Formal Methods in Agent-Oriented Software Engineering

Haiping Xu and Sol M. Shatz
Computer Science Department
The University of Illinois at Chicago

Outline

- Part 1: Background.
- Part 2: Design of agent-based G-net model.
- Part 3: Modeling agent-oriented software.
- Part 4: Analysis of agent-oriented model.
- Part 5: Our current research work.
- Part 6: Conclusions and future work.

Part 1: Background

- Formal methods in Software Engineering.
- Introduction to Petri net.
- G-net: A high level Petri net.
- Introduction to software agent.
- Agent-oriented software engineering.
- Our approach to agent-oriented design.

Purpose of Formal Methods

"The term "formal methods" denotes software development and analysis activities that entail a degree of mathematical rigor. (…) A formal method manipulates a precise mathematical description of a software system for the purpose of establishing that the system does or does not exhibit some property, which is itself-precisely defined." (Dillon and Sankar, 1997)

Dillon, L. K. and S. Sankar (1997), Introduction to the Special Issue, *IEEE Transactions on Software Engineering,* Special Issue on Formal Methods in Software Practice, 23(5): 265-266.

Formal Methods in Software Engineering

- 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

Introduction to Petri Net

- "Three-in-one" capability of a Petri net model.
  - 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

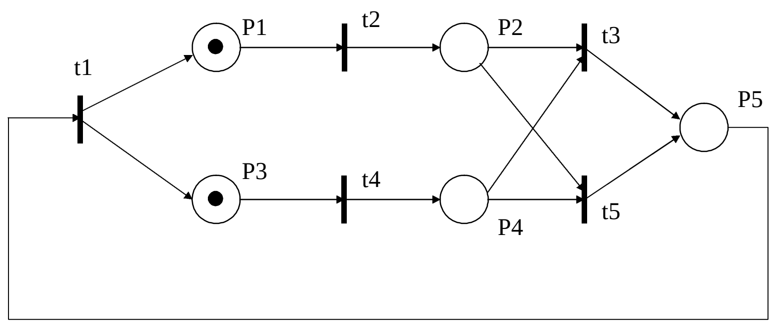


Figure 1. A simple Petri net model example

G-net: A High Level Petri Net

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

An Example

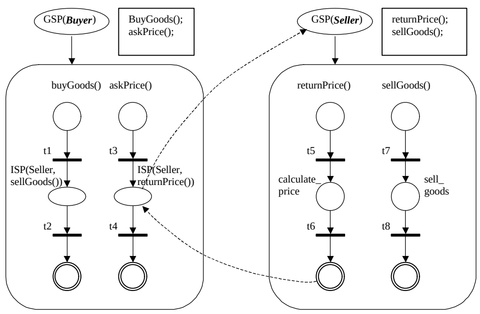


Figure 2. G-net models of buyer and seller objects

Introduction to Software Agent

- The term “agent” comes from greek “agein”, which means to drive or to lead.
- Today the term “agent” denotes something that producing an effect, e.g., drying agent, a shipping agent.
- 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 refers to software agent.

