

1-25-2011

# A Nested Petri Net Framework for Modeling and Analyzing Multi-Agent Systems

Lily Chang

Florida International University, chang9577@gmail.com

**DOI:** 10.25148/etd.FI11040601

Follow this and additional works at: <https://digitalcommons.fiu.edu/etd>

---

## Recommended Citation

Chang, Lily, "A Nested Petri Net Framework for Modeling and Analyzing Multi-Agent Systems" (2011). *FIU Electronic Theses and Dissertations*. 339.

<https://digitalcommons.fiu.edu/etd/339>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact [dcc@fiu.edu](mailto:dcc@fiu.edu).

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

A NESTED PETRI NET FRAMEWORK FOR MODELING AND ANALYZING  
MULTI-AGENT SYSTEMS

A dissertation submitted in partial fulfillment of the  
requirements for the degree of  
DOCTOR OF PHILOSOPHY

in  
COMPUTER SCIENCE

by  
Lily Chang

2011

To: Dean Amir Mirmiran  
College of Engineering and Computing

This dissertation, written by Lily Chang, and entitled A Nested Petri Net Framework for Modeling and Analyzing Multi-Agent Systems, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

---

Shu-Ching Chen

---

Peter J. Clarke

---

Malek Adjouadi

---

Xudong He, Major Professor

Date of Defense: January 25, 2011

The dissertation of Lily Chang is approved.

---

Dean Amir Mirmiran  
College of Engineering and Computing

---

Interim Dean Kevin O'Shea  
University Graduate School

Florida International University, 2011

## ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Dr. Xudong He for his intellectual guidance, generous advice, and kindly support during my studies. I would also like to thank my committee members, Dr. Peter J. Clarke, Dr. Shu-Ching Chen, and Dr. Malek Adjouadi, for their invaluable comments and advices on my dissertation. Thanks to the members of CADSE (Center for Advanced Distributed System Engineering) for their fruitful discussions and friendships.

Last but not least, I would like to thank my husband Eric for his understanding and support when I had to spend my time doing my Ph.D and not being available for him.

This dissertation research was partially supported by NSF grants HRD-0317692, HRD-0833093, IIP-0534428 and the Dissertation Year Fellowship of the University Graduate School at Florida International University.

ABSTRACT OF THE DISSERTATION

A NESTED PETRI NET FRAMEWORK FOR MODELING AND ANALYZING

MULTI-AGENT SYSTEMS

by

Lily Chang

Florida International University, 2011

Miami, Florida

Professor Xudong He, Major Professor

In the past two decades, multi-agent systems (MAS) have emerged as a new paradigm for conceptualizing large and complex distributed software systems. A multi-agent system view provides a natural abstraction for both the structure and the behavior of modern-day software systems. Although there were many conceptual frameworks for using multi-agent systems, there was no well established and widely accepted method for modeling multi-agent systems. This dissertation research addressed the representation and analysis of multi-agent systems based on model-oriented formal methods. The objective was to provide a systematic approach for studying MAS at an early stage of system development to ensure the quality of design.

Given that there was no well-defined formal model directly supporting agent-oriented modeling, this study was centered on three main topics: (1) adapting a well-known formal model, predicate transition nets (PrT nets), to support MAS modeling; (2) formulating a modeling methodology to ease the construction of formal MAS models; and (3) developing a technique to support machine analysis of formal MAS models using model checking technology. PrT nets were extended to include the notions of dynamic structure,

agent communication and coordination to support agent-oriented modeling. An aspect-oriented technique was developed to address the modularity of agent models and compositionality of incremental analysis. A set of translation rules were defined to systematically translate formal MAS models to concrete models that can be verified through the model checker SPIN (Simple Promela Interpreter).

This dissertation presents the framework developed for modeling and analyzing MAS, including a well-defined process model based on nested PrT nets, and a comprehensive methodology to guide the construction and analysis of formal MAS models.

## TABLE OF CONTENTS

CHAPTER	PAGE
1 INTRODUCTION .....	1
1.1 Motivation .....	1
1.2 Research Problems .....	3
1.3 Approach .....	4
1.4 Contributions .....	8
1.5 Scope and Limitations .....	10
1.6 Outline of the Dissertation .....	12
2 BACKGROUND AND RELATED WORKS .....	14
2.1 Multi-Agent Systems .....	14
2.2 Predicate Transition Nets .....	18
2.3 The Model Checker SPIN .....	21
2.4 Related Works .....	23
2.4.1 General Agent-Oriented Design Methodologies .....	23
2.4.2 Coordination Models .....	25
2.4.3 Aspect-Oriented Modeling .....	26
2.4.4 Specific Agent Modeling Languages .....	26
2.4.5 Petri Nets .....	29
3 A NESTED PETRI-NET FRAMEWORK .....	33
3.1 An Overview of the Framework .....	35
3.1.1 The Conceptual Model .....	36
3.1.2 The Formal MAS Model .....	37
3.1.3 The Concrete Model .....	39
3.2 The Underlying Formalism .....	40
3.2.1 Two-Level Nested Predicate Transition Nets .....	40
3.3 Net Representations and Semantics .....	44
3.3.1 Agent Nets .....	45
3.3.2 The Mediator Agent Net .....	47
4 A METHODOLOGY FOR MODELING MULTI-AGENT SYSTEMS .....	55
4.1 Agent-Oriented Modeling .....	56
4.2 Aspect Orientation of the Internal Structure of an Agent Net .....	58
4.2.1 Specifying Aspects .....	60
4.3 Modeling a Single Agent Net .....	63
4.3.1 Modeling Examples .....	64
4.3.2 Constructing an Agent Net with the BDI Model .....	70
4.4 Coordination Modeling .....	73
4.4.1 Coupling Agent Nets .....	74
4.5 Summary .....	78

5	A METHOD FOR ANALYZING FORMAL MAS MODELS .....	80
5.1	The Target Language PROMELA .....	82
5.1.1	The Program Structure .....	82
5.1.2	The Semantics .....	83
5.2	A Translation Method .....	84
5.2.1	Translation Rules .....	86
5.2.2	Model Transformation .....	96
5.3	Correctness of the Translation.....	97
5.3.1	Completeness .....	98
5.3.2	Consistency .....	99
5.4	A Translation Example.....	101
5.5	Model Analysis .....	102
5.5.1	Encoding System Properties in PROMELA .....	103
5.5.2	Model Checking the MAS Model.....	110
5.6	Summary .....	112
6	CASE STUDY .....	114
6.1	Wireless Sensor Network with Mobile Devices .....	114
6.1.1	The Interest-Management Server Model .....	116
6.1.2	The Mobile Device Model .....	119
6.1.3	A Query Request Scenario.....	124
6.1.4	Discussion .....	128
6.2	E-Market.....	129
6.2.1	Discussion .....	133
6.3	Disaster Mitigation .....	134
6.3.1	Modeling Interactions in an Agent Net.....	136
6.3.2	Modeling Interactions in the System Net.....	138
6.3.3	Soundness of BCIN Net.....	142
6.3.4	Discussion .....	143
7	CONCLUSION.....	144
7.1	Contributions.....	145
7.2	Limitations .....	146
7.3	Future Works.....	148
	REFERENCE.....	151
	APPENDICES .....	159
	VITA.....	170



## LIST OF TABLES

TABLE	PAGE
1. A comparison of this dissertation and other Petri-net-based works.....	10
2. Interaction elicitation table.....	67
3. The coordination elicitation table.....	76
4. Mapping relations of net elements to PROMELA objects.....	87
5. Place translation rules.....	90
6. Transition translation rules.....	93
7. Encoding system properties.....	109
8. Statistics of model transformation.....	110
9. The server's actions.....	117
10. A mobile device's actions.....	120
11. Communication channels.....	128
12. Actors and their associated roles and actions.....	136

## LIST OF FIGURES

FIGURE	PAGE
1. The overall framework .....	7
2. (a) The problem domain; (b) the solution domain.....	11
3. Components of the nested Petri net framework .....	36
4. MAS components: a conceptual model .....	37
5. An upper level view of a nested PrT net with input/output channels and net tokens.....	42
6. (a) Pro-activeness of an agent net; (b) reactivity of an agent net .....	46
7. Net representation of an output communication channel .....	48
8. Net representation of an input communication channel .....	49
9. Transportation .....	50
10. Choice; (a) output channels; (b) input channels .....	51
11. Synchronization .....	52
12. Instantiation .....	52
13. Cooperation .....	52
14. A control of resources .....	53
15. Replication.....	54
16. (a) Agent-oriented modeling; (b) essential modeling steps.....	57
17. A conceptual model for an aspect-oriented agent net .....	59
18. (a) An aspect specification table; (b) An aspect weaving process .....	62
19. Weaving patterns .....	63
20. Modeling steps for an agent net with communication channels.....	64

21. Agent nets in the gas station scenario .....	67
22. (a) The Pumping gas aspect; (b) the Car wash aspect.....	69
23. (a) The basic net for a Regular_consumer agent; (b) the woven net.....	70
24. John's traveling plan .....	71
25. (a) An input channel coordinator; (b) an output channel coordinator .....	74
26. (a) The steps for generating CET; (b) the steps for generating the mediator agent net.....	76
27. (a) The coordinators of the Diesel_consumer agent net; (b) the intermediate net; (c) the mediator agent net.....	77
28. A generic PROMELA program structure.....	82
29. The operational model of the PROMELA semantics engine .....	84
30. A constraint formula translation example .....	97
31. The BCIN net .....	102
32. The PROMELA codes for a never claim .....	107
33. The verification report for the never claim .....	108
34. The verification output with non-progress cycle detected .....	111
35. The sequence chart of agent interactions in a simulation run .....	112
36. Message flows between the entities in a WSN with mobile devices .....	115
37. The behavior model of the information server.....	118
38. The mobile device's behavior model.....	123
39. (a) Group address multicast; (b) cluster head data communication .....	124
40. The seller's and buyer's interaction model .....	131
41. The broker's interaction model .....	132
42. An algorithm for generating a role model.....	137

