From Object to Agent: An Approach to Using Formal Methods in Software Design and Analysis

HAIPING XU

PH.D. THESIS PROPOSAL
Electrical Engineering and Computer Science Department
The University of Illinois at Chicago, 2001
Chicago, Illinois

Outline

- Introduction: Related work and our approach.
- Part 1: Inheritance modeling in object-oriented design.
- Part 2: Design of agent-based G-net model.
- Part 3: Modeling agent-oriented software.
- Part 4: Analysis of agent-oriented models.
- Part 5: 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

Formal Methods in Object-Oriented Design

- Object-oriented formal methods
  - OPN (Object Petri Nets), VDM++, Object-Z etc.
- Examples of OPN approaches
  - OBJSA/CLOWN: CLass Orientation with Nets*
  - CO-OPN/2: Concurrent Object-Oriented Petri Nets*
  - LOOPN++: Language for Object-Oriented Petri Nets*
  - G-nets: Generic Petri Nets**

* Do not support net-based analysis and verification.

** Do not support inheritance.

- Our proposed formal models are based on G-nets.

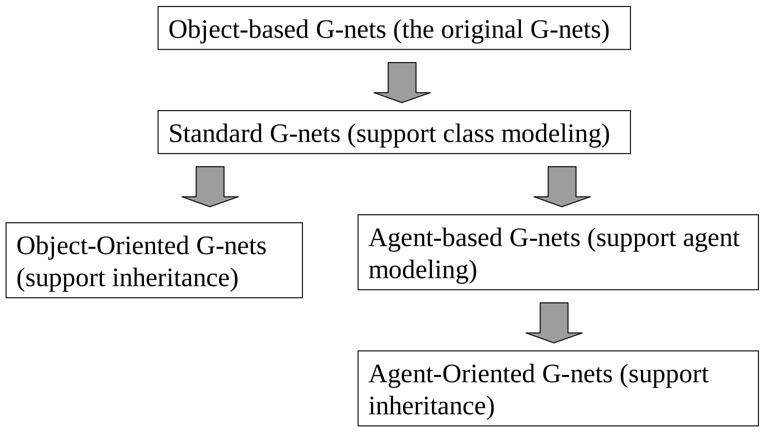
Agent-Oriented Approaches and Formal Methods

- 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 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.
  - support net–based behavior analysis and verification.
  - may unify the object-oriented G-nets and agent-oriented G-nets to model complex software systems.

Contributions of Our Work

- Extended the original G-net model to support class modeling and inheritance modeling.
- Designed an agent-based G-net model, and proved properties related to *liveness*, *concurrency* and *effectiveness* for agent communication.
- Extended the agent-based G-net model to support inheritance modeling in agent-oriented design.
- Performed experiments with an existing Petri net tool to model and analyze agent-oriented software systems.

Part 1: Inheritance Modeling in Object-Oriented Design

- Allows users to specify a subclass that inherits features from its superclass.
- Question: how inheritance can be incorporated into formalisms such as object Petri nets (OPN)?
- CLOWN, LOOPN++ and CO-OPN/2 use text-based grammar to incorporate inheritance into Petri nets.
- Examples of net-based inheritance modeling:
  - life-cycle inheritance [Aalst and Basten 1997]
  - extended SBOPN formalism [Xie 2000]

G-nets: A 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.
- Are not fully object-oriented due to a lack of support for inheritance.

An Example

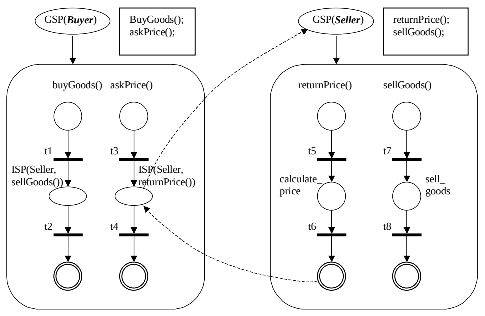


Figure 1. G-net models of buyer and seller objects

Extending G-nets to Support Class Modeling

- Motivation: to support 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*}).

Different Forms of Inheritance

- *Augment Inheritance*: new protocols are added to a subclass model.
- *Restrictive Inheritance*: some superclass methods are absent from the protocol of the subclass.
- *Replacement Inheritance*: a subclass can completely redefine the behavior of its superclass for a particular method defined in the superclass.
- *Refinement Inheritance*: the subclass contains a method that includes the behavior of its superclass, but extends it in some way.

Extending G-net to Support Inheritance

- *Default Place*: a default entry place defined in the internal structure of a subclass model.
- The default place is marked only if the method is not defined in the subclass model.
- *Superclass Switch Place (SSP)*: is used to forward a method call to a subobject of the object itself.