Space of Software Agents

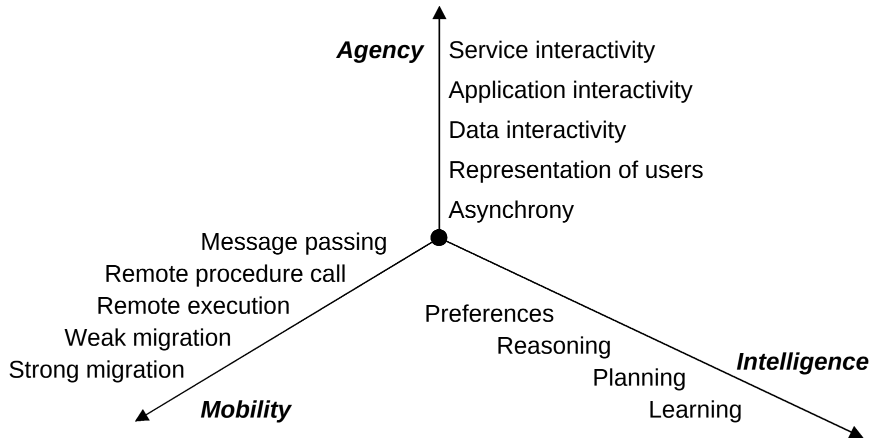


Figure 3. Space of software agents defined by IBM

Current Researches on Agents

- Do not exploit all the capabilities classified by these three dimensions.
- Multi-agent systems (MAS)
  - Execute a given task.
  - Use distributed but static agents.
  - Collaborate and cooperate in an intelligent manner.
- Mobile agents (MA)
  - Model agent mobility and agent coordination.
  - Assume very limited or even no intelligence.

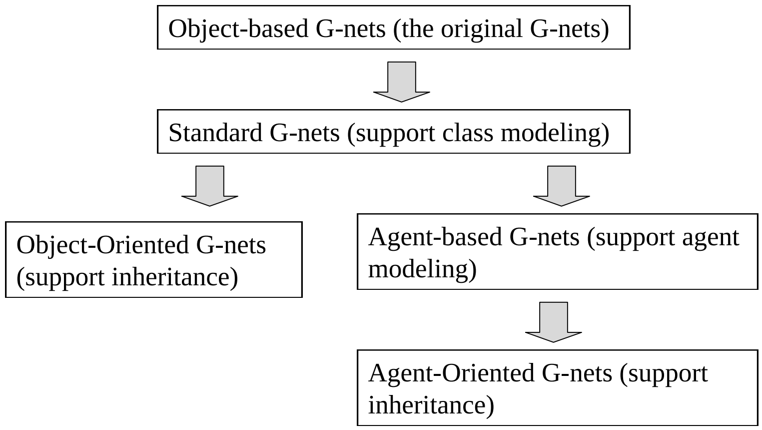
Agent-Oriented Software Engineering

- 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
  - mental state: plan, goal and knowledge
  - autonomous behavior
- 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

- 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



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 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 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

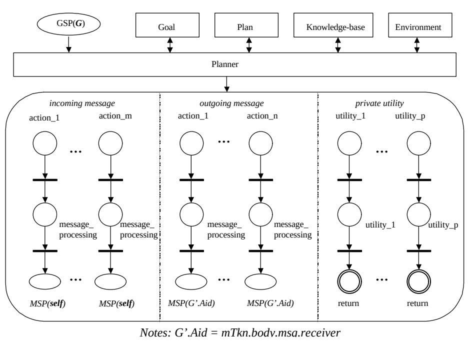


Figure 4. A generic agent-based G-net model

A Template of *Planner* Module

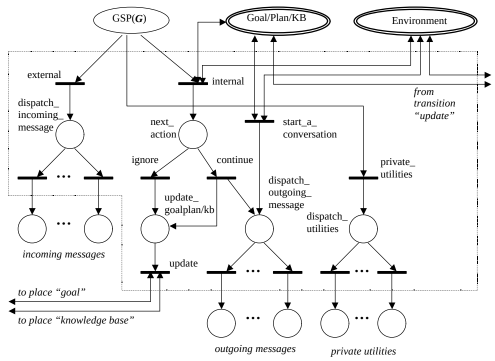


Figure 5. A template of *Planner* module

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

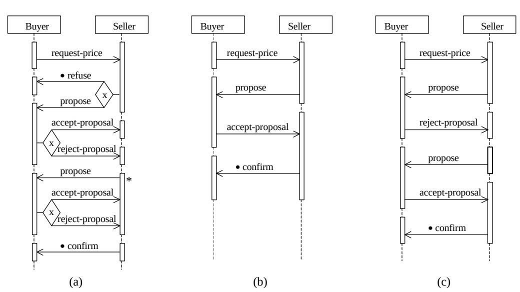
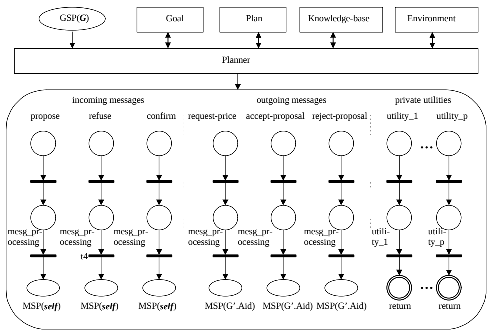