43. A role model for a supervisor role.....	138
44. The algorithm for generating a RC net.....	141
45. A resource acquisition operation.....	141

## LIST OF ACRONYMS

AI	Artificial Intelligence
AML	Agent Modeling Language
AOP	Aspect-Oriented Programming
AOSE	Agent-Oriented Software Engineering
AUML	Agent Unified Modeling Language
BCIN	Business Continuity Information Network
BDI	Belief-Desire-Intention
CPN	Colored Petri Nets
CSP	Communicating Sequential Processes
CTL	Computation Tree Logic
DAI	Distributed Artificial Intelligence
FIPA	Foundation for Intelligent Physical Agents
FSA	Finite State Automaton
GPS	Global Positioning System
IT	Information Technology
LTL	Linear Temporal Logic
MAS	Multi-agent Systems
MASIF	Mobile Agent System Interoperability Facilities
MAS-ML	Multi-Agent System Modeling Language
MULAN	Multi-Agent Nets
OMG	Object Management Group
PROMELA	Process Meta Language

PrT Nets	Predicate Transition Nets
SMV	Symbolic Model Verifier
SPIN	Simple PROMELA Interpreter
TAO	Taming Agents and Objects
UML	Unified Modeling Language
WSN	Wireless Sensor Network
Z	Z Notation

# **CHAPTER 1**

## **INTRODUCTION**

This dissertation presents a framework for modeling and analyzing multi-agent systems based on model-oriented formal methods. This chapter gives an overview of the dissertation. Section 1.1 introduces the motivation of this dissertation research. Section 1.2 identifies the research problems. Section 1.3 presents the approach to address the research problems. Section 1.4 summarizes the contributions of this dissertation. Section 1.5 discusses the scope and limitations. Section 1.6 gives an outline of the dissertation.

### **1.1 Motivation**

Multi-agent systems (MAS) [1] have been intensively studied in the distributed artificial intelligence (DAI) community to address distributed problem solving [2] that involves the collective efforts of multiple agents. There are two major motivations for distributed problem solving: (1) utilizing distributed resources concurrently in order to speed up a problem solving process; and (2) integrating problem solving capabilities that are geographically distributed when a centralized approach is not possible [2]. The research topics were primarily on distributed planning and coordination in which the concerns were on system-wide coherence and conflict controls over resources. In the past two decades, however, MAS has emerged as a paradigm in the software engineering community for structuring complex systems running in an open computing environment [3]. An important idea of using the MAS view for system design is to decompose a complex problem into a number of functionally specific modular components (agents) that are specialized at solving a particular problem aspect [4]. That is, the MAS

architecture offers the *modularity* to tackle a complex problem. The underlying concept of MAS is the use of *agent* as a key abstraction for system design. In the MAS context, agents represent heterogeneous computation entities that are geographically distributed and have the properties of *autonomy*, *reactivity*, *pro-activeness* and *social ability* [5, 6]. These properties differentiate an agent from an object in terms of conceptualization. That is, an *agent* performs the actions of its best interest as opposed to an *object* that performs the actions invoked by the others. It is obvious that the traditional design approach based on the object-oriented paradigm is inadequate for conceptualizing complex systems with the MAS architecture.

Many research activities have been conducted to develop the languages, methodologies and tools for conceptualizing complex systems based on the MAS view [7]. Previous works in this regard were centered on agent-oriented software engineering (AOSE) [8], with a focus on the following research topics: (1) requirement engineering; (2) design, specification and verification techniques; (3) ontologies; (4) generic agent models and design patterns; (5) validation and testing techniques; and (6) tools for MAS development. The objective of AOSE is to establish a systematic approach for developing complex systems based on the MAS architecture. The AOSE approach has been applied to a diverse range of information technology (IT) sub-disciplines, such as information retrieval, control systems, e-commerce, mobile agent systems, workflow management and grid computing. In recent years, disaster management [9, 10, 11, 12] has also become an active application area due to its socially significant nature. Although the emergence of AOSE [8] is well recognized as a significant contribution for the conceptualization of



developing MAS, it is still in its infancy towards a mature engineering paradigm to be widely adopted by the software engineering community.

Despite many studies in developing design methods based on MAS [7], few of them were in formal methods. In addition, there is no well established and widely accepted method for the representation of MAS. However, formal methods are an important means to study the critical aspects of the system and to reveal system flaws at an early stage of the development process [13]. It is well recognized that the design flaws or missing requirements found at a later stage are more expensive to fix. In response to the efforts in developing complex systems by applying the AOSE approach, this work aims at the representation and analysis of MAS based on model-oriented formal methods. A model-oriented formal method describes a system's behaviors by constructing a model in terms of mathematical structures [13]. Therefore, the model has a well-defined semantics, and is amenable for machine analysis. The objective of this study is to provide a systematic approach for constructing MAS specifications, which can be analyzed using existing model checking techniques [14, 15].

## **1.2 Research Problems**

This dissertation research focuses on applying formal methods at an early stage of a system development process in an effort to ensure the dependability of complex systems. The idea is to formulate a systematic approach for modeling and analysis of complex systems with the MAS view based on model-oriented formal methods. Therefore, the research problems being explored include the modeling of MAS in formal specification languages, and the techniques for MAS analysis. Modeling is a process of building an abstraction for a system. Modeling a complex system based on the MAS view addresses

the complexity by decomposing the system into multiple agents that can be built and analyzed independently. In order to properly apply the MAS paradigm, the modeling language needs to support the representation of individual agent with essential characteristics [16], as well as the representation of the MAS architecture.

Given that there is no well-defined formal model directly supports agent-oriented modeling, the idea is to find a suitable formal model and adapt it to support MAS modeling. In addition, the resulting models should be amenable for machine analysis by applying existing model checking techniques. Therefore, the research problems have been centered on three main topics:

- (1) Investigating a suitable formal model and extending the formal model to support MAS modeling; the extended formal model should be able to address the essential characteristics [5] of MAS, including: (i) individual agent modeling with essential properties, such as autonomy, reactivity, pro-activeness, and social ability; (ii) MAS architecture modeling with a compositional agent organization; and, (iii) coordination modeling with agent interactions.
- (2) Formulating a modeling methodology based on the extended formal model to ease the construction of MAS models; the methods should be easy to apply and produce adaptable and concise MAS models.
- (3) Developing a model analysis technique to support machine analysis of formal MAS models using model checking technology [14, 15].

### **1.3 Approach**

Among model-oriented formal methods, predicate transition nets (PrT nets) [17] are a powerful formal model for studying distributed and concurrent systems. PrT nets not only

are a visual specification language providing intuitive notations, but also very expressive in encapsulating meaningful and important data. More important, PrT nets have a precise operational semantics, which can be manipulated for machine analysis. Therefore, PrT nets are chosen as the underlying formalism in this study to address the representation of MAS. Despite the expressiveness, PrT nets have a flat net structure that provides only a single-level view of the system model. Thus, they are inadequate for modeling MAS architecture. Another drawback of PrT nets in terms of modeling MAS is that they do not have the element for modeling agent communications. Therefore, to address the research problem (1) identified in Section 1.2, PrT nets need to be extended to cope with agent-oriented modeling. My approach to address the representation of MAS by extending the semantics of PrT nets is summarized as follows.

- (i) Incorporate the channel command in CSP (Communicating Sequential Process) [18] into the transition constraints of PrT nets for modeling agent communications. A channel command is used as an expression in a constraint formula indicating a unidirectional information flow, either an input message or an output message, between two agent nets. A channel expression is composed of a channel name, the direction of an information flow and a message token.
- (ii) Adapt the net-within-net paradigm [19, 20] and define a two-level nested PrT net to address the MAS architecture. A nested PrT net includes an upper level net modeling either a mediator agent net or a system net, and multiple lower level nets modeling heterogeneous agents. The coupling of the upper level net and a lower level net is through a pair of transitions with channel expressions. The channel name

specified in a channel expression is used to define the vertical communication between the upper level net and a lower level net.

- (iii) Introduce an agent model incorporating aspect-oriented concepts [21] to address the modularity of the internal structure of an agent net. The idea is to define agent features as aspects [22]. An aspect is a modular net that can be woven into the agent net based on requirements. The aspect specification and aspect weaving are formally defined.

To address the research problem (2) identified in Section 1.2, a modeling methodology is developed to systematically transform a MAS conceptual model into a nested PrT net. Given that the model checker SPIN (Simple PROMELA Interpreter) [23] is a well developed tool for model checking distributed systems with asynchronous process interactions, SPIN is chosen as the analysis tool for MAS model analysis. The idea is to transform a nested PrT net describing a multi-agent system into a PROMELA program for verification. As a consequence, a full line of functionalities in the model checker SPIN can be utilized to analyze the PrT net model. A model transformation technique is developed to tackle the research problem (3). The translation rules for model transformation from a nested PrT net to a PROMELA program are explicitly defined.

The overall framework can be depicted in Figure 1. Given a conceptual model identifying the agent roles within a multi-agent system, a nested PrT net model based on a two-layered MAS architecture can be built, and transformed into a PROMELA program in SPIN. The framework supports MAS modeling in a model construction process with three different stages in which each stage produces a transformational model that leads to a model of the next stage. First, the MAS model construction process starts with a

*conceptual model* resulting from requirement elicitation activities. Next, the conceptual model can be transformed into a *nested PrT net model* describing the MAS. Finally, the nested PrT net model is transformed into a *concrete model* in PROMELA. A methodology is developed to systematically transform the model at one stage to the other model at the next stage, including an agent-oriented modeling method, an aspect-oriented modeling method, an agent coordination modeling method, and a model transformation method.