A G-net Model of Unbounded Buffer UB

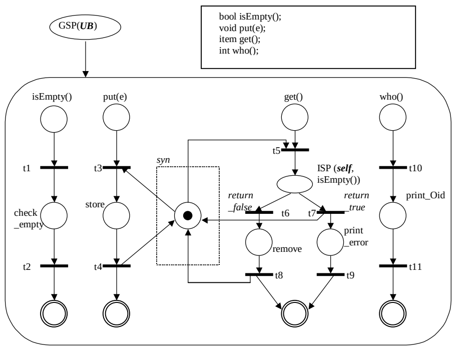


Figure 2. G-net model of unbounded buffer class (UB)

A G-net Model of Bounded Buffer BB

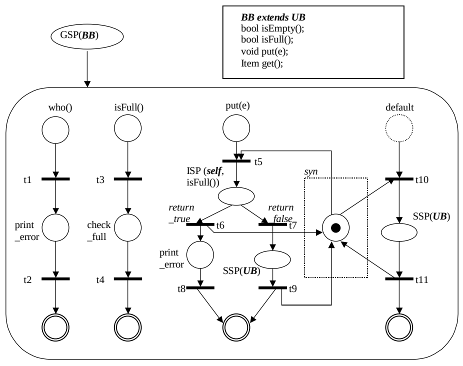
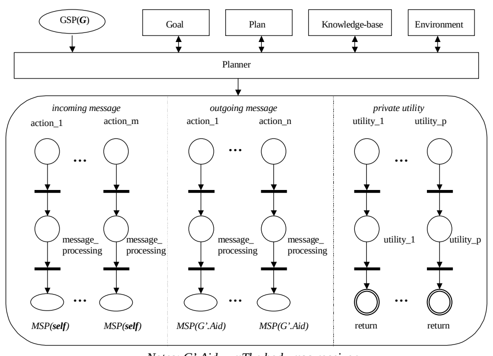


Figure 3. G-net model of bounded buffer class (BB)

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



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

Figure 4. A generic agent-based G-net model

A Temple of Planner Module

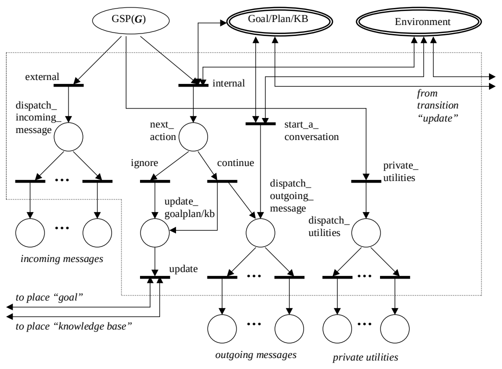


Figure 5. A template of *Planner* module

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*}) 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*}).

Selling and Buying Agent Design

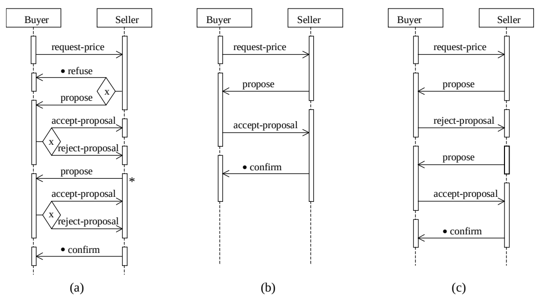
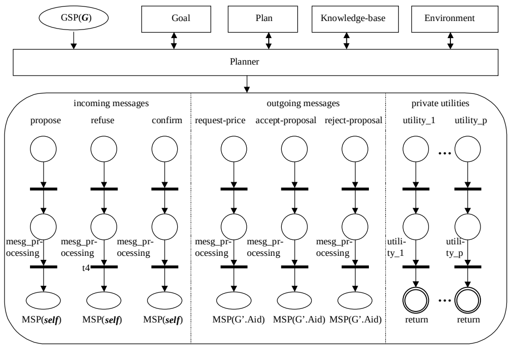


Figure 6. A contract net protocol between buying and selling agent

Selling and Buying Agent Design
(continue)



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

Figure 7. 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)

