A Framework for Modeling Agent-Oriented Software

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: An agent-oriented G-net model.
- Part 3: Analysis of agent-oriented models.
- Part 4: Comments and future research plans.

Why Formal Methods?

- To write formal requirements specification, which serves as a contract between the user and the designer.
- To be used in software design. Design errors may be caught in an early design stage.
- To support system analysis and verification.
  - model checking
  - theorem proving

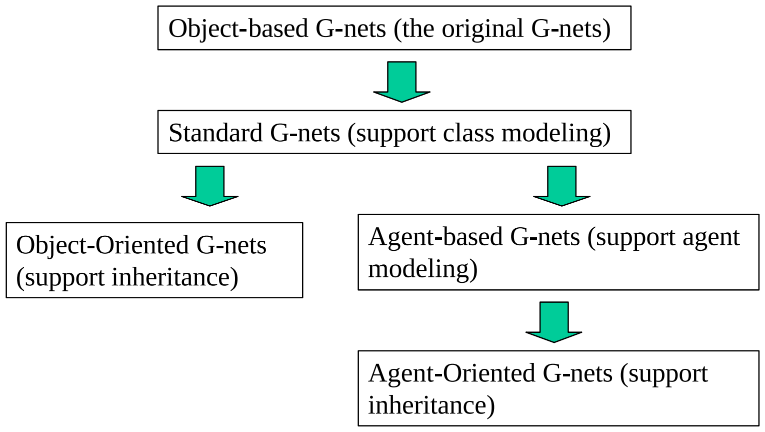
Agent-Oriented Approaches and Formal Methods

- 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
  - autonomous behavior modeling
- Agent-oriented approaches: provide guidelines for agent specification and design.
  - AAII 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 interactions, 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


Object-based G-nets (the original G-nets)
Standard G-nets (support class modeling)
Object-Oriented G-nets (support inheritance)
Agent-based G-nets (support agent modeling)
Agent-Oriented G-nets (support inheritance)

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.
  - do not use text-based formalism in our formal models.
  - 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

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

An Example

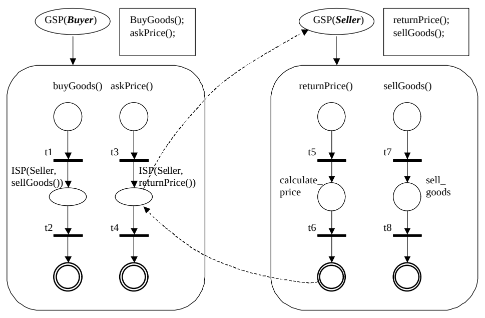


Figure 1. G-net models of buyer and seller objects

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*}) x T) ∪ ((T x (P-{*entry place*}).

Part 2: An Agent-Oriented 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-Oriented G-net Model

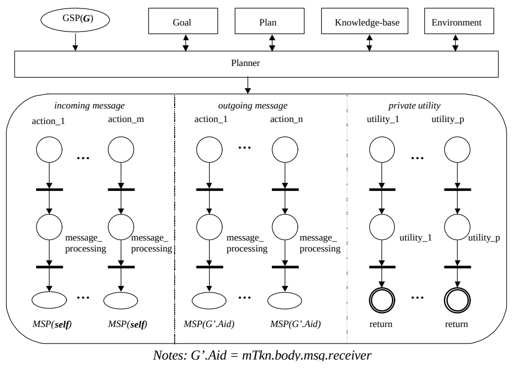
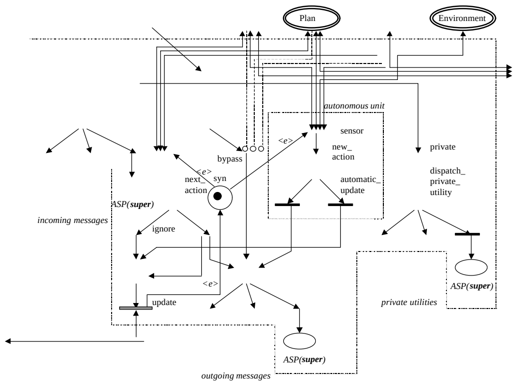


Figure 2. A generic agent-oriented G-net model

A Template for the *Planner* Module

(initial design)



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-Oriented G-net Model

Definition 3.1 *Agent-Oriented 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*}) x T) ∪ ((T x (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*}) x T) ∪ ((T x (P-{*entry*}).

Examples of Agent-Oriented Design
(class hierarchy)

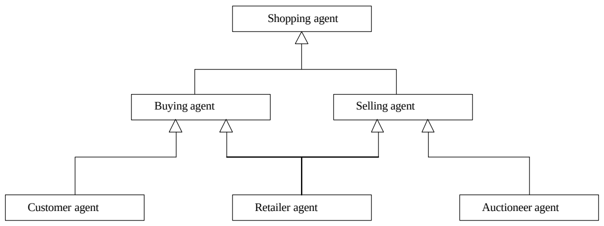


Figure 4. The class hierarchy diagram of agents in an electronic marketplace

Examples of Agent-Oriented Design
(contract net protocol)

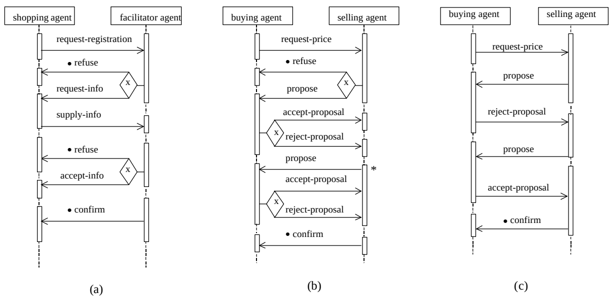


Figure 5. Contract net protocols (a) A template for the registration protocol (b) A template for the price-negotiation protocol (c) An example of the price-negotiation protocol