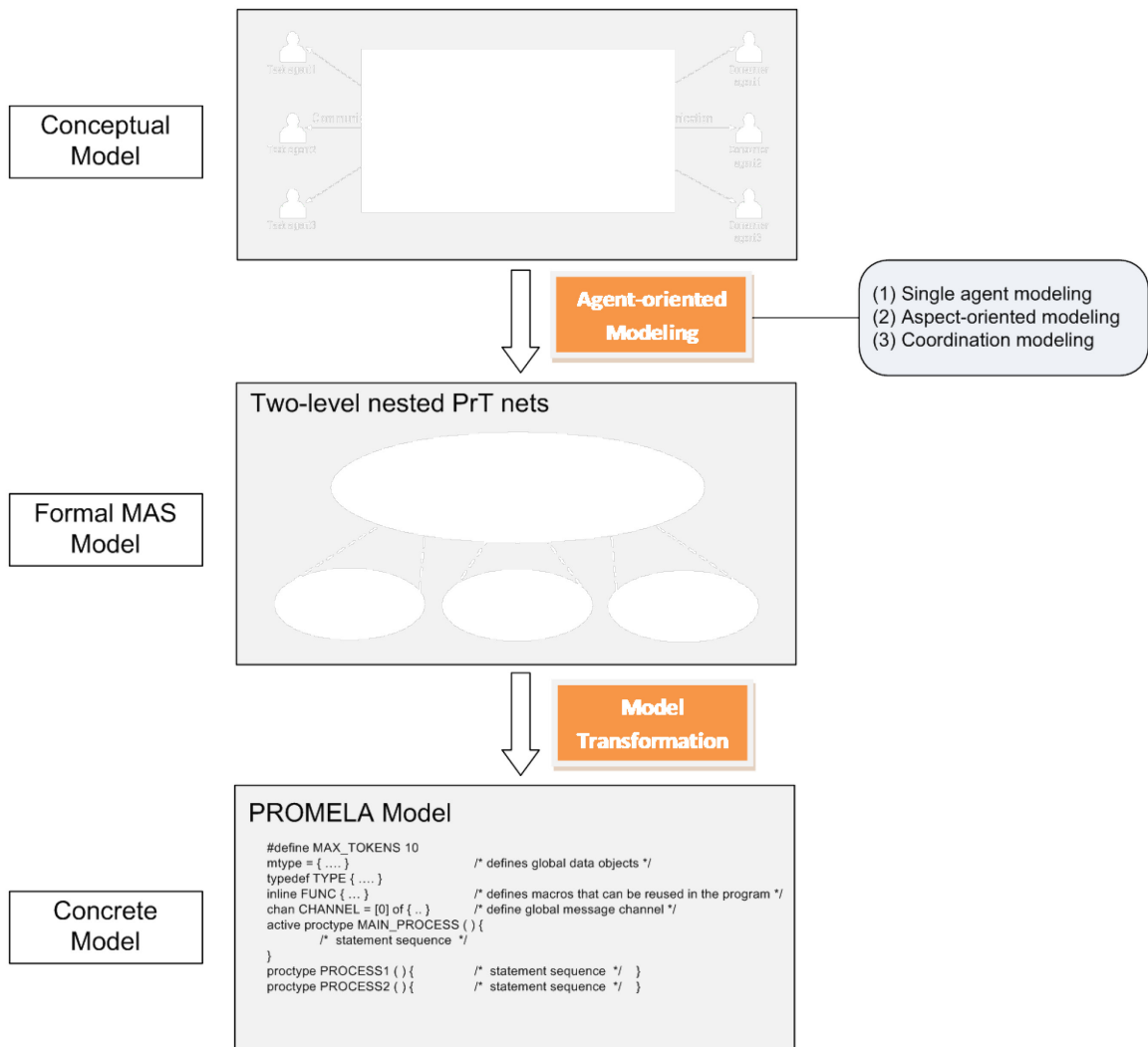


Figure 1. The overall framework.

## 1.4 Contributions

The framework presented in this dissertation provides a comprehensive methodology for modeling and analyzing complex systems based on the MAS view. The major contributions are summarized as follows.

(1) *Developed a well-defined process model for the representation of multi-agent systems.*

- Support agent-oriented modeling in which the system model is a modular composition of multiple nets instead of a single net; that is, this dissertation achieves the *modularity* of PrT nets, which is an important means to address complexity.
- Support the MAS view in which the system model has a two-level nested net structure following the two-layered MAS architecture. The upper level net models the coordination among agents, while the lower level net models heterogeneous agents. A nested net structure has two advantages: (i) a more compact and manageable net model by separating the global and local view, and (ii) can naturally model agent *autonomy* and *mobility* by treating agent nets as tokens at the upper level net.
- Support the modeling of dynamic agent communications by incorporating the channel commands in CSP (Communicating Sequential Processes) [18] into the transition constraints in PrT nets. As a consequence, agent nets can be individually modeled and loosely coupled through communication channels. The new feature naturally models the *reactivity* and *pro-activeness* of agents, and supports the modeling of agent *interactions*.

(2) *Developed a methodology for constructing multi-agent net models.*

- Provide an aspect-oriented technique for constructing individual agent nets with different features. As a result, common behaviors of agents can be modularized into features and woven as required. Aspect-oriented PrT nets have the following advantages: (i) enhancing the *adaptability* of behavior models in future modifications, extensions and reconfiguration of agent models; (ii) promoting the *reusability* of aspects; (iii) improving the *conciseness* of agent nets; and (iv) facilitating the *compositionality* of agent models for incremental verification.
- Provide a technique for agent *coordination* modeling to ease the construction of MAS models.

(3) *Developed a model transformation technique for model checking multi-agent models.*

- Provide a technique that systematically transforms a nested PrT net to a PROMELA program in SPIN for automatic *model analysis*. As a result, a full line of functionalities in SPIN can be utilized for model analysis.

Table 1 shows a comparison of this work to the most relevant works using high-level Petri nets to model agent-based systems. First column summarizes the contributions of this dissertation research. Most of the works listed in the table did not provide a comprehensive approach for MAS modeling and analysis, and only addressed a certain aspect of agent-oriented modeling. My work provides a comprehensive approach from MAS modeling to analysis, and covers essential agent properties within the MAS context.

Table 1. A comparison of this dissertation and other Petri-net-based works.

Contributions	This dissertation	Colored Petri nets Ref. [24, 25, 26] Cabac, Kohler	PrT nets Ref. [27 ] D. Xu	PrT nets Ref. [28] Ding	G-nets Ref. [29] H. Xu	Colored Petri nets Ref. [30 ] Lian
Dynamic structure (nets-within-nets)	✓	✓	✓	✓		
Agent Communication	(1) channel commands in transition constraints (2) formal semantics	(1) based on OO; (2) transitions (channels) are method calls	through transitions (called connectors)	through transitions (no formal semantics)	message passing through special places	
Coordination modeling	Coordinators					Potential arcs
Aspect-oriented agent models	✓					
Tool support for analysis	SPIN (model checking tool)	RENEW (simulation tool)				CPN (simulation tool)
within MAS context	✓	✓	mobile agents	mobile agents	✓	✓

## 1.5 Scope and Limitations

This dissertation research is focused on investigating model-oriented formal methods [13] for modeling and analyzing software systems based on the MAS design paradigm. The formal model used is PrT nets, and the analyzing tool is the model checker SPIN. The modeling methodology in this dissertation research is based on the agent concept defined in AOSE [8, 31], and is targeted at constructing models for large and complex systems relying on the MAS architecture. The scope is depicted in Figure 2.



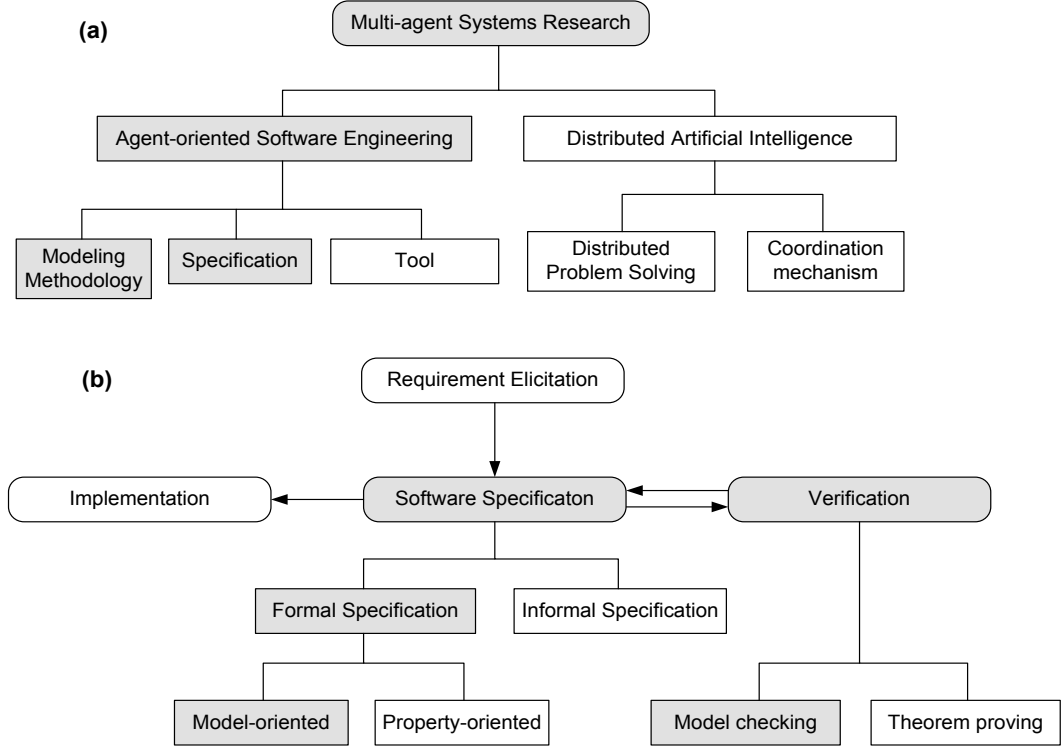


Figure 2. (a) The problem domain; (b) the solution domain.

