A Model-Based Approach for Development of Multi-Agent Software Systems

Haiping Xu, Ph.D.
Computer and Information Science Department
University of Massachusetts Dartmouth
http://www.cis.umassd.edu/~hxu

Acknowledgments

- Sol M. Shatz
  Thesis Advisor, Professor at UIC
  Co-Director, Concurrent Software Systems Laboratory
- Tadao Murata
  UIC Distinguished Professor
  Director, Concurrent Software Systems Laboratory

Outline

- Part 1: Background and Motivations
- Part 2: Design of Agent-Based G-net Model
- Part 3: Modeling Agent-Oriented Software
- Part 4: Analysis of Agent-Oriented Model
- Part 5: Design of Intelligent Mobile Agents
- Part 6: Agent Development Kit (ADK)
- Part 7: Conclusions and Future Work.

Part 1: Background and Motivations

- The development of software systems starts with two main activities:
  - Software requirements analysis
  - Software design
- Software requirements analysis: to reduce potential errors caused by incomplete and ambiguous requirements
- Software design: to depict the overall structure of a system by decomposing the system into its logical components.

Formal Methods in Software Engineering

- The purpose of software requirements analysis can be achieved in two ways:
  - Write a specification in natural languages
  - Choose a formal language, e.g., Petri nets
- Ideally, formal methods can be applied in each phase of the software development life cycle, e.g., the design phase
- However, to create a formal model in the design phase and to verify its correctness is rare.

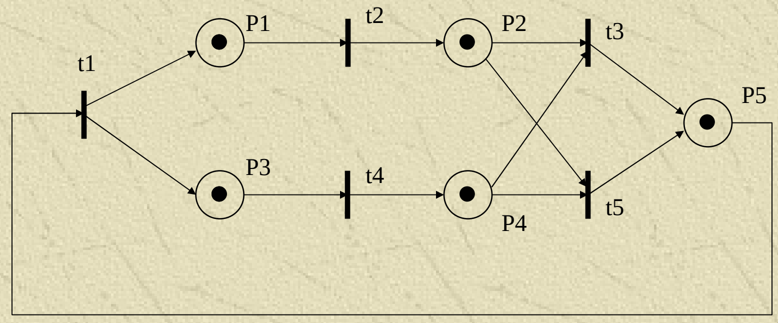
Introduction to Petri Net

- "Three-in-one" capability of Petri net models [Murata 1989]
  - Graphical representation
  - Mathematical description
  - Simulation tool
- Definition:

A Petri net is a 4-tuple, PN = (P, T, F, M_0) where

P = {P1, P2, …, Pm} is a finite set of places;
T = {t1, t2, …, tn} is a finite set of transitions;
F $\subseteq$ (P x T) $\cup$ (T x P) is a set of arcs (flow relation);
M_0: P --> {0, 1, 2, 3, …} is the initial marking.

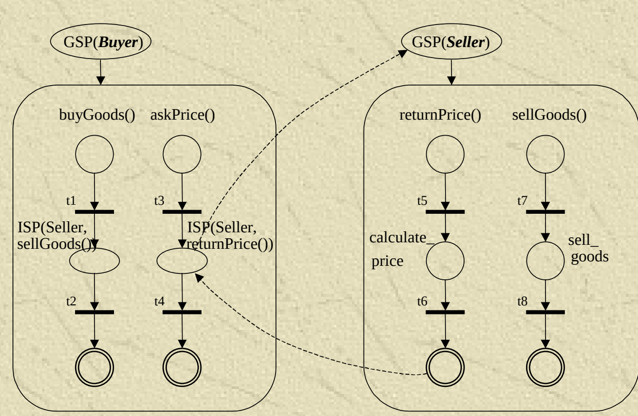
An Example



G-Net: A High Level Petri Net

- Defined to support modeling of systems as a set of independent and loosely-coupled modules [Deng *et al.* 1993]
- Provides support for incremental design and successive modification
- Is not fully object-oriented due to a lack of support for inheritance.

An Example



Introduction to Agents

- The term “agent” comes from the greek word “agein”, which means to drive or to lead
- A agent is a computer system that is situated in some *environment*, and that is capable of *autonomous action*
- It is suitable to describe current trends in computer science: active instruments (to which work can be delegated) vs. passive tools
- The term “agent” in computer science generally refers to a software agent.