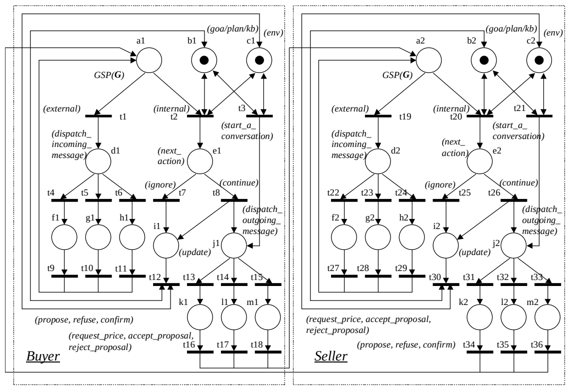
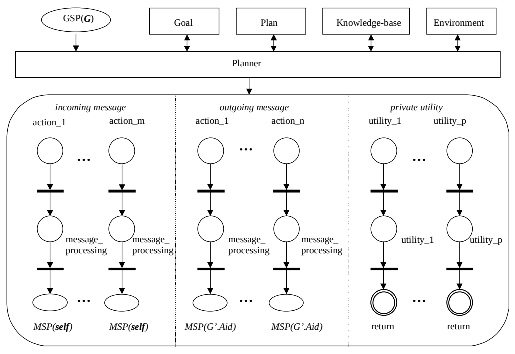

Buyer
Seller
GSP(G)
(goa/plan/kb)
(env)
(external)
(internal)
(start_a_conversation)
(dispatch_incoming_message)
(next_action)
(ignore)
(continue)
(dispatch_outgoing_message)
(update)
(propose, refuse, confirm)
(request_price, accept_proposal, reject_proposal)

Figure 8. A transformed model of buying and selling agents

Part 3: A Framework for Modeling Agent-Oriented Software

- Extend existing methodologies:
  - object-oriented (OO) methodologies ==> KGR approaches.
  - knowledge engineering (KE) methodologies ==> Gaia 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 Agent-based Model



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

Figure 9. A generic agent-based G-Net model

Redesign of the *Planner* Module to Support Inheritance

- *Abstract transitions*: represents abstract units of decision-making or mental-state-updating.
- *Autonomous units*: makes an agent autonomous and internally-motivated.
- *Asynchronous Superclass switch Place (ASP)*: is used to forward a method call to a subagent of the agent itself.

A Template for the *Planner* Module
(initial design)

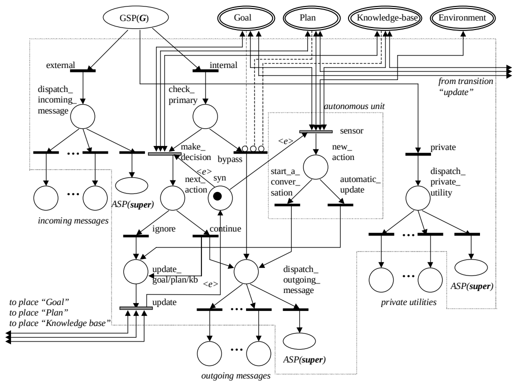


Figure 10. A template for the *Planner* module (initial design)

Examples of Agent-Oriented Design
(class hierarchy)

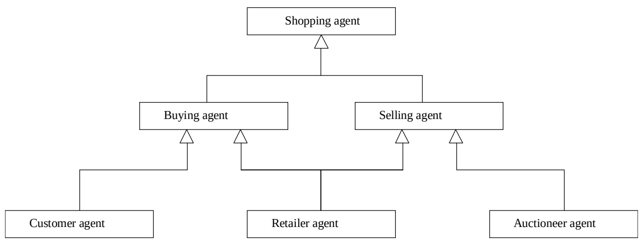


Figure 11. The class hierarchy diagram of agents in an electronic marketplace

Examples of Agent-Oriented Design
(contract net protocol)

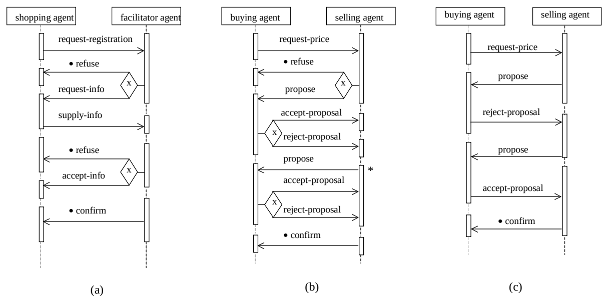


Figure 12. Contract net protocols (a) A template for the registration protocol (b) A template for the price-negotiation protocol (c) An example of the price-negotiation protocol

Examples of Agent-Oriented Design
(shopping agent class)

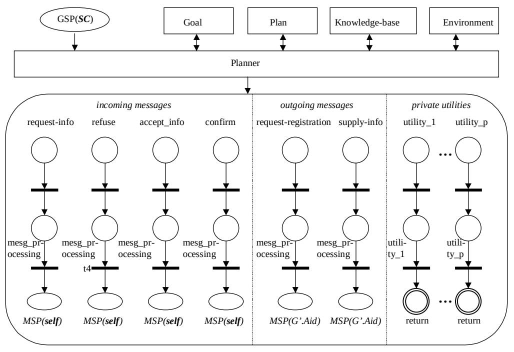


Figure 13. An agent-based G-Net model for shopping agent class (SC)

Examples of Agent-Oriented Design
(buying agent class)