Figure 6. A contract net protocol between buying and selling agent

Selling and Buying Agent Design
(continue)



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

Figure 7. An Agent-based G-net model for buying agent

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)

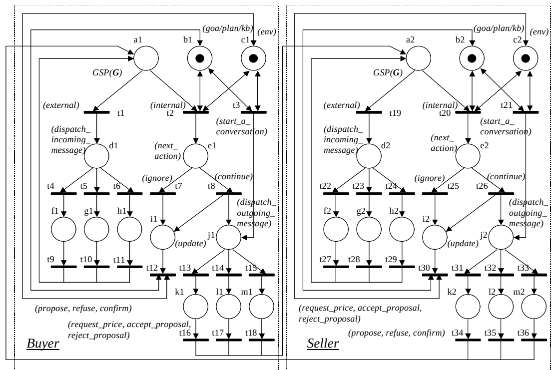


Figure 8. A transformed model of buying and selling agents

Part 3: A Framework for Modeling Agent-Oriented Software

- Extend existing methodologies:
  - object-oriented (OO) methodologies
  - 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 Agent-based Model

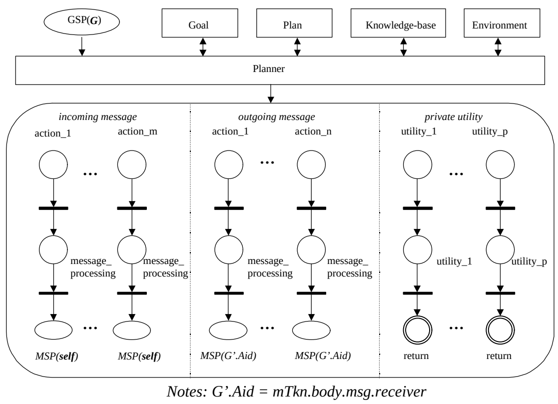


Figure 9. A generic agent-based G-Net model

Redesign of the *Planner* Module

- *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)

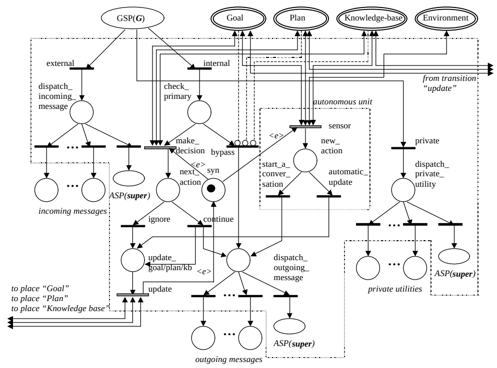


Figure 10. A template for the *Planner* module (initial design)

Examples of Agent-Oriented Design
(class hierarchy)

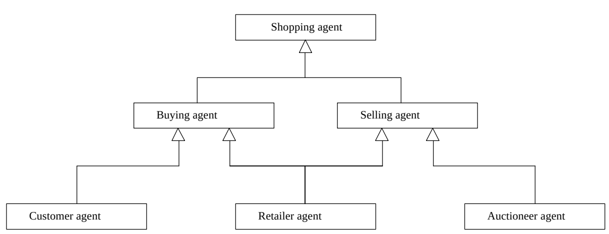


Figure 11. The class hierarchy diagram of agents in an electronic marketplace

Examples of Agent-Oriented Design
(contract net protocol)

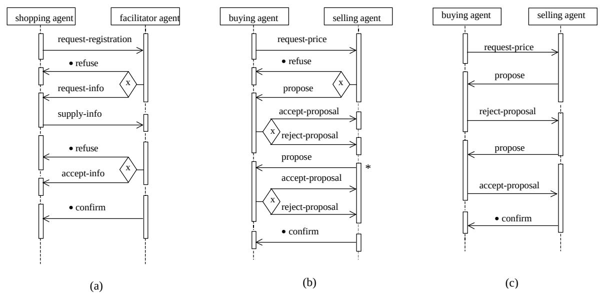


Figure 12. Contract net protocols (a) A template for the registration protocol (b) A template for the price-negotiation protocol (c) An example of the price-negotiation protocol