Research Directions

- Multi-agent systems (MAS)
  - Agents act as “active” objects (intelligence)
  - Collaborative or competitive
  - Generally use distributed but static (non-mobile) agents
- Mobile agents (MA)
  - Model agent mobility and agent coordination
  - Generally assume very limited or even no intelligence.

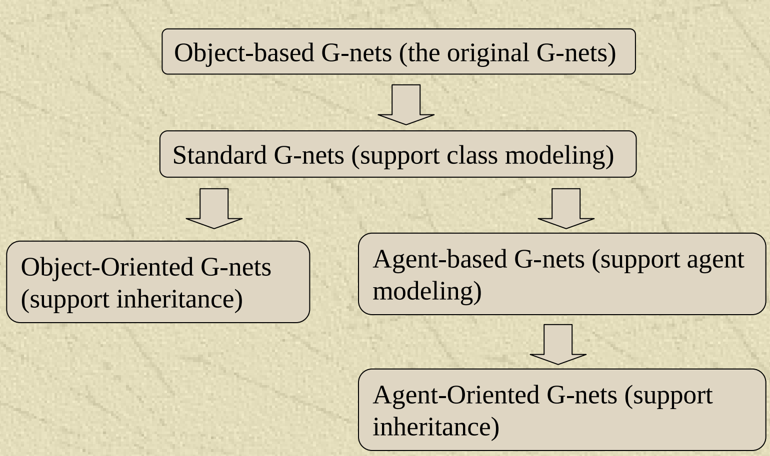
Agent-Oriented Software Engineering

- 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 (mental states)
- Agent-oriented approaches: provide guidelines for agent specification and design
  - AAII methodologies: based on BDI model
  - Gaia methodologies: based on role modeling.

Formal Methods in Agent-Oriented Software Engineering

- Limited work on *formal* specification and design for agents
  - Use formal languages, such as Z, to provide a framework for describing the agent architecture [Luck and d'Inverno 1995]
  - Use temporal logic to specify agent behavior [Fisher 1995]
  - Design formal languages for specifying agent-based systems, e.g., DESIRE [Brazier *et al.* 1997] and SLABS [Zhu 2001]
  - 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:
  - Serve as a high-level design for agent implementaion
  - Support protocol-based agent interaction/communication
  - Support reuse of functional units of our agent class models.