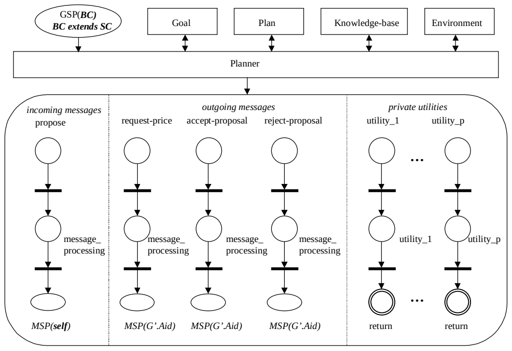


Figure 14. An agent-based G-Net model for buying agent class (BC)

Part 4: 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

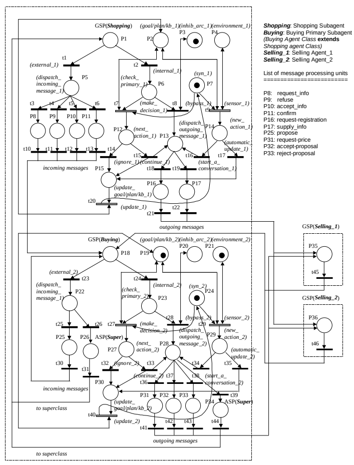


Figure 15. A transformed model of one buying agent and two selling agents

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

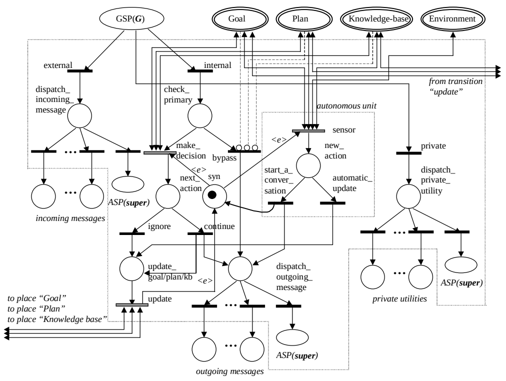


Figure 16. A template for the *Planner* module

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)*

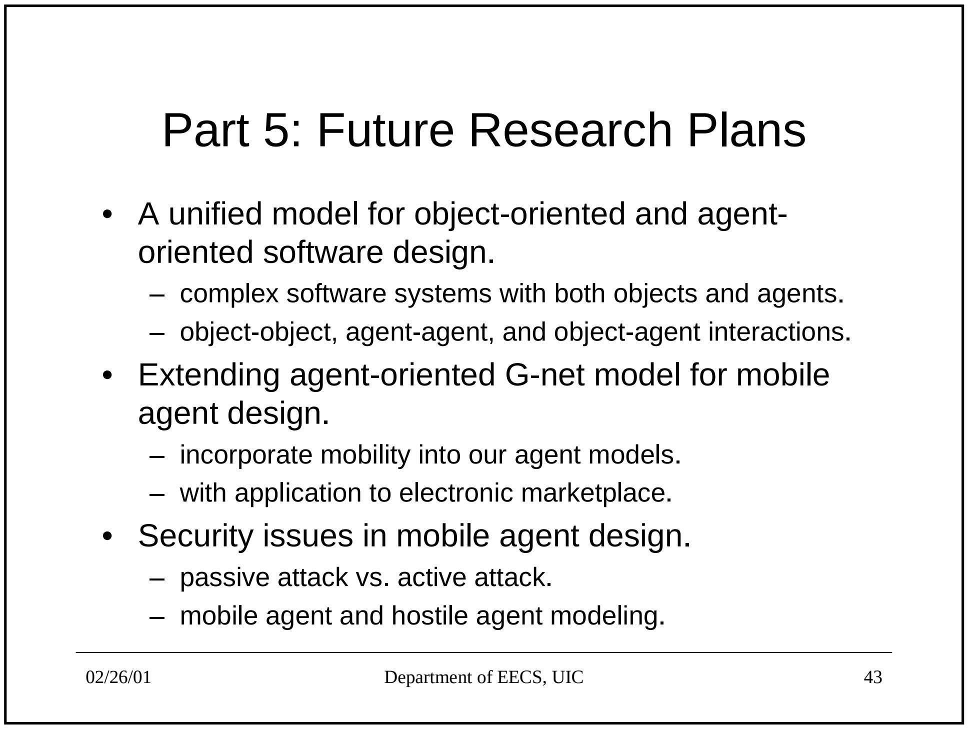
```
A[(P26 VP34)B(P5 VP6)]        Result: The formula is TRUE
```

- *Inheritance Mechanism (ASP message forwarding II)*

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

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.



Part 5: 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.
  - passive attack vs. active attack.
  - mobile agent and hostile agent modeling.