Since the objective of this dissertation research is to provide a means for modeling and verifying MAS at the system design stage, the principle is to avoid unnecessary redundancies, and build a model without irrelevant details for machine analysis. Therefore, this dissertation research is developed based on the following assumptions.

- (1) The MAS model is focused on the interdependencies between agent nets, rather on the computations.
- (2) The research is aimed at the construction of abstract models based on the MAS paradigm using model-oriented formal methods. The detail of how to implement the search algorithm for solving a problem efficiently using an agent program [32] is outside of the scope of this dissertation research.

- (3) The nested PrT nets defined in this study is limited to a two-level nested net structure, which is adequate for modeling complex systems with the MAS architecture.
- (4) Agent communications are in a one-to-one and unidirectional fashion.
- (5) There are limitations for model checking nested PrT nets: (i) Due to tractability, the expressive power of PrT nets is restricted by limiting the sorts and the quantity of tokens in each place. As a result, the nested net to be analyzed have a finite state space such that the model execution in the model checker SPIN will terminate appropriately. (ii) For simplicity, all transitions fire immediately after the guard conditions are evaluated to be true, and the firings are interleaving given that this restricted semantics does not affect the verification of state-based properties. (iii) The properties can be checked are restricted within the scope of the model checker SPIN with XSpin version 5.2.3.

## **1.6 Outline of the Dissertation**

The rest of the dissertation is organized as follows.

Chapter 2 reviews the theoretical background of this dissertation, including multi-agent systems, Petri nets, predicate transition nets and the model checker SPIN. The contributions that are most related to this dissertation research are also discussed.

Chapter 3 presents a nested PrT net framework that provides the theoretical foundation of this dissertation research, including the underlying formalism, and the overall approach to tackle the research problems.

Chapter 4 introduces a modeling methodology to guide the construction of agent models and the mediator agent model that coordinates agents. An aspect orientation

technique is also presented in this chapter to address the modularity of agent models and compositionality of incremental analysis.

Chapter 5 discusses the verification of formal MAS models. A model transformation technique is developed for model checking nested PrT nets. This chapter introduces the methodology to systematically transform the conceptual model of multi-agent systems to a concrete model that can be automatically verified using the model checker SPIN.

Chapter 6 demonstrates three case studies in different application domains, including wireless sensor network with mobile devices, e-market and disaster mitigation.

Chapter 7 concludes this dissertation research.

## **CHAPTER 2**

### **BACKGROUND AND RELATED WORKS**

This dissertation research is concerned with the representation and analysis of multi-agent systems based on high-level Petri nets (Figure 2(a) and (b)). This chapter is organized as follows. Section 2.1 discusses the research in multi-agent systems. Section 2.2 introduces Petri nets and gives a formal definition of predicate transition nets. Section 2.3 presents the characteristics of the model checker SPIN. Section 2.4 discusses the related works.

#### **2.1 Multi-Agent Systems**

Multi-agent systems [1, 4] (MAS), stem from the Distributed Artificial Intelligence (DAI) community [2], is the study of distributed problem solving for large and complex systems. Since MAS offer the modularity for structuring complex systems, it has emerged as a new paradigm in the software engineering community for conceptualizing, designing and implementing dynamic software systems. The idea of using MAS for conceptualizing a complex system is to decompose the problem space into a number of functionally specific components (agents) that can be tailored to solve a particular problem. As a result, several dimensions of performance can be enhanced, such as computational efficiency, reliability, extensibility, maintainability, flexibility and reusability [33]. The research in system design based on the MAS paradigm is concerned with the construction of a collection of possibly preexisting autonomous agents that interact with each other and their environments. Therefore, MAS can be defined as a loosely coupled network of autonomous agents that cooperate to solve a problem, and is

characterized as follows: (1) each agent has incomplete information or capabilities for solving the problem; (2) there is no global control; (3) data are decentralized; and (4) the computation is asynchronous. The motivations for the increasing interest in structuring a complex system with the MAS architecture include: (1) increasing the efficiency, reliability and robustness of the system due to a large and complex problem space; since a centralized approach produces bottlenecks that lead to a system failure [4]; (2) interconnecting preexisting components that were designed by different organizations; and (3) retrieving and synthesizing spatially distributed information resources [4].

There are several application areas [1] found to be suitable for the study of MAS: (1) *Workflow and business process management*. A workflow is an automated process that coordinates services from heterogeneous organizations over the Internet. Each step in a workflow corresponds to a decision that must be made by some computational process based on the states of the environment. An agent is usually used to model a provider, a requestor, a service, or an executor in a workflow process. (2) *Distributed sensing systems*. In a distributed sensing system, spatially distributed sensors are deployed to monitor environmental conditions, such as temperature or traffic. Typically, sensor nodes are scattered in a region to collect data cooperatively. Some sensor systems incorporated mobile devices into the system to gather sensor data effectively [34] by utilizing the storage and computation power of mobile devices. Mobile devices and sensor nodes are typically modeled as agents that cooperate for gathering and sharing data. (3) *Information retrieval and management*. In light of a distributed computation environment, information resources are usually spread over the internet. An information agent is designed to act on behalf of a user to manipulate the information obtained from various

sources, and providing the user a unified way to access the data obtained. (4) *Electronic commerce*. The application domain is concerned with the business automation over electronic systems such as the Internet. For instance, buyer agents and seller agents auction for goods through an e-market on behalf of the users. (5) *Industrial system management and control systems*. Supply chain management and production management are typical industrial systems for the study of MAS. These systems involve the process control between agents that belong to different organizations. Control systems, such as air traffic management, involve the resource control among autonomous agents through some negotiation mechanisms to avoid conflicts.

In light of the growing interest in MAS, agent-oriented software engineering [8] has emerged as a new approach for engineering complex software systems. The main idea is to use the *agent* as a key abstraction for conceptualizing complex systems based on the MAS architecture. In [8] and [4], an *agent* is defined as *an encapsulated computer system that is situated in some environment and capable of flexible, autonomous actions in that environment in order to meet its design objectives*. By flexible and autonomous, agents can take the initiative actions, retrieve current environment information, react based on the information, and achieve their design goals. In [35], an agent was defined as *a component of software, which acts on behalf of its user to accomplish tasks*. Software agents were classified by their functionalities or exhibited behaviors into seven types: collaborative agents, interface agents, mobile agents, information agents, reactive agents, hybrid agents and smart agents. In summary, the essential properties of an agent are *autonomy*, *reactivity*, *pro-activeness* and *social ability*. By autonomy, agents have their own computation logic and various degree of intelligence to perform actions without the

direct intervention of humans; by reactivity, agents are able to perceive situated environment and respond in a timely fashion; by pro-activeness, agents are able to take the initiative to perform goal-directed tasks; by social ability, agents are capable of interacting with others within an agent society. Agent interactions include (i) *communications*, in which agents exchange messages; (ii) *cooperation*, in which agents work together towards a common goal; and (iii) *coordination*, in which agents are coordinated to avoid conflicts when sharing resources. The objective of AOSE is to formulate a systematic approach to effectively develop software systems based on the MAS architecture. The research area includes: (1) requirement engineering; (2) design, specification and verification techniques; (3) ontologies for agent models; (4) generic agent models and design patterns; (5) validation and testing techniques; and (6) modeling and simulation tools for MAS development.

Although many research activities have been conducted to develop languages, methodologies and tools based on AOSE, the methodologies [7] developed so far did not cover enough details of how to represent the system and an agent. There is still lack of a well established and widely accepted method for representation of MAS. Moreover, there are several essential issues on applying the MAS paradigm [36] for complex system design. First, how to specify an agent with the reasoning capability for external information and with the coordinating capability in a simultaneously participating event is a challenge issue. Second, agent communications involve high level knowledge exchange, which is usually domain specific and context-based. For example, what are the communication languages and protocols, and how can heterogeneous agents communicate and interoperate. Third, agents are heterogeneous and autonomous, how to

ensure that agents act coherently in making decisions and how to ensure the overall system consistency are also challenging issues.

## 2.2 Predicate Transition Nets

Petri nets are a model-oriented modeling language and were first developed by Carl Adam Petri in 1962. Petri nets are a graphical and mathematical modeling tool that is excellent in describing the controls of dynamic systems. A Petri net can be formally defined as follows [37].

A Petri net is a 5-tuple  $(P, T, F, W, M_0)$  where:

- $P$  is a finite set of places.
- $T$  is a finite set of transitions.
- $F \subseteq (P \times T) \cup (T \times P)$  is a set of arcs (flow relations).
- $W: F \rightarrow \{1, 2, 3, \dots\}$  is a weight function.
- $M_0: P \rightarrow \{0, 1, 2, 3, \dots\}$  is the initial marking.
- $P \cap T = \emptyset$  and  $P \cup T \neq \emptyset$

The change of markings denoting the dynamic behaviors of the net is according to the following transition rules:

- A transition  $t$  is said to be enabled if each input place  $p$  of  $t$  has at least  $w(p, t)$  tokens, where  $w(p, t)$  is the weight of the arc from  $p$  to  $t$ .
- An enabled transition may fire.
- A firing of an enabled transition  $t$  remove  $w(p, t)$  tokens from each input place  $p$  of  $t$ , and add  $w(t, p)$  tokens to each output place  $p$  of  $t$ , where  $w(t, p)$  is the weight of the arc from  $t$  to  $p$ .



