (N, M_0) \).

**Theorem 4.1** An n-vector \( x \) is a T-invariant of a Petri net \( N \) iff there exists a marking \( M_0 \) and a firing sequence \( \sigma \) that reproduces the marking \( M_0 \), and \( x \) defines the firing count vector for \( \sigma \).

**Theorem 4.2** A Petri net N with initial marking \( M_0 \) is L3-live if there exists a set of minimal-support T-invariants

---

![Figure 6 A Transformed Model of Buyer and Seller Agents](image-url)
that covers all the transitions in the net, and for each minimal-support T-invariant there exists a firing sequence that reproduces the initial marking $M_0$.

**Proof:** Let $T$ be the set of transitions in Petri net $(N, M_0)$. $\Gamma$ be the set of minimal-support T-invariants that covers all the transitions in $T$. From the given condition, we know that for $\forall t \in T$, $\exists \chi \in \Gamma$, which covers transition $t$. Since for the minimal-support T-invariant $\chi$, there exists a finite firing sequence $\rho$ that reproduces the initial marking $M_0$, $t$ appears in $\rho$. Let the infinite firing sequence $\sigma = \rho \cdot \rho \cdot \rho \cdot \rho \cdots$, where “$\cdot$” is the concatenation operator between finite sequences, $t$ appears in $\sigma$ infinitely often.

By definition 4.5, Petri net $(N, M_0)$ is $L_3$-live.

The incidence matrix $A$ of the Petri net in Figure 6 is listed in Table 1. By using Definition 4.1 and 4.4, we can calculate a set of minimal-support T-invariants as follows:

$$x_1 = [\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}]$$

Since the above minimal-support T-invariants cover all the transitions in the net, and for each minimal-support T-invariant, there exists a firing sequence that reproduces the initial marking $M_0$, from Theorem 4.2, we conclude that our Petri net model with initial marking $M_0$ is $L_3$-live, i.e., for any transition $t$ in our net model, we can find an infinite firing sequence that $t$ appears infinitely often. Consequently, any communicative act can be performed as many times as needed.\(^1\)

In Figure 6, it is obvious to see that our net model is unbounded. This is because transitions $t3$ and $t21$ can fire as many times as needed. This behavior shows that both the buyer and seller agent may initiate conversations autonomously and concurrently (as we stated before, the initiation of a new conversation is goal driven). There can be as many conversations as necessary between the buyer agent and the seller agent. As an example, a buyer agent may request prices of several goods from a seller agent at the same time, and several buyer agents may request price of the same goods from a seller agent concurrently.

In addition, we may trace an agent communication protocol $p$ in our net model with a firing sequence $\sigma$. For a protocol $p$, a corresponding firing sequence $\sigma$ in our net model has more semantics than the protocol itself because when we actually execute a protocol in our net, we need to do additional work, such as updating the goal or knowledge base after each communicative act. Since a marking $M$ that is reachable from $M_0$, but $M \neq M_0$, represents that there are still some ongoing conversations in the net, to correctly trace a protocol $p$ in our net model, it is essential for us to find a firing sequence $\sigma$ that reproduces the initial marking $M_0$. In other words, we need to make sure that there will be no residual tokens for a conversation left in the net after that conversation completes. In this case, we say that the protocol $p$ can be effectively traced as a firing sequence $\sigma$ in our net model.

To show that a protocol $p$ can be effectively traced, we use the contract net protocol examples in Figure 4 (b) and Figure 4 (c). These two protocols can be traced in our net model as follows:

$\sigma_0 = \langle t3, t13, t16, t19, t22, t27, t20, t26, t30, t31, t34, t14, t9, t2, t12, t14, t17, t19, t23, t28, t20, t26, t30, t33, t36, t11, t16, t11, t2, t17, t12\rangle$

$\sigma_c = \langle t3, t13, t16, t19, t22, t27, t20, t26, t30, t31, t34, t14, t9, t2, t12, t15, t18, t19, t24, t29, t20, t26, t30, t31, t34, t14, t9, t2, t12, t14, t17, t19, t23, t28, t20, t26, t30, t33, t36, t11, t16, t11, t2, t17, t12\rangle$

By Definition 4.2, we calculate their corresponding firing count vectors $x_b$ and $x_c$ as follows:

\(^1\) One of the limitations for invariant approach is that it is not sufficient to prove a Petri net is $L_4$-live or $L_5$-live, i.e., it is possible to ultimately fire any transition of the net from any marking $M$ that is reachable from $M_0$.\)
By Definition 4.3, it is easy to verify that both $x_b$ and $x_c$ are T-invariants because both of the equations $A'x_b = 0$ and $A'x_c = 0$ are satisfied. This shows that both firing sequences $q_b$ and $q_c$ can reproduce the initial marking $M_0$. In other words, we prove that both protocols in Figure 4(b) and 4(c) can be effectively traced in our agent-based model.

5. Conclusion and Future Work

One of the most rapidly growing areas of interest for Internet technology is that of electronic commerce. Consumers are looking for suppliers selling products and services on the Internet, while suppliers are looking for buyers to increase their market share. For convenience and efficiency, we believe that ADS in a form of multi-agent systems (MAS) is an effective way to automate the time consuming process of looking for buyers or seller and negotiate in order to obtain the best deal. Although there are several implementations of agent-based electronic marketplaces available [4][5], formal frame works for such systems are few. It is an increasing need to provide formal methods in multi-agent systems specification and design to ensure robust and reliable products.

In this paper, we introduced an agent-based G-Net model for buyer and seller agent modeling in electronic commerce. Using this model, sellers and buyers can be modeled as agents with the characteristics of autonomous, reactive and internally-motivated. Agent-based G-Net models also provide a clean interface between agents, and agents may communicate with each other by using contract net protocols. Furthermore, these models are based on the Petri net formalism, which is a mature formal model in terms of both existing theory and tool support. An example of price-negotiation protocol between buyers and sellers is used to illustrate our basic idea, and we prove that the agent communication mechanism in our net model meets the requirements of L3-live, concurrent and effective properties.

For our future work, we will try to refine the Goal, Knowledge-base modules, and the decision-making mechanisms in Planner module, and try to use this formal model to prove the correctness of contract net protocols. Furthermore, to capture more semantics of our agent-based G-net models, and to obtain performance metrics of multi-agent systems, we will translate our net models into colored Petri nets, and use existing Petri net tools, such as Design/CPN, to do the analysis. We will also look into issue like deadlock avoidance and state exploration problems in the agent design and verification processes.

6. References