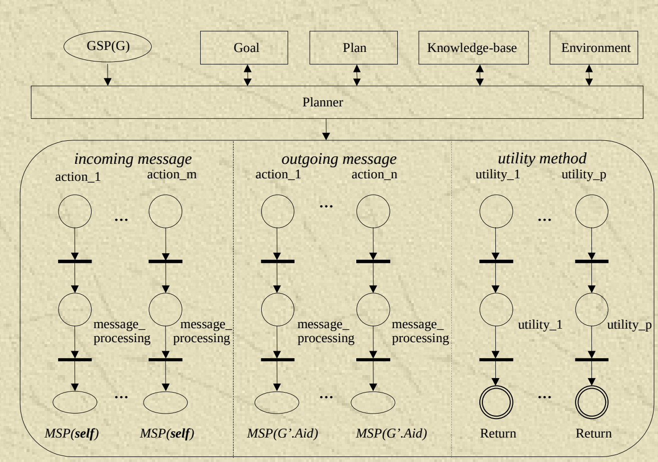
Our Incremental Approach



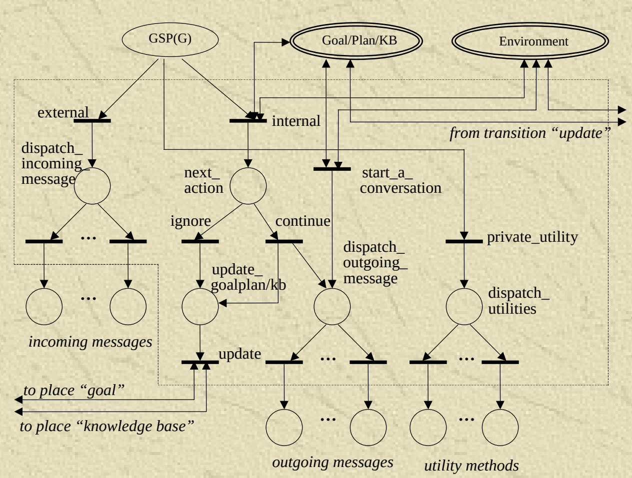
Part 2: An Agent-Based G-Net Model

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

An Agent-Based G-Net Model



A Template of *Planner* Module



Definitions of the Message Token

```
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*.

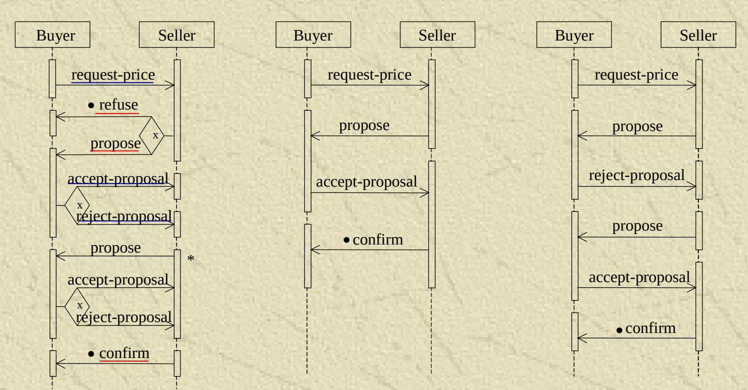
Definition 3.2 Planner Module
A *Planner module* of an agent-based G-net *AG* is a colored sub-net defined as a 7-tuple *(IGS, IGO, IPL, IKB, IEN, IIS, DMU)*, where *IGS, IGO, IPL, IKB, IEN* and *IIS* are interfaces with *GSP*, *Goal* module, *Plan* module, *Knowledge-base* module, *Environment* module 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 *ISPs*. *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*}).

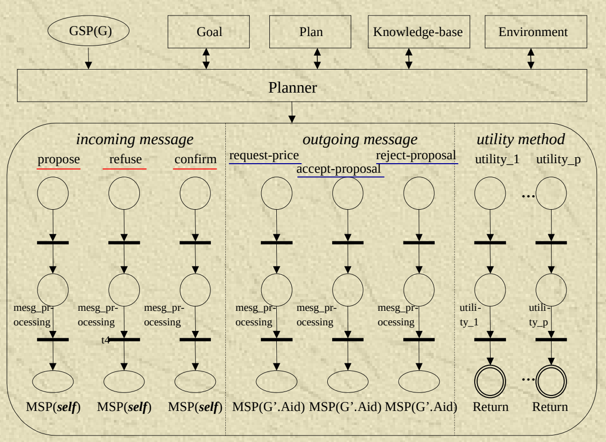
Definition 3.5 U-Method
A *U-Method* or *method* is a triple *(P, T, A)*, where *P* is a set of places with three special places: *entry* place, *ISP* and *return* place. Each method has only one *entry* place and one *return* place, but it may contain multiple *ISPs*. *T* is a set of transitions, and 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



(a) (b) (c)

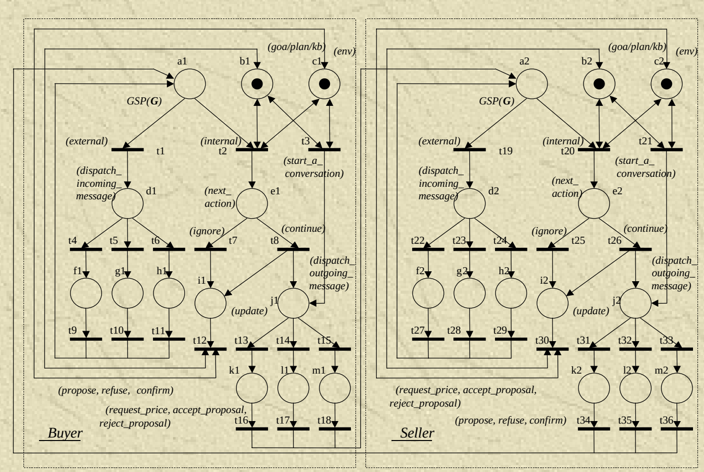
Buying Agent Class Design



Verifying the 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 concurrently
- *Effective*: an agent communication protocol can be correctly traced in the agent models.

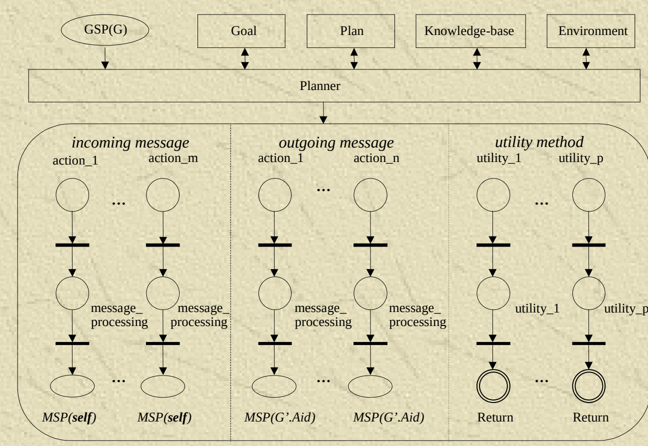