Despite their use in a wide variety of applications [37], Petri nets are limited to specifying rather small systems or certain aspects of a system since the net structure can become very large and complex when specifying a large system. The tangling net structure results in poor readability and flexibility. As a consequence, many extended versions, including hierarchical Petri nets, timed Petri nets and stochastic Petri nets, were proposed to enhance the comprehensibility of Petri nets. These extensions are called high-level nets. Identical net structures in a low-level Petri net are simply replaced by a single transition or a place in a high-level net. Thus, the structural complexity in low-level nets can be greatly reduced. Moreover, tokens in a high level net can be individually defined and differentiated; they are amenable to encapsulate important information and meaningful data.

This dissertation research adopted predicate transition nets [17] (PrT nets) as the underlying formalism. PrT nets are high-level nets and defined to be manipulated in a mathematical way to working with logical formulas or algebraic expressions. The net structure of a PrT net can be conceived as a set of components that is structured by functions and relations. Since both data and controls can be addressed, PrT nets are more concise and expressive for modeling concurrent systems. A PrT net is formally defined as follows.

*Definition 2.2.1* A PrT net is a tuple  $(N, Spec, ins)$ , where:

- (1)  $N = (P, T, F)$  is a net structure.  $P$  and  $T$  are finite sets of places and transitions of  $N$ , where  $P \cap T = \emptyset$ ,  $P \cup T \neq \emptyset$ ; and  $F \subseteq (P \times T) \cup (T \times P)$  is a set of arcs, which define the flow relations;

(2) *Spec* is an algebraic specification, which includes sorts, operators, and equations.

Terms defined in *Spec* include tokens in  $P$ , labels on  $F$  and constraints associated with  $T$ ; and

(3)  $ins = (\varphi, L, R, M)$  is an inscription that maps net elements to their denotations in the algebraic specification *Spec* where:

- $\varphi: P \rightarrow \wp(SORT)$ , is the token type definitions of  $N$ ;
- $L: F \rightarrow Label_{sort}(X)$ , is a sort-respecting mapping from  $F$  to the set of labels  $Label_{SORT}(X)$  where  $X$  is the set of sorted variables;
- $R$  is a mapping from  $T$  to the set of constraints, which are a set of first order logic formulas defined in *Spec*, and  $R$  associates each transition  $t$  in  $T$  with a first-order logic formula that defines the meanings of transition  $t$ ;
- $M_0$  is the sort-respecting initial marking that assigns a multi-set of tokens to each place  $p$  in  $P$ .

The dynamic semantics of a PrT net can be defined as follows:

(1) A marking of a PrT net is a mapping from  $P$  to sorts defined in *Spec*;

(2) An occurrence mode of  $N$  is a substitution  $\alpha = \{ x_1 \leftarrow c_1, \dots, x_n \leftarrow c_n \}$ , which instantiates typed label variables.  $e: \alpha$  is used to denote the result of instantiating an expression  $e$  with  $\alpha$ , in which  $e$  can be either a label expression or a constraint;

(3) Given a marking  $M$ , a transition  $t \in T$ , and an occurrence mode  $\alpha$ ,  $t$  is  $\alpha$ -enabled at  $M$  iff the following predicate is true:  $\forall p: p \in P. (\bar{L}(p, t): \alpha \subseteq M(p)) \wedge R(t): \alpha$ , where

$$\bar{L}(x, y) = \begin{cases} L(x, y) & \text{if } (x, y) \in F \\ \emptyset & \text{otherwise} \end{cases}$$

- (4) If  $t$  is  $\alpha$ -enabled at  $M$ ,  $t$  may fire in occurrence mode  $\alpha$ . The firing of  $t$  with  $\alpha$   $\square$  returns the marking  $M$  defined by  $M'(p) = M(p) - \bar{L}(p,t):\alpha \cup \bar{L}(t,p):\alpha$  for  $p \in P$ .  $M[t/\alpha > M'$  is used to denote the firing of  $t$  with occurrence  $\alpha$  under marking  $M$ . As in traditional Petri nets, two enabled transitions may fire at the same time as long as they are not in conflict;
- (5) For a marking  $M$ , the set  $[M >$  of markings reachable from  $M$  is the smallest set of markings such that  $M \in [M >$  and if  $M' \in [M >$  and  $M'[t/\alpha > M''$  then  $M'' \in [M >$ , for some  $t \in T$  and occurrence mode  $\alpha$  (note: concurrent transition firings do not produce additional new reachable markings);
- (6) An execution sequence  $M_0 T_0 M_1 T_1 \dots$  of  $N$  is either finite when the last marking is terminal (no more enabled transition in the last marking) or infinite, in which each  $T_i$  is an execution step consisting of a set of non-conflict firing transitions;
- (7) The behavior of  $N$  is the set of all execution sequences starting from the initial marking.

## 2.3 The Model Checker SPIN

In general, a complex software system cannot be efficiently verified. However, a finite-state abstract model of a complex software system can be efficiently verified using model checking technology [14, 15, 23]. Model checking involves the technique that exhaustively checks the correctness of a simplified model that preserves the essential characteristics of the system being investigated. SPIN (Simple PROMELA Interpreter) [23] is a generic model checking system that can be used to thoroughly check the high-level model of a concurrent system. It was developed at Bell Labs and is one of the most

widely used logic model checkers in the world [23]. SPIN verification models aim at providing the correctness of process interactions, which can be specified with rendezvous primitives, with asynchronous message passing through buffered channels, and through access to shared variables [23, 38]. That is, SPIN focuses on checking the correctness of the asynchronous control in software systems, rather than synchronous control in hardware systems.

The modeling language used in SPIN is called PROMELA (a Process Meta Language) [23]. A PROMELA model consists of one or more user-defined process templates and at least one process instantiation. The templates define the behavior of different types of processes. Each process is translated by the semantics engine of SPIN into a finite automaton. The behavior of a concurrent system is an asynchronous interleaving product of automata. A correctness claim is specified in a LTL (Linear Temporal Logic) formula [39], which is converted by the semantics engine into a Buchi automation [38]. Given a concurrent system specified in PROMELA, and a correctness claim specified in a LTL formula, the verification process in SPIN either proves that the claim is impossible or provides detailed examples of behaviors that match.

In summary, the model checker SPIN has the following features and functionalities:

- (1) It is focused on the efficient verification of concurrent and asynchronous software systems.
- (2) It accepts correctness claims specified in LTL.
- (3) It includes an XSpin graphical interface at the front-end for constructing PROMELA models, a random/guided simulator for interactive simulation, and a verifier

generator that generates an optimized on-the-fly verification program from a PROMELA model.

The above characteristics make SPIN a suitable verification tool for the study of MAS where intensive agent interactions are involved.

## **2.4 Related Works**

The main idea of designing complex systems in a MAS view is to use the *agent* as the key abstraction for structuring the system [8]. Therefore, MAS have several essential characteristics: (1) multiple agents are engaged; (2) each agent has its own computation logic and controls; (3) agent communications are dynamic and context-based; and (4) the emergent behaviors of agent interactions occur at runtime. As such, the major concern of MAS design is how to handle the complexity due to the heterogeneity and dynamics of multiple agents. There exists a significant amount of research that aimed at supporting MAS design in various aspects [7]. In this section, we review some efforts in designing methodologies, specification languages and coordination mechanisms for MAS design, respectively. More specifically, some of the Petri-net-based works that are most related to this study are also discussed.

### **2.4.1 General Agent-Oriented Design Methodologies**

While the object-oriented paradigm has gained popularity among developers, many agent-oriented design methodologies were extended from the object-oriented paradigm [7]. The works in this category attempt to provide conceptual guidelines and steps for MAS development at the design stage by infusing the object-oriented paradigm with the agent concept. However, these approaches were very different from each other in modeling agent-based systems with respect to the state-of-the-art MAS theories. As yet,

only a few of them have been applied to solve real-world problems. For example, at the Australian Artificial Intelligent Institute (AAIL), an agent-based methodology has been proposed based on the widely accepted BDI (Belief-Desire-Intention) agent architecture [40]. Their model includes an internal model describing an agent's beliefs, desires and intentions, an external model describing the hierarchical relation of agent classes, and an interaction model describing the control structure among agents. The methodology is based on the object-oriented paradigm with an enhancement on agent concepts. It has been applied in several real world applications related to air traffic controls.

Tropos [41] is an agent-oriented software development methodology, which employs actors and goals as fundamental modeling concepts to all stages of a system development process. Tropos methodology supports MAS design from requirement analysis to implementation based on UML notations. The modeling activities proposed include actor modeling, dependency modeling, goal modeling, plan modeling and capability model. Tropos has a rather restricted view in terms of agent architecture, and is weak in semantic models.

Gaia methodology [42] is a general methodology supporting MAS design in agent structure and agent organization. It allows developers to design an agent system from abstract models to increasingly detailed models, i.e., from abstract entities to concrete entities that can be directly implemented. Abstract entities and concrete entities are used in the requirement analysis stage and design stage, respectively. However, the Gaia methodology does not support the concept of agent autonomy.

Other agent-based design methodologies [7], including MESSAGE, AALAADIN and MASE, provide only conceptual models without concrete models.