Examples of Agent-Oriented Design
(shopping agent class)

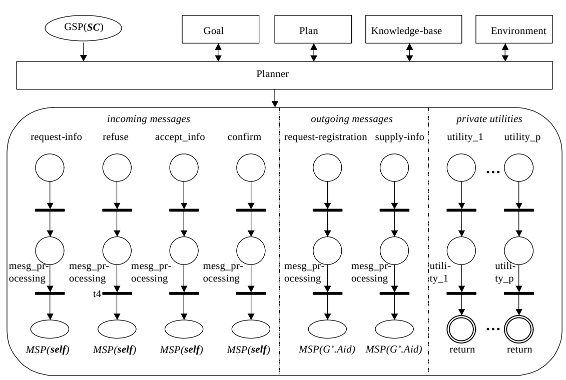


Figure 13. An agent-based G-Net model for shopping agent class

Examples of Agent-Oriented Design

(buying agent class)

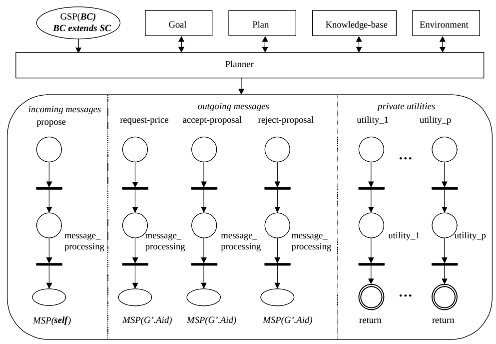


Figure 14. An agent-based G-Net model for buying agent class

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

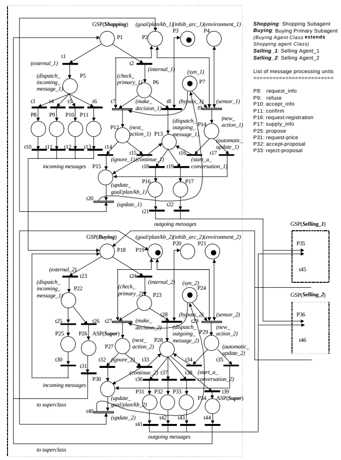


Figure 15. A transformed model of one buying agent and two selling agents

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

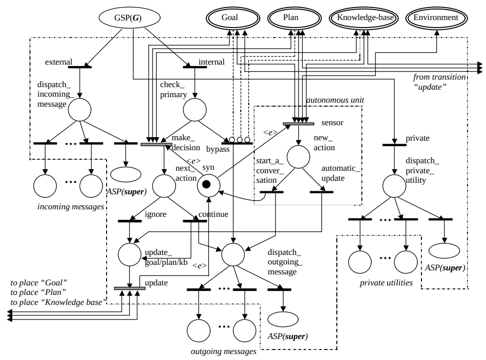


Figure 16. A template for the *Planner* module

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`

Part 5: Our Current Research Work

- Model intelligent mobile agents (IMA).
  - Introduce mobility into agent-oriented software model.
  - Provide a framework for intelligent mobile agent.
- Implement a model-based agent development prototype (Mad-Pro).
  - Use Jini middleware for agent communication.
  - Use the agent-oriented G-net model as guidelines for agent detailed design and implementation.