Verifying the Agent-Based G-Net Model (continued)



Part 3: A Framework for Modeling Agent-Oriented Software

- Core approaches that can be extended:
  - Object-oriented (OO) methodologies
  - Knowledge engineering (KE) methodologies
- We follow the first approach, and separate traditional object-oriented features and reasoning mechanism to enhance reuse of communication components
- Show the useful role of inheritance in agent-oriented software design.

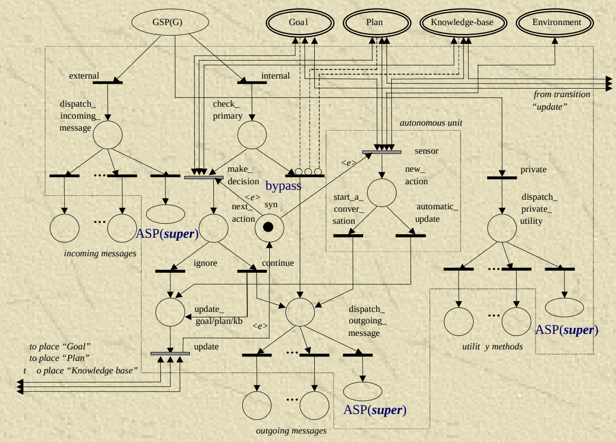
The Agent-Based Model (Revisited)



A Revised *Planner* Module

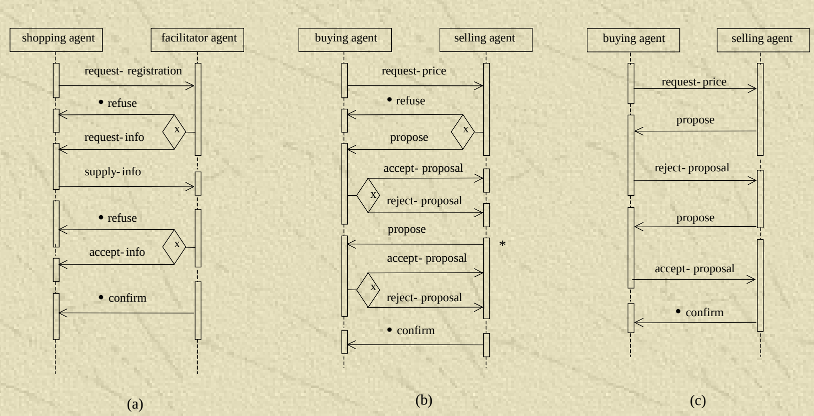
- *Abstract transition*: represents abstract units of decision-making or mental-state-updating (with synchronization)
- *Autonomous unit*: makes an agent autonomous and internally-motivated
- *Asynchronous Superclass switch Place (ASP)*: used to forward a MPU or a method call (token) to a "superclass" model in the case of inherited communication mechanisms.

A Template for the *Planner* Module
(Initial Design)

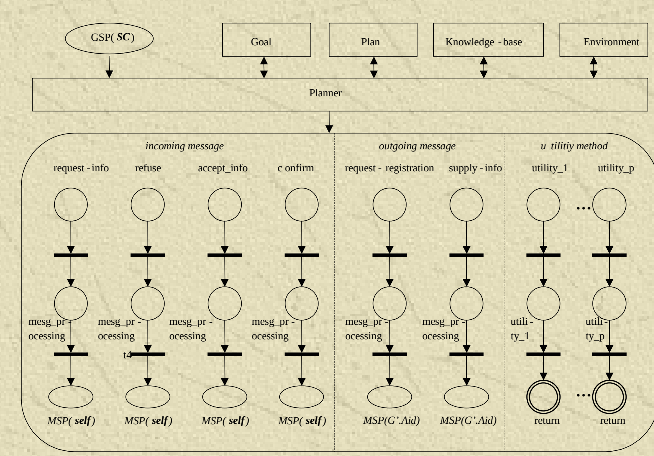


Examples of Agent-Oriented Design
(class hierarchy)

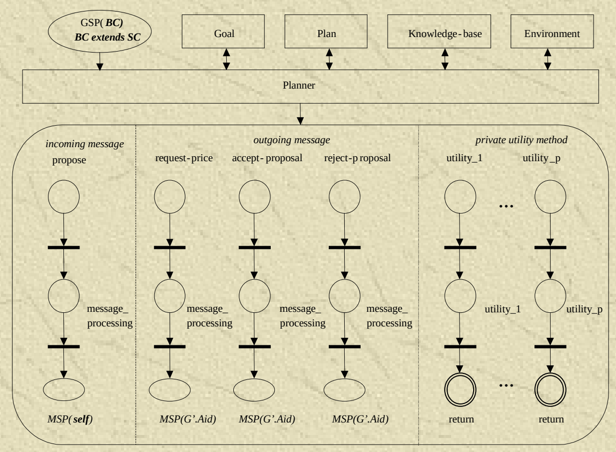
Examples of Agent-Oriented Design
(contract net protocol)



Examples of Agent-Oriented Design
(shopping agent class)



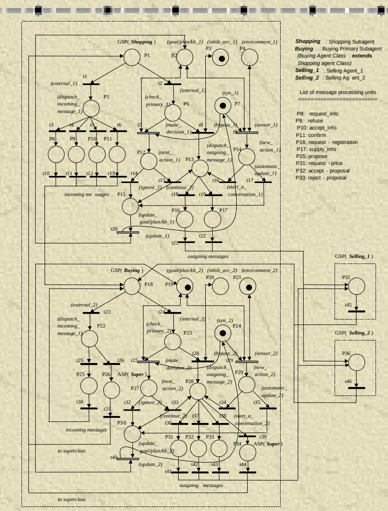
Examples of Agent-Oriented Design
(buying agent class)



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

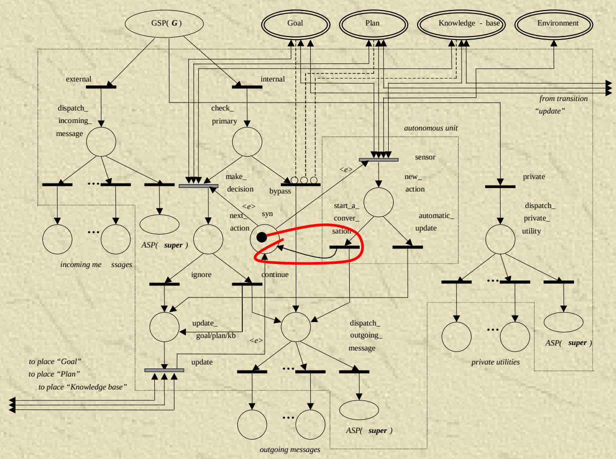


Experiment: Result -1