#### 2.4.2 Coordination Models

Distributed controls and resources are essential characteristics of MAS. Thus, a coordination mechanism ensures overall system consistency. While many agent-oriented modeling methodologies have been proposed [7], few of them dealt with coordination modeling using formal methods. In [43], a constructing method for a skeleton of an agent model was studied. The externally visible events related to the coordination of an agent were first identified; and then, based on the events, the skeleton of an agent was defined using finite state automata. This approach started from building a Dooley graph based on the conversations among agents. The histories of the conversations among agents were then analyzed to induce an agent skeleton. The resulting meta-model represented by a finite state automata was used to validate specified coordination requirements, which were represented by temporal relations. In [44], a coordination problem was defined as a process by which an agent reasons about its local actions and the anticipated actions of other agents in order to act coherently. A distributed goal-search formalism was developed to reason about an agent's commitments and conventions, where commitments are pledges to undertaking some activities, and conventions serve to monitor the commitments. Linda model [45] is a well known coordination mechanism that addresses the coordination within an agent society. In the Linda model, the coordination process is a separated activity from computation activities. The coordination model serves as the glue that binds computational activities in which the communication is through a tuple space. As a result, agents are loosely coupled and communicate through a virtual space and do not need to know any detail about other agents. Law Governed Interaction (LGI) [46] is a message-exchange mechanism based on the Linda model. It allows an open

group of distributed agents to engage in the interactions that are regulated by an explicitly specified policy. The policy is called the interaction law of the group. The controls are decentralized. Each agent has a private controller that realizes interaction laws. The Contract net protocol [47] is a high level communication protocol that was developed to facilitate distributed controls in task sharing [2]. The protocol provides a specification for contracting tasks during a problem solving process of a distributed sensing system.

#### 2.4.3 Aspect-Oriented Modeling

The key idea of the aspect-oriented concept [21] is the separation of cross-cutting concerns in which non-functional properties are modularized into *aspects*. This section introduces two major works in aspect-oriented modeling within the MAS context.

In [48], a concept of model roles was proposed to model aspects in MAS. This is a UML-based modeling technique, which defines subtypes of agency meta-classes describing the MAS architectural diagram elements. A directory facilitator aspectual model was studied and mapped to AspectJ [22] codes.

In [49], a meta-modeling framework was proposed to enrich agent-oriented models with aspects. The framework was composed of three models: the Agent Model, the Aspect Model and the Composition Model. The Agent Model comprised a set of fundamental agent-oriented design elements. The Aspect Model subsumed concepts, relationships and properties for aspect-oriented modeling. The Composition Model provided semantic description of the crosscutting composition mechanism.

#### 2.4.4 Specific Agent Modeling Languages

Specifications are the products of requirement analysis activities in which intangible requirements are transformed into tangible specifications that serve as an important



means for communications among stakeholders. Formal specifications [13] are especially useful to reveal ambiguity and incompleteness of system design at an early stage of a system development process. There has been much research focused on specification techniques for specifying and modeling MAS in informal (in absence of precise semantic definitions) and formal specification languages (with precise mathematical-based notations). Some of the works are discussed in the following sections.

*a. UML-Based Modeling Languages*

Since the Unified Modeling Language (UML) is a widely accepted modeling tool in the main stream software engineering community, many research works extend UML notations with the agent concept for modeling agent-based systems. Agent UML (AUML) [50] exploited the stereotypes, tagged values and constraints in UML to support the modeling of agent-based systems. The sequence diagram and class diagram in UML were extended to model agent interactions and the agent hierarchy. AUML was applied in later works [51, 52] to specify agent interactions in MAS. In [53], AUML diagrams for agent interaction protocols were translated into Petri nets and integrated into a tool called RENEW. Thus, the formal semantics of AUML was defined through Petri nets. Generally speaking, most of the works based on AUML focused on specifying FIPA (Foundation of Intelligent Physical Agents) interaction protocols [54]. Thus, AUML was limited to describing the message passing relations among agents, and has no precise semantic definitions for its modeling elements.

The most recent extension based on UML to support the modeling of agent-based systems is the Agent Modeling Language (AML) [55] and the Multi-Agent System Modeling Language (MAS-ML) [56]. In AML, the authors intended to introduce an agent

modeling language for specifying, modeling and documenting agent-based systems by integrating MAS theories with UML version 2.0. Although AML provides a more comprehensive approach than AUML, it also introduced complexity into the AML specifications.

MAS-ML was developed based on the Taming Agents and Objects (TAO) conceptual frameworks [57]. MAS-ML extended UML class diagram and sequence diagram to describe the structural and dynamic aspects, respectively, based on the TAO meta-model. New meta-classes and stereotypes were created and associated with the extensions of UML meta-model [56]. Although MAS-ML provides a solid conceptual model from the TAO meta-model, it does not cover all elements defined in the TAO meta-model. Thus, the notations of MAS-ML elements are difficult to construct and apply for MAS specifications.

In summary, these extended UML-based modeling languages did not provide precise semantics, thus did not support formal analysis.

#### *b. Temporal Logic and the Z Notation*

Temporal logic is a property-oriented modeling language [13] and is very popular in specifying the system properties of distributed systems. A property-oriented specification language defines system behaviors indirectly by a set of properties. Temporal logic is well suited to specify reactive systems and has been adopted in [16, 58] to specify agents. Although it is very successful in specifying the properties of reactive agents based on the BDI (Belief-Desire-Intention) agent architecture [59, 60], its property-oriented nature provides no state transition relations. Thus, not only it is difficult to map a temporal logic specification to an implementation, but it is also difficult to specify agent interactions.

Z notation (Z) is a formal specification language for specifying the functionality of sequential systems based on typed set theory and first order logic. It offers a rich type definition facility and supports formal reasoning [61]. In [62, 63], Z was used to define MAS in a four-tier hierarchy, namely entity and environment, objects, agents and autonomous agents. An agent specification is defined by agent goals, agent perceptions, agent actions, agent states and agent operations. Although Z provides rich type definitions, it does not provide explicit operational semantics and effective definitions of the concurrency [61]. Thus, it is insufficient to specify the concurrent and interacting behaviors in agent-based systems. Furthermore, there was no discussion about agent architecture and specifying rational behaviors in [62, 63].

#### 2.4.5 Petri Nets

Since Petri nets are an excellent concurrent model for studying distributed systems, they have been used in many research works to specify agent-based system. However, most of the works focused on specifying the global control structure and mobility of agents, while agent autonomy and its social behaviors were not covered. In summary, each of the previous research works based on Petri nets explored different aspects in modeling agent-based systems; such as, agent mobility and construction of agent plans. There is still a lack of systematic approaches for modeling and verifying MAS. These works are discussed based on their approaches and contributions.

##### *a. Nets-within-Nets*

Nets-within-nets paradigm addresses the dynamic structure of an agent system. Therefore, several works [25, 27, 28, 64] adopted the concept of nets as token objects [19] to model agent systems with a layered structure. In [25, 64], colored Petri nets were

extended by incorporating the object-oriented concept with the nets-within-nets paradigm to address the modeling of agent mobility. Synchronous communications between nets at different layers were modeled through a fusion of two enabled transitions, where the information exchange was bi-directional. In [27], a layered PrT net structure was used to specify agent systems with mobile agents. The information flows between an agent net and the system net at different layers were through internal connectors, which have to be constantly updated according to the changing number of agents to maintain consistency within the system net. In [28], a layered PrT net structure was also adopted. However, the information flows between layers were not explicitly defined. Instead, the focus was on modeling the behavior of a mobile agent system, which was constructed based on the MASIF (Mobile Agent System Interoperability Facilities) defined by the OMG (Object Management Group). These works [25, 27, 28, 64] addressed the mobility and structural adaptability of agent models by representing agents as net tokens in the system net. As a result, the movement of net tokens models the change of locations (mobility) of mobile agents.

*b. Modeling Agent Communication*

In [29], an agent-based G-net model derived from the object-oriented G-net was applied at the design level of a software development process. An agent-oriented G-net model, which incorporated the BDI agent architecture, was proposed to address the message passing among agents. The focus was on dealing with the incoming and outgoing messages processing in modeling agent communications without agent cooperation and coordination. In [27, 28], agent communications were specified in transitions; however, there was no formal semantics defined. In [25], synchronous

communications between nets at different layers were modeled through a fusion of two enabled transitions, where the information exchange was bi-directional.

*c. Modeling Coordination*

Resource sharing is the most common interdependency among agents in MAS. Thus, conflicts may occur when limited resources are demanded by multiple agents. In [30, 65], a component-based modeling approach was developed based on Colored Petri nets by the invention of potential arcs that are introduced to resolve conflicts for resources among agents. Potential arcs modeled the resources incoming from or outgoing to the external environment, and were added as additional constraints for enabling transitions. The potential arcs were later transformed into coordinators, which include all possible alternate paths coordinating shared resources. The focus of this approach was on constructing a conflict-free global plan for MAS at design level. In [28], the interoperability was achieved through the modeling of a finder, which served as the yellow page in a mobile agent system. However, there was no detail about modeling agent cooperation. Another work on agent coordination modeling was the moderator coordination model proposed in [66]. This work focused on agent interaction protocols and the ontology of agent conversations. The moderator was separated from agent models for handling the conversations among agents in an organizational view. The protocols were isolated from agent models and considered as the resources and predefined processes that agents have to follow. A moderator encapsulating a well-identified process was generated for each conversation between agents; and, a conversation server was defined to keep the information of all active conversations. The moderators were used as the coordination model to grant roles to agents and control their ongoing conversations.

The behavior model was specified using a formalism based on Petri nets, which were extended with object-oriented features called CoOperative Objects.

*d. Modeling Methodology within the MAS context*

The works in [29, 30], which are discussed in the previous section, provided the modeling methodology based on high-level Petri nets within the MAS context. However, a recent work [67] used low-level net as the modeling tool for modeling and analysis of multi-agent systems. In this work, low-level Petri nets were used to model an intelligent agent abstract architecture. The authors focused on developing an analytical methodology to analyze the structural properties of a multi-agent system. In [68], PrT nets were used to model a multi-agent blocks problems. The authors demonstrated the viability of verifying PrT net models based on planning graph reachability analysis, and the procedures for verifying hierarchical multi-agent plans with explicit parallel actions. In [26], a modeling methodology based on object-oriented Colored Petri nets was introduced. The authors introduced the object-oriented concept into Colored Petri nets to model agents with mental states, and used a logic programming language for the declaration of agent programs.

