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

Outline

- Introduction: Related work and our approach.
- Part 1: G-net background and the standard G-nets.
- Part 2: Design of agent-based G-net model.
- Part 3: Modeling agent-oriented software.
- Part 4: Future research plans.

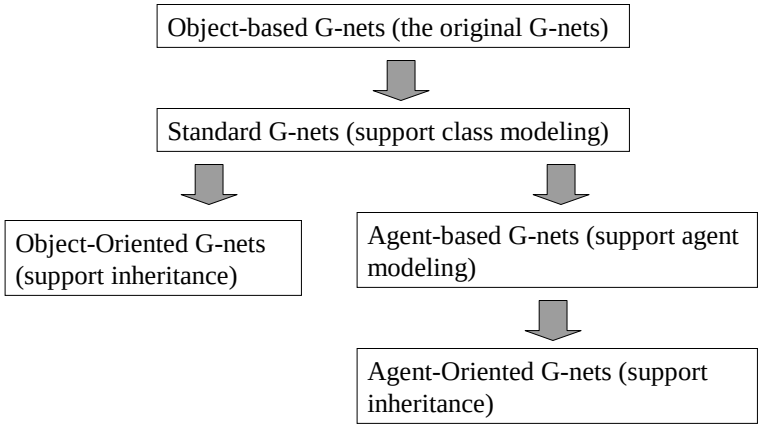
Agent-Oriented Approaches and Formal Methods

- The agents can be considered as *active* objects, i.e., objects with a mental state.
- However, object-oriented methodologies do not address the following aspects:
  - asynchronous message-passing mechanism
  - mental state: plan, goal and knowledge
  - autonomous behavior
- Agent-oriented approaches: provide guidelines for agent specification and design.
  - KGR methodologies: based on BDI model.
  - Gaia methodologies: based on role modeling.

Agent-Oriented Approaches and Formal Methods (continue)

- Very little work on how to *formally* specify and design agents.
  - DESIRE (DEsign and Specification of Interacting REasoning components) provides a compositional framework for modeling agents.
  - dMARS (distributed MultiAgent Reasoning System) is based on Procedure Reasoning System (PRS) and supports formal reasoning.
  - Agent models based on Petri nets, such as [Moldt and Wienberg 1997] [Merseguer et al. 2000] [Xu and Deng 2000]
- However, they do not explicitly model agent communication, and they do not address the issue of inheritance.
- Unlike the previous work, our proposed agent models:
  - support protocol-based agent interaction/communication.
  - support reuse of functional units of our agent class models.

Our Incremental Approach



Advantages of Our Approach

- Based on the Petri net formalism, which is a mature formal model in terms of both existing theory and tool support.
- Support reuse of object or agent designs.
- Provide a nature way for object-oriented software designers to design agent systems.
- Support net-based modeling and analysis.
  - provide a clean interface among objects or agents.
  - support net–based behavior analysis and verification.
  - may unify the object-oriented G-nets and agent-oriented G-nets to model complex software systems.

Part 1: G-net Background and the Standard G-nets

- Defined to support modeling of systems as a set of independent and loosely-coupled modules.
- Provide support for incremental design and successive modification.
- Are not fully object-oriented due to a lack of support for inheritance.

An Example

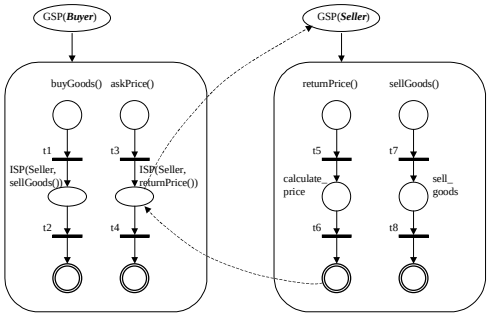


Figure 1. G-net models of buyer and seller objects

Extending G-nets to Support Class Modeling

- Motivation: to support abstraction and inheritance.
- Interpret a G-net as a model of class.
- Instantiate a G-net *G*:
  - generates a unique object identifier *G.Oid*
  - initializes the state variables defined in *G*
  - *ISP* method invocation becomes 2-tuple (G'.Oid, mtd)

Formal Definitions of the *Standard* G-net Model

Definition 2.1 *G-net system*
A *G-net system (GNS)* is a triple GNS = (INS, GC, GO), where INS is a set of initialization statements used to instantiate G-nets as G-net objects; GC is a set of G-nets which are used to define classes; and GO is a set of G-net objects which are instances of G-nets.

Definition 2.2 *G-net*
A *G-net* is a 2-tuple G = (GSP, IS), where GSP is a *Generic Switch Place (GSP)* providing an abstraction for the G-net; and IS is the *Internal Structure*, which is a set of modified Pr/T nets. A G-net is an abstract of a set of similarly G-net objects, and it can be used to model a class.

Definition 2.3 *G-net object*
A *G-net object* is an instantiated G-net with a unique object identifier. It can be represented as (G, OID, ST), where G is a G-net, OID is the unique object identifier and ST is the state of the object.