```
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 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 Using Modeling Checking

- *Concurrency (e.g., process incoming and outgoing messages)*

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

- *Mutual Exclusion (don't want simultaneous updating of goals, plan, etc.)*

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

- *Inheritance Mechanism (decision-making in a superclass model)*

  `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`

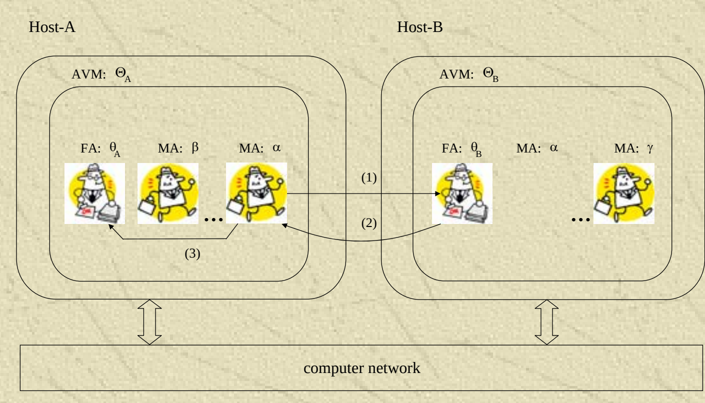
- *Inheritance Mechanism (ASP message forwarding II)*