## **CHAPTER 3**

### **A NESTED PETRI-NET FRAMEWORK**

Nets-within-nets, originated from Valk's work [19, 20], are high-level Petri nets that allow nets to be tokens within a net. The paradigm provides a natural abstraction of a hierarchical MAS organization. In recent years, several approaches based on the nets-within-nets paradigm were proposed for modeling the communication and mobility of a mobile agent system [25, 28]. However, previous works in this regard were primarily focused on modeling agent mobility rather on supporting an overall MAS modeling. Building on the object-oriented paradigm, these works with the nets-within-nets paradigm focused on the dynamic structure of Petri nets, but did not employ an agent-oriented modeling approach [8] and did not demonstrate how an agent net can be built with essential properties, such as autonomy, pro-activeness, reactivity and sociality [5]. More specifically, the work in [28] was limited to mobile agent systems and was not in the MAS context.

In [25], the nets-within-nets paradigm was used to model a mobile agent system based on a Colored Petri net extension called Reference nets [24], which are object-oriented high-level Petri nets. This work was based on a multi-agent architecture MULAN [24], which is defined to describe the hierarchical structure of an agent system. The communication between an agent net and the system net was synchronized through the concept of a channel defined by a fusion of two enabled transitions. The firing of these transitions allows bi-directional information exchange between the agent and the system, as well as the movement of the agent. An active token in the system net refers to

an object net, which can be triggered by using synchronous channels [69, 70]. Although a hierarchical structure MULAN is provided to model agent mobility, there is no detail of how to model an individual agent and agent coordination.

In [28], the nets-within-nets paradigm is used for modeling agent mobility based on the MASIF defined by OMG. Agent mobility is addressed through the synchronization of some input and output transitions; however, there was no formal definition provided. The work in [27] also used a layered PrT net to specify mobile agents. However, the information flows between an agent net and the system net at different layers were through internal connectors. The internal connector had to be constantly updated according to the changing number of agents to maintain the consistency within the system net.

This dissertation research is based on the nets-within-nets paradigm as well, however targets at modeling MAS. PrT nets are used as the underlying formalism. Since original PrT nets have a static net structure and provide only one level view of an overall system process, they are inadequate to model a hierarchical multi-agent organization. Moreover, PrT nets do not have the notion of agent communication. The basic idea to support MAS modeling is modularity [4]. That is, each agent having its own computation logic (behaviors) has to be independently modeled and loosely coupled in a hierarchical structure. My approach to address the representation of MAS includes two facets: (1) adapting the nets-within-nets paradigm to address the MAS architecture, and (2) extending PrT nets to address agent communication. In this study, the constraint formula of a transition defined in a PrT net is extended to include a channel expression for agent communication. A new definition of the channel command and PrT nets with this channel



concept are given. The new channel concept adapts the input and output commands in Hoare's process algebra CSP (Communicating Sequential Processes) [18]. The communication between agents consists of synchronous control and unidirectional information flow, with which the essential characteristics of reactivity (input) and proactiveness (out) of agents can be nicely modeled. In addition, the channel expression is used to dynamically couple agents.

Building on the PrT net extension, a framework has been developed to provide a systematic approach for modeling and analyzing MAS. The framework is based on a conceptual model that identifies essential MAS components based on the widely-accepted definition of MAS [1, 4, 71]. Unlike the other works, the framework provides a full line of support from modeling to analysis, including a process model for modeling MAS, a methodology that systematically transforms the conceptual model to a nested PrT net describing a multi-agent system, and a technique that systematically translates a nested PrT net to a PROMELA program in the model checker SPIN.

The rest of this chapter is organized as follows. Section 3.2 gives an overview of the nested Petri net framework. Section 3.3 presents the formal definition of two-level nested PrT nets. Section 3.4 introduces net patterns and semantics of two-level nested PrT nets.

### **3.1 An Overview of the Framework**

One of the reasons to build an abstract model at the system design stage is to visualize system behaviors at an early stage of system development. The objective is to detect design errors and avoid costly fixes at a later stage of system development. Since MAS involve dynamic agent interactions, their behaviors are difficult to be directly observed through textual specifications. Thus, this research work takes advantage of the modeling

power and visual representation of PrT nets, and of SPIN tool in its well developed verifier, to formulate a methodology for the study of multi-agent systems prior to implementation. The components of the framework are shown in Figure 3.

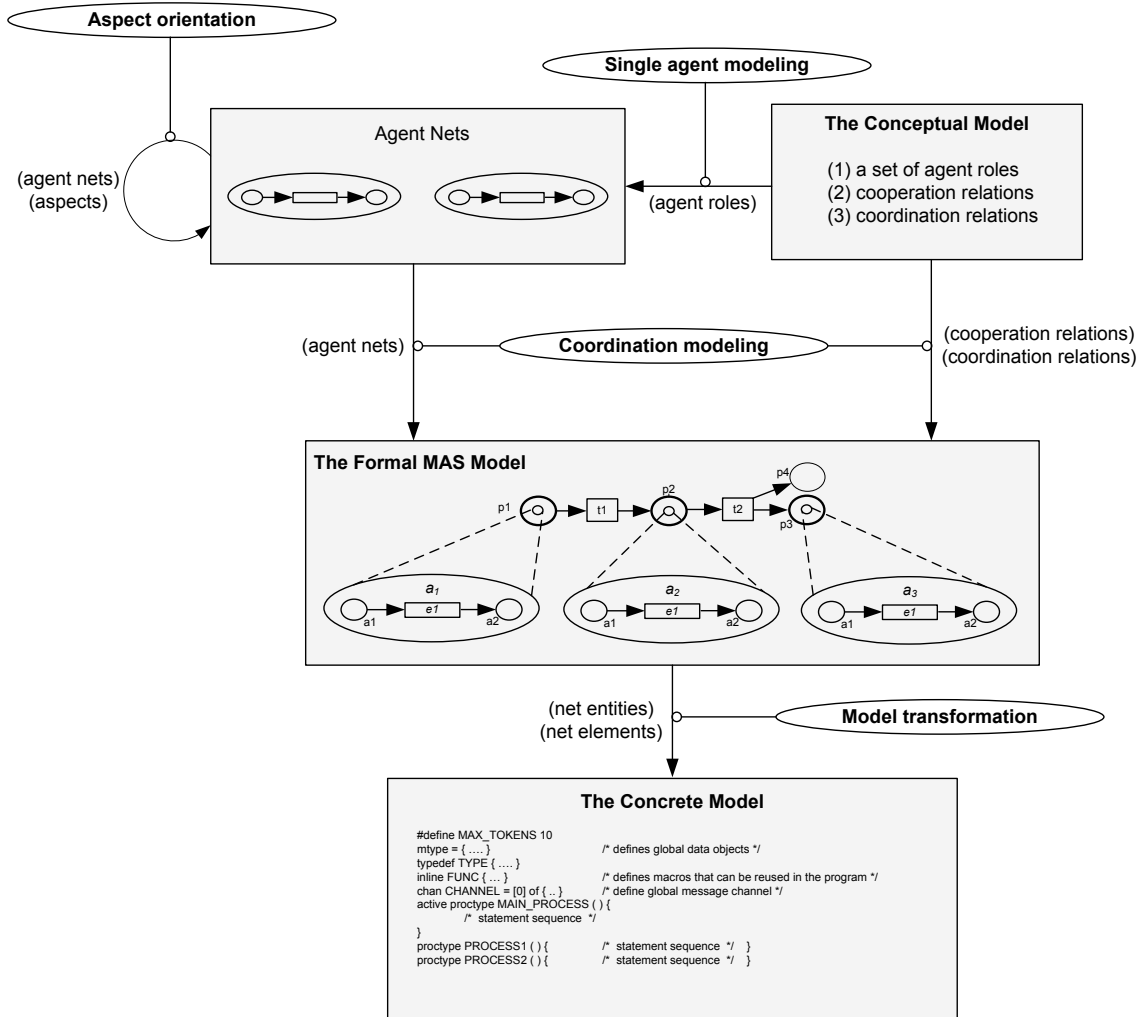


Figure 3. Components of the nested Petri net framework.

### 3.1.1 The Conceptual Model

MAS are a distributed problem solving approach [2, 4] for large and complex problems. Thus, MAS design generally involves several essential components: (1)

multiple distributed agents, which are designed to accomplish specific tasks; (2) communication, which allow agents to exchange information; (3) cooperation, which defines the process to accomplish a given task that relies on multiple agents; and (4) coordination, which defines the mechanism for managing available resources. A conceptual model is shown in Figure 4.

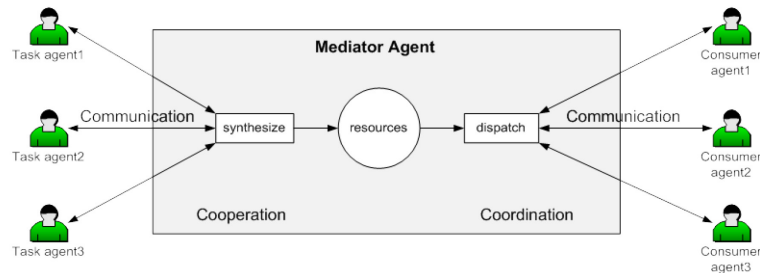


Figure 4. MAS components: a conceptual model.

Here, a mediator agent is considered as a special type of agent that is instrumented with some coordination mechanism for handling cooperation processes among agents within an agent community. Since agents evolve constantly in a distributed and heterogeneous environment, separation of the coordination logic eases the complexity of agent communications and allows the flexibility of coordination mechanisms for future extensions [36]. The framework developed in this study is based on the conceptual model shown in Figure 4.