Overview of Agent Design Architecture

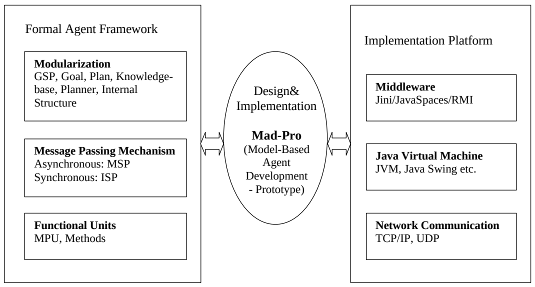


Figure 17. Overview of Agent Design Architecture

The Jini Community

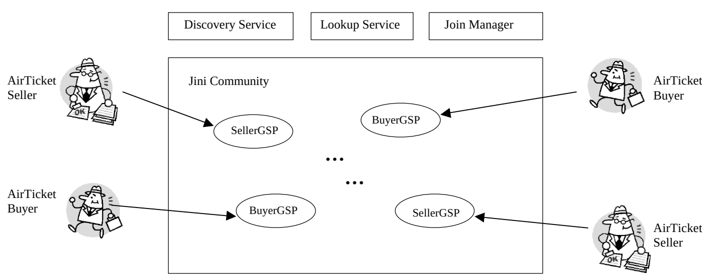


Figure 19. The Jini Community with Agents of AirTicketSeller and AirTicketBuyer

The Class Hierarchy of Agents in an Electronic Marketplace

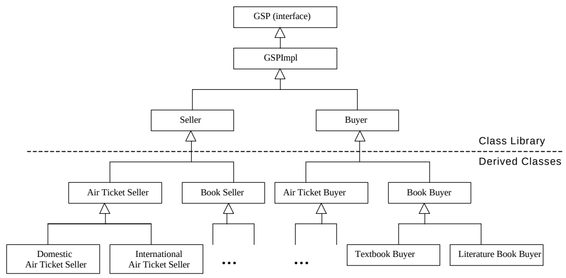


Figure 18.The class hierarchy diagram of agents in an electronic marketplace

Agent Interface Design

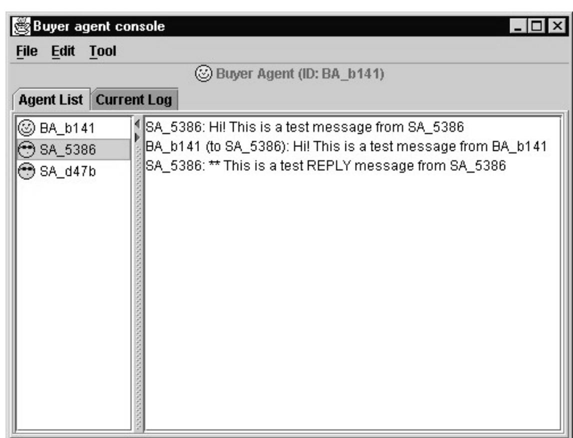


Figure 20. The agent interface for a buyer agent

Part 6: 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 agent-oriented design provides an effective way for modeling complex software systems.

Future Work

- Provide a class library for agent design.
- Define the agent communication language (ACL) in electronic commerce.
- Design and implement a compiler to automatically translate agent communication protocols into MPUs and decision-making units.
- Develop a model-based agent development environment (MADE) for rapid agent design and implementation.

References

- A. Perkusich and J. de Figueiredo, “G-nets: A Petri Net Based Approach for Logical and Timing Analysis of Complex Software Systems,” *Journal of Systems and Software*, 39(1): 39–59, 1997.
- N. R. Jennings, K. Sycara and M. Wooldridge, “A Roadmap of Agent Research and Development,” *International Journal of Autonomous Agents and Multi-Agent Systems*, 1(1): 7-38, 1998.
- H. Xu and S. M. Shatz, “An Agent-based Petri Net Model with Application to Seller/Buyer Design in Electronic Commerce,” *Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems (ISADS 2001)*, March 26-28, 2001, Dallas, Texas, USA, pp.11-18.
- H. Xu and S. M. Shatz, “A Framework for Modeling Agent-Oriented Software,” *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, April 16-19, 2001, Phoenix, Arizona, USA, pp.57-64.
- D. Kinny, M. Georgeff, and A. Rao, “A Methodology and Modeling Technique for Systems of BDI Agents,” In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modeling Autonomous Agents in a Multi-Agent World, (LNAI Volume 1038)*, pp. 56-71, Springer-Verlag: Berlin, Germany, 1996.
- C. A. Iglesias, M. Garrijo, J. Centeno-González, “A Survey of Agent-Oriented Methodologies,” *Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Language (ATAL-98)*, 1998, pp. 317-330.

The End

The copy of the slides for this lecture may be downloaded from
http://www.cs.uic.edu/~hxu1/Papers/Lecture542-2.PDF