Definition 2.4 *Generic Switching Place (GSP)*
A *Generic Switch Place (GSP)* is a triple of (NID, MS, AS), where NID is a unique identifier (class identifier) of a G-net G; MS is a set of methods defined as the interface of G-net G; and AS is a set of attributes defined as a set of instance variables.

Definition 2.5 *Internal Structure (IS)*
The *internal structure* of G-net *G* (representing a class), *G.IS*, is a net structure, i.e., a modified Pr/T net. *G.IS* consists of a set of *methods*.

Definition 2.6 *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 *goal place*. Each method can have only one *entry place* and one *goal 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: $((P-\{goal\ place\}) \times T) \cup ((T \times (P-\{entry\ place\})$.

Part 2: An Agent-based G-net Model

- Becomes one of the most important topics in distributed and autonomous decentralized systems.
- Multi-agent systems (MAS): autonomous, reactive and internally-motivated agents.
- However, the standard G-net model is not sufficient for agent modeling because:
  - Do not support a common communication language and common protocols among agents.
  - Do not support asynchronous message passing directly.
  - Be awkward to model agent's mental state, such as goals, plans and knowledge.

An Agent-based G-net Model

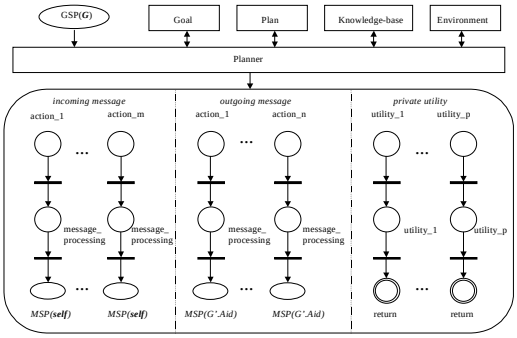


Figure 2. A generic agent-based G-net model

A Temple of Planner Module

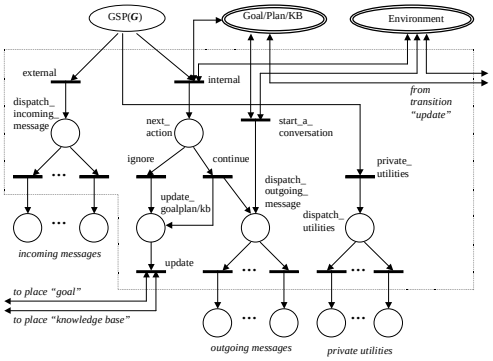


Figure 3. A template of *Planner* module

Definitions of the Message Token: *mTkn*

```
struct Message{
    int sender;             // the identifier of the message sender
    int receiver;           // the identifier of the message receiver
    string protocol_type;   // the type of contract net protocol
    string name;            // the name of incoming/outgoing messages
    string content;         // the content of this message
};

enum Tag {internal, external};

struct MtdInvocation {
    Triple (seq, sc, mtd);  // as defined in Section 2.1
}
if (mTkn.tag ∈ {internal, external})
then mTkn.body  =  struct {
    Message msg;            // message body
}
else mTkn.body =  struct {
    Message msg;            // message body
    Tag old_tag;            // to record the old tag: internal/external
    MtdInvocation miv;      // to trace method invocations
}
```

Formal Definitions of Agent-based G-net Model

Definition 3.1 *Agent-based G-net*
An *agent-based G-net* is a 7-tuple *AG = (GSP, GL, PL, KB, EN, PN, IS)*, where *GSP* is a *Generic Switch Place* providing an abstract for the agent-based G-net, *GL* is a *Goal* module, *PL* is a *Plan* module, *KB* is a *Knowledge-base* module, *EN* is an *Environment* module, *PN* is a *Planner* module, and *IS* is an *internal structure* of *AG*.

Definition 3.2 *Planner Module*
A *Planner module* of an agent-based G-net *AG* is a colored sub-net defined as a 7-tuple *(IGS, IGO, IPL, IKB, IEN, IIS, DMU)*, where *IGS, IGO, IPL, IKB, IEN* and *IIS* are interfaces with *GSP*, *Goal* module, *Plan* module, *Knowledge-base* module, *Environment* module and *internal structure* of *AG*, respectively. *DMU* is a set of decision-making unit, and it contains three abstract transitions: *make_decision*, *sensor* and *update*.

Definition 3.3 *Internal Structure (IS)*
An *internal structure (IS)* of an agent-based G-net *AG* is a triple *(IM, OM, PU)*, where *IM/OM* is the *incoming/outgoing message* section, which defines a set of *message processing units (MPU)*; and *PU* is the *private utility* section, which defines a set of *methods*.