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

Part 5: Design of Intelligent Mobile Agents – A Generic Model

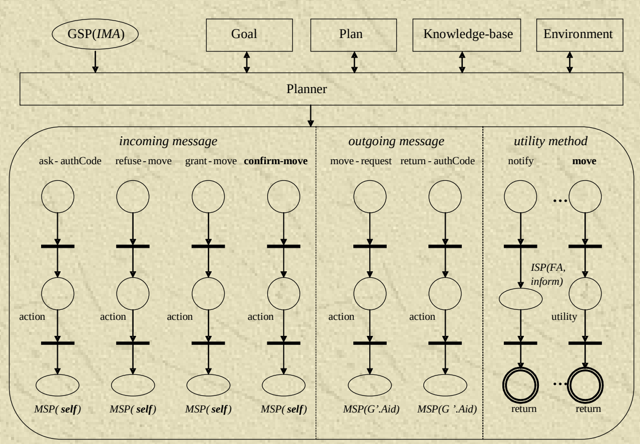
- Two schemes for agent development:
  - Weak agent approach
  - Strong agent approach
- Most of the existing work on mobile agents use weak agent approach (not flexible, security issues …)
- In contrast, we propose a generic model for *intelligent* mobile agent.

Agent World Architecture

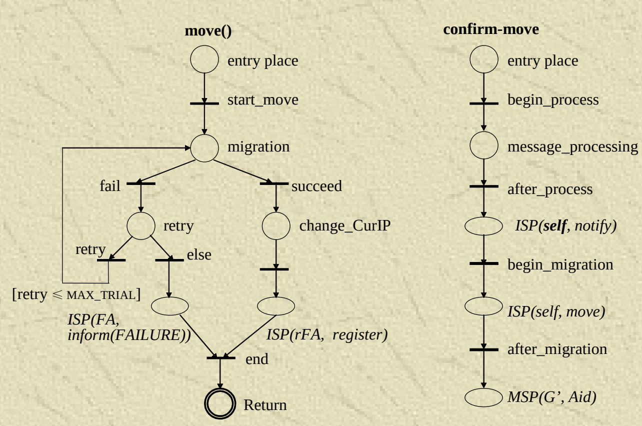


(1) move-request (2) grant (3) notify (4) move

Intelligent Mobile Agent (IMA)



Refinement of Functional Units



(a) (b)

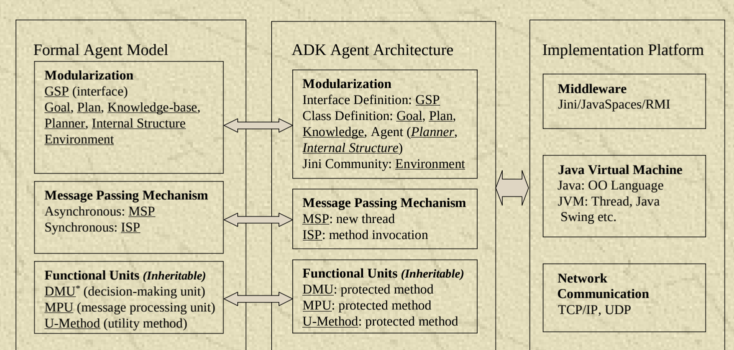
Advantages of This Approach

- Application-specific mobile agent class can be defined as a subclass of IMA
- With further refinement of functional units, the design of mobile agents can be verified to meet certain requirements
- The strong agent approach leaves adequate room for security modeling.