Examples of Agent-Oriented Design
(shopping agent class)

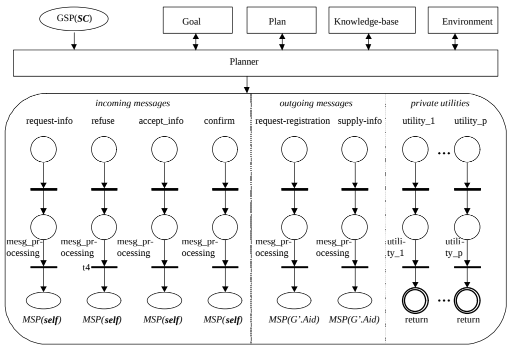


Figure 6. An agent-based G-net model for shopping agent class (SC)

Examples of Agent-Oriented Design
(buying agent class)

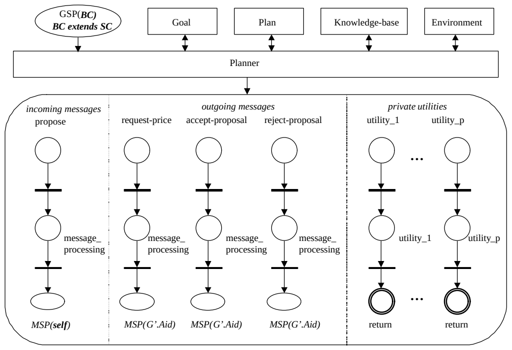


Figure 7. An agent-based G-net model for buying agent class (BC)

Part 3: Analysis of Agent-Oriented Models

- To help ensure a correct design that meets certain specifications.
- To meet certain requirements such as liveness, deadlock freeness and concurrency.
- Use Petri net tool: INA (Integrated Net Analyzer)
  - verifying structural properties
  - verifying behavioral properties
  - modeling checking (CTL formulas)

A Transformed Model of One Buying Agent and Two Selling Agents

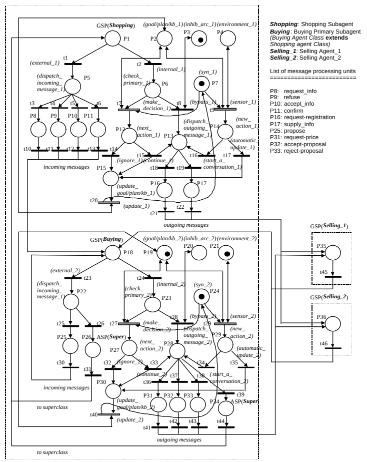


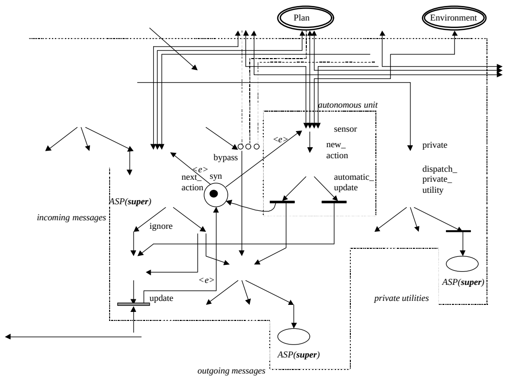
Figure 8. A transformed model of one buying agent and two selling agents

Experiment Result -1

```
Computation of the reachability graph
States generated: 8193
Arcs generated: 29701

Dead states:
      484, 485,8189
Number of dead states found: 3
The net has dead reachable states.
The net is not live.
The net is not live and safe.
The net is not reversible (resetable).
The net is bounded.
The net is safe.
The following transitions are dead at the initial marking:
       7, 9, 14, 15, 16, 17, 20, 27, 28, 32, 33
The net has dead transitions at the initial marking.
```

Redesign of the *Planner* Module



Experiment Result - 2

```
Computation of the reachability graph
States generated: 262143
Arcs generated: 1540095

The net has no dead reachable states.
The net is bounded.
The net is safe.
The following transitions are dead at the initial marking:
       7, 9, 14, 15, 16, 17, 20, 28
The net has dead transitions at the initial marking.

Liveness test:
Warning: Liveness analysis refers to the net where all dead transitions
   are ignored.
The net is live, if dead transitions are ignored.
The computed graph is strongly connected.
The net is reversible (resetable).
```

Property Verification by Using Modeling Checking

- *Concurrency*

```
EF(P5 &(P13 &(P22 &P28)))     Result: The formula is TRUE
```

- *Mutual Exclusion*

```
EF(P27 &P30) V (P29 &P30))    Result: The formula is FALSE
```

- *Inheritance Mechanism (decision-making in subagent)*

```
AG(-P12 &(-P14 &-P15))        Result: The formula is TRUE
```

- *Inheritance Mechanism (ASP message forwarding I)*

```
A[(P26 VP34)B(P5 VP6)]        Result: The formula is TRUE
```

- *Inheritance Mechanism (ASP message forwarding II)*

```
A[P26 BP5]VA[P34 BP6]         Result: The formula is FALSE
```

Part 4: Concluding Comments

- There is an increasing need to ensure that complex software systems being developed are robust, reliable and fit for purpose.
- Petri nets are an excellent formalism for formal specification because they tend to provide a visual, and thus easy to understand, model.
- Extending G-nets to support inheritance in object-oriented design and agent-oriented design provides an effective way for modeling complex software systems.

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.