Definition 3.4 *Message Processing Unit (MPU)*
A *message processing unit (MPU)* is a triple *(P, T, A)*, where *P* is a set of places consisting of three special places: *entry* place, *ISP* and *MSP*. Each *MPU* has only one *entry* place and one *MSP*, but it may contain multiple *ISP*s. *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 3.5 *Method*
A *method* is a triple *(P, T, A)*, where *P* is a set of places with three special places: *entry* place, *ISP* and *return* place. Each method has only one *entry* place and one *return* place, but it may contain multiple *ISP*s. *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: $((P-\{return\}) \times T) \cup ((T \times (P-\{entry\})$.

Selling and Buying Agent Design

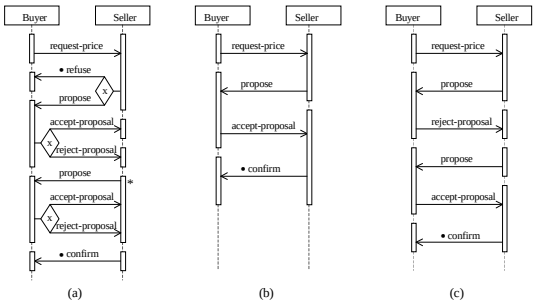
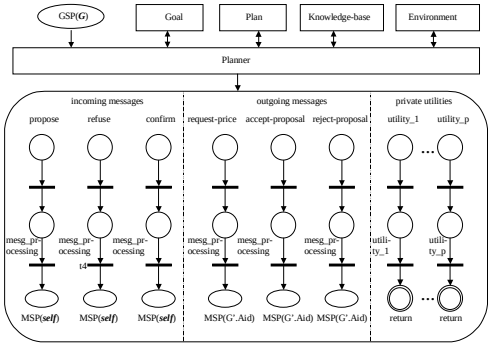


Figure 4. A contract net protocol between buying and selling agent

Selling and Buying Agent Design
(continue)



Notes: G'.Aid = mTkn.body.msg.receiver

Figure 5. An Agent-based G-net model for buying agent class

Verifying Agent-based G-net Model

- *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.

Verifying Agent-based G-net Model
(continue)

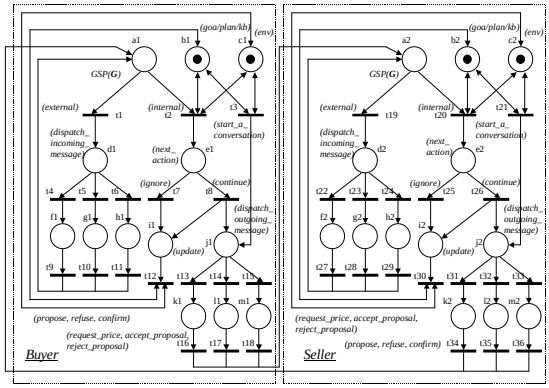
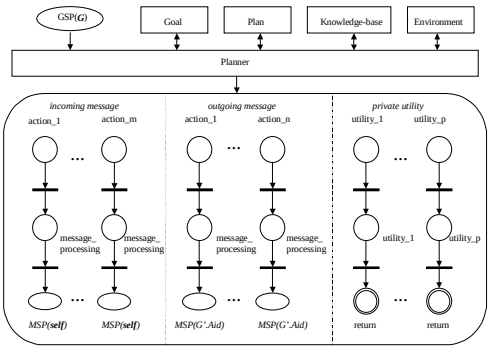


Figure 6. A transformed model of buying and selling agents

Part 3: A Framework for Modeling Agent-Oriented Software

- Extend existing methodologies:
  - object-oriented (OO) methodologies
  - knowledge engineering (KE) methodologies
- Follow the first approach, and separate traditional object-oriented features and reasoning mechanism to enhance reuse.
- Show the useful role of inheritance in agent-oriented software design.

Reuse of the Framework for the Agent-based Model



Notes: G'.Aid = mTkn.bodv.msa.receiver

Figure 7. A generic agent-based G-Net model

Redesign of the *Planner* Module
(initial design)

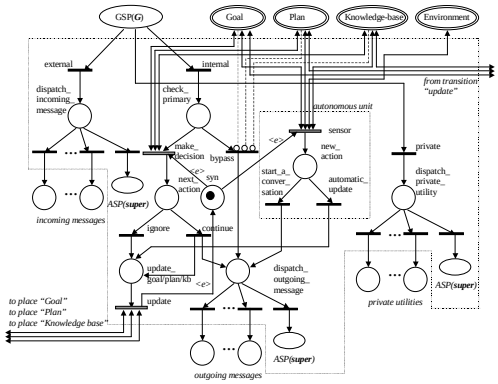


Figure 8. A template for the *Planner* module (initial design)

Part 4: Future Research Plans

- A unified model for object-oriented and agent-oriented software design.
  - complex software systems with both objects and agents.
  - object-object, agent-agent, and object-agent interactions.
- Extending agent-oriented G-net model for mobile agent design.
  - incorporate mobility into our agent models.
  - with application to electronic marketplace.
- Security issues in mobile agent design.
  - mobile agent and hostile agent modeling.