Part 6: Agent Development Kit (ADK)

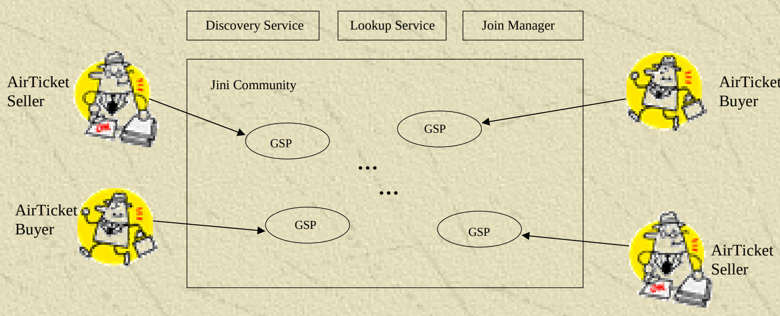
- Most of existing work on agent formal modeling defines *what* properties are to be realized by an agent system instead of *how*
- In contrast, agent-oriented G-net model serves as a high-level design for agent development
- Implement Agent Development Kit (ADK)
  - Use Jini middleware for agent communication
  - Provide a framework and a full class library to support development of application-specific agents for MAS.

Overview of Agent Design Architecture

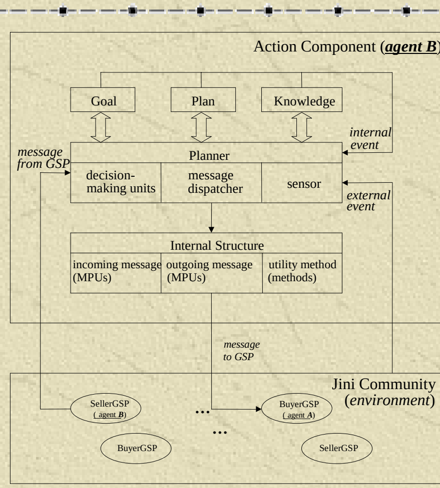


** DMUs are not inheritable in agent-oriented G-net model*

The Jini Community



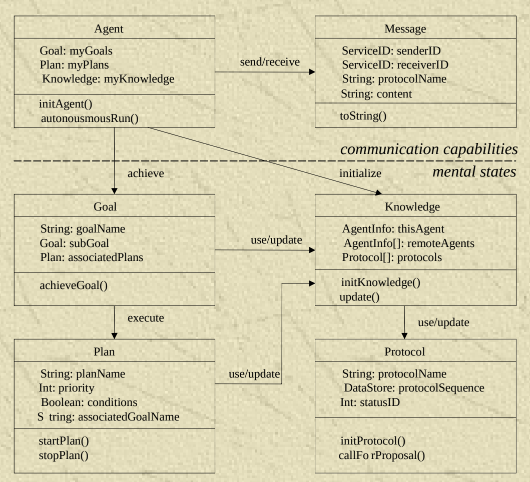
The Agent Architectural Design



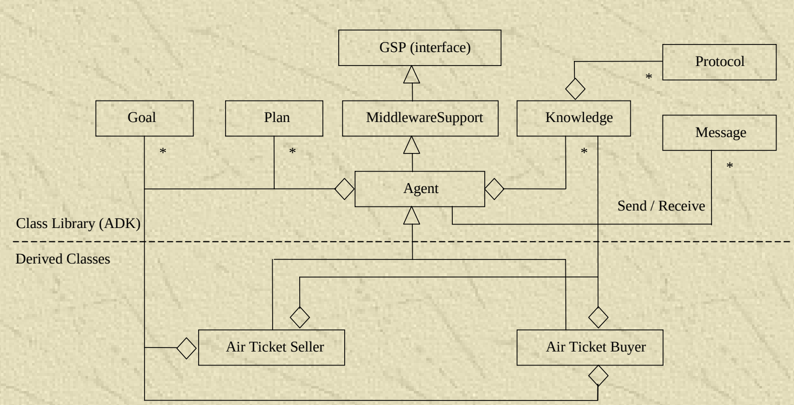
A Design Pattern for Intelligent Agents

- Define the *Agent* class to provide a pattern for agent implementation
- Derive the *Agent* class from *MiddlewareSupport*, which deals with the Jini community
- Separate agent communication capabilities and agent mental states to enhance reuse of agent design.

Classes Diagram: Communication Capabilities and Mental States



Classes defined in ADK and derived classes of the *Agent* class



A Design Pattern for Application-Specific Agents

```
public class ApplicationSpecificAgent extends Agent {

        /************************************************
         * Class Variables for Knowledge, Goal and Plan *
        ************************************************/
        Goal myGoals; // committed goals, redefinition of Goal class is optional
        Plan myPlans; // plans, redefinition of Plan class is optional
        Knowledge myKnowledge; // knowledge-base, redefinition of Knowledge
                               // class is optional
     …

     /***********
      * Planner *
      ***********/
     protected void dispatchMessage(Message message) {…}  // refinement
                                                          // or redefinition
     protected Message makeDecision(Message message) {…}  // refinement
                                                          // or redefinition
     protected void updateMentalState() {…}               // refinement
                                                          // or redefinition
     …
```

A Design Pattern for Application-Specific Agents (continued)

```
        /**********************
         * Internal Structure *
         **********************/
        // incoming message section – a set of message processing units
        protected void MPU_In_1(Message message) {…}         // new definition
        …

        // outgoing message section – a set of message processing units
        protected void MPU_Out_1(Message outgoingMessage) {…}// new definition
        …

        // utility method section – a set of private utility methods
        protected void initAgent(String[] args) {…}          // refinement
                                                             // or redefinition
        protected void autonomousRun() {…}                   // refinement
                                                             // or redefinition
        protected void other_Inherited_Method_1() {…}        // refinement
                                                             // or redefinition
        …
        protected void other_New_Method_1() {…}              // new definition
        …

        public static void main(String[] args) {
            initAgent(args);
            autonomousRun();
        }
}
```


A Case Study: Air-Ticket Trading

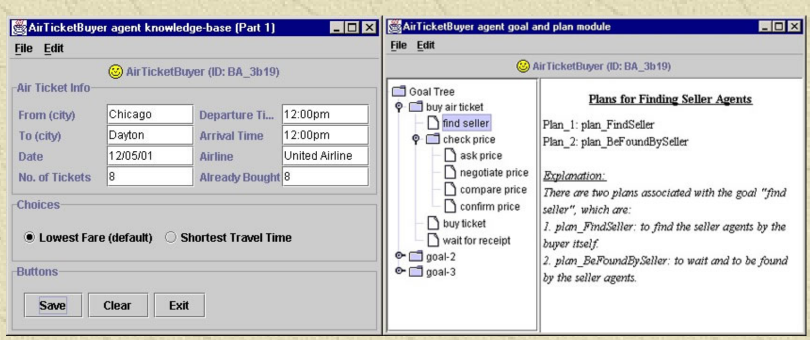


Figure I. The agent interface for knowledge-base, goal and plan modules

A Case Study: Air-Ticket Trading (continued)

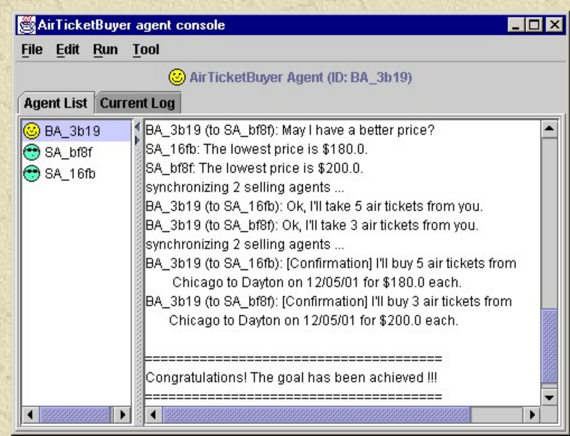


Figure II. The agent interface for a buyer agent

Part 7: Concluding Comments

- There is an increasing need to ensure that complex software systems are robust, reliable and fit for purpose (Agent-Oriented SE)
- Petri nets provide a formal and visual model with natural expression for concurrency and coordination
- Adapt Petri net models to define a model-based approach for AOSE and an associated ADK.

Future Work

- Design and implement a compilation process and tools to automatically translate agent communication protocols into MPUs and decision-making units
- Develop a model-based agent development environment (ADE) for rapid agent design and implementation (i.e., synthesis of the work)
- Bring formal methods into the testing phase for concurrent and distributed software systems
- Perform research on the following areas:
  - Agent mobility
  - Agent-based peer-to-peer computing
  - Ubiquitous computing.

Thanks for your attention!

My Ph.D. thesis and the slides for this talk may be downloaded from

http://www.cis.umassd.edu/~hxu