### 3.1.2 The Formal MAS Model

An important concept of applying the MAS paradigm is modularity. Traditional PrT nets have a flat structure that models only a single-level view of distributed systems. Thus, in this study, PrT nets are enhanced in modularity by incorporating agent-oriented concepts and aspect-oriented concepts. Agent-oriented concepts address the modularity of

system architecture, while aspect-oriented concepts address the modularity of an agent's internal structure. The modularity at the system architecture level is achieved by employing a two-level nested PrT net structure, and the modularity at an individual agent level is achieved by employing an aspect orientation technique. Agents are essentially heterogeneous in the MAS context, and are specific problem solvers. The aspect-oriented concept [21] is adopted to address different features of an agent. For example, it is well recognized that the essential properties of an agent are autonomous, proactive, reactive and social in terms of AOSE. However, agents can be further classified by their functionalities [35]; such as, task agents, interface agents, mobile agents, information agents, reactive agents, hybrid agents and smart agents. Aspect orientation of agent features provides structural modularity of an agent model, and promotes the reusability of common behavior models. Since the process model provides system-wide modularity, it also achieves several non-functional performance measurements, such as adaptability (a dynamic structure), reusability (aspect orientation), and extensibility.

In this study, an agent-oriented modeling mechanism is employed to transform the conceptual model (Figure 4) to the formal MAS model in a two-level nested net structure. The mechanism includes a single agent modeling technique for constructing individual agent nets, and an aspect-oriented modeling technique for modularizing the internal structure of an agent net (Figure 3). As a consequence, the MAS model can be constructed and contains multiple nets in a two-level nested net structure that provides a global view (described by the mediator agent) and a local view (described by agent nets).

### 3.1.3 The Concrete Model

Although a PrT net can be unfolded to a low-level Petri net and analyzed using existing Petri-net-based analysis techniques, such as reachability tree [37]. However, the unfolding process is tedious and error prone. Given that the reachability tree technique is difficult to apply to PrT nets, a viable method for model analysis is to directly execute the model. Previous works in this regard generally fall into two major categories: (i) transforming a high-level net model to an executable program written in high level programming languages, such as C or JAVA [72, 73, 74, 75]; and (ii) transforming a high-level net model to an executable model described by the meta-language supported in some simulation tools [76, 77, 78, 79]. This study is similar to the works in the second category. However, the focus here is model checking nested PrT nets. That is, the problem is not just to facilitate the analysis of high-level nets, but also two-level nested PrT nets.

Given that there is no well-established method for the analysis of two-level nested PrT nets, the problem here is to establish an efficient method for analyzing two-level nested PrT nets. Drawing upon existing tools for the analysis of abstract models, the model checker SPIN is a well developed tool for model checking concurrent systems. More important, it focuses on the modeling of asynchronous process interactions, which is an important feature for MAS analysis. This study takes advantage of the well-developed verifier in SPIN for the analysis of nested PrT nets. The idea is to transform nested PrT nets into PROMELA programs for model analysis. Thus, a model transformation technique is developed to transform formal MAS models to PROMELA programs in SPIN. The transformation technique includes a set of translation rules that

systematically translate net entities and net elements of nested PrT nets to their associated PROMELA statements. This results in a concrete model (written in PROMELA) that can be verified through the model checker SPIN.

### **3.2 The Underlying Formalism**

The underlying formalism of this dissertation research is predicate transition nets, which are formally defined in Section 2.3. This section formally defines two-level nested PrT nets.

#### **3.2.1 Two-Level Nested Predicate Transition Nets**

Although a general structure of deeply nested nets can be defined, here I elect to just define a two-level net structure that is adequate for this study. Since lower level nets serve as tokens in the upper level net, some care must be taken to ensure the upper level net is well defined. First, the upper level net has its own data abstraction and processing capabilities; thus, it needs the full description power of PrT nets. Second, individual computation entities have their own behaviors and logics that cannot be described by static data. As a result, net tokens are treated as black boxes. Only their identities are visible and accessible in the upper level net. This treatment allows these net tokens to be grounded and thus well-defined. This treatment also respects the autonomous characteristic of agent tokens. Third, the creation and removal of a net token can be done through some boundary transitions without input places and transitions without output places, respectively. The functionality (constraints) of these transitions is left open and viewed as the responsibilities of an external environment.

To address the interactions between the upper level net and agent nets, the constraint definition of PrT nets is extended to include channel expressions. The channel commands

in CSP (Communicating Sequential Processes) [18] are adopted for the channel expressions in transition constraints. A channel expression is either an output command  $n!e$  or input command  $n?x$ , where  $n$  is a channel name,  $e$  is an expression, and  $x$  is a variable. A channel name is the identification of a net and is used to control the synchronization of a pair of transitions. A synchronized communication occurs when two enabled transitions at different level have a matching pair of input and output expressions. After synchronization, a unidirectional information flow occurs such that the value in  $e$  of the output command is assigned to the variable  $x$  of the input command.

For example, as shown in Figure 5, place  $A1$  at the upper level net contains four agent net tokens, including  $a1$ ,  $a2$ ,  $a3$  and  $a4$ ; and place  $A2$  contains three agent net tokens, including  $a5$ ,  $a6$  and  $a7$ . A transition  $t2$  in the upper level net with identification  $S$  contains  $a?x$ , and a transition  $e1$  in an agent net with identification  $a1$  contains  $S!e$  are a matching pair of input and output expressions. The firing of both transitions is an *interaction* denoted as  $(t2, e1)$ . Transition  $t2$  is called *input channel* that inputs a data token from a lower level net, and transition  $e1$  is called *output channel* that outputs a data token to the upper level net.

To enforce vertical communications (that is, no direct communications between agent nets), the channel names in agent nets must be the identification of the upper level net. In this case (in Figure 5), the channel name in each agent net should be  $S$ . However, the channel name in the upper level net can be a variable ranging over agent identifications. It is instantiated when an agent token is instantiated. For example, variable ' $a$ ' in transition  $t1$  and  $t2$  of the upper level net shown in Figure 5.

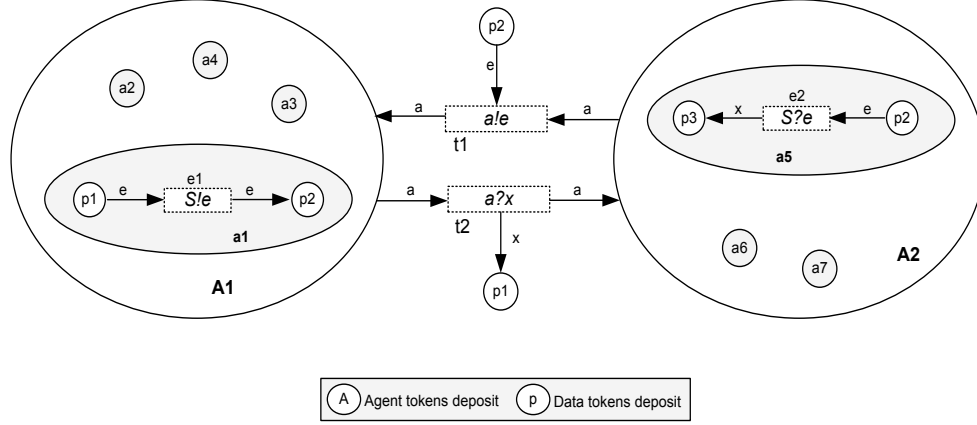


Figure 5. An upper level view of a nested PrT net with input/output channels and net tokens.

*Definition 3.2.1* A channel expression is either  $n!e$  or  $n?x$ , where

- $n$  is a net identification,
- “?” denotes an incoming direction,
- “!” denotes an outgoing direction,
- $n!e$  denotes an output of value  $e$  to net  $n$ , and
- $n?x$  denotes an input of value incoming from net  $n$  to  $x$ .

*Definition 3.2.2* A communication channel is a transition  $t$  with the constraint formula

$R(t) = R_u(t) \wedge R_c(t)$ , where

- $R_u(t)$  is a non-communication constraint, and
- $R_c(t) = n!e \mid n?x$  is a non-empty communication constraint.

*Definition 3.2.3* A two-level nested net is a tuple  $(S, AG, I)$ , where

- $S$  is a PrT net at the upper level called *mediator net*;
- $AG$  is a finite set of PrT nets called *agent net*;



- $I \subseteq T_S \times T_{AG}$  is the set of interaction relations, where
  - $I \neq \emptyset$ ,
  - $T_S$  is the set of transitions in  $S$ ,
  - $T_{AG} = \bigcup T_i$  such that  $i = 1$  to  $|AG|$ , and  $T_i$  is the set of transitions in agent net  $AG_i$ , and
  - $(t, e) \in I$  is an *interaction* relation of a nested net, where  $t \in T_S$  and  $e \in T_{AG}$ ;  $t$  and  $e$  are communication channels.

The dynamic semantics of two-level nested PrT nets can be defined as follows:

- (1) A marking of a PrT net is a mapping from  $P$  to sorts defined in *Spec*;
- (2) An occurrence mode of  $N$  is a substitution  $\alpha = \{x_1 \leftarrow c_1, \dots, x_n \leftarrow c_n\}$ , which instantiates typed label variables.  $e: \alpha$  is used to denote the result of instantiating an expression  $e$  with  $\alpha$ , in which  $e$  can be either a label expression or a constraint;
- (3) Given a marking  $M$ , a transition  $t \in T$ , and an occurrence mode  $\alpha$ ,  $t$  is  $\alpha$ \_enabled at  $M$  iff the following predicate is true:  $\forall p: p \in P. (\bar{L}(p, t): \alpha \subseteq M(p)) \wedge R(t): \alpha$ , where  $R(t) = R_u(t) \wedge R_c(t)$ .  $R_u(t)$  is a non-communication constraint, and  $R_c(t) = n!e \mid n?x$  is a communication constraint.
- (4) An  $\alpha$ \_enabled transition  $t$  with communication constraint  $R_c(t)$  is ready if a transition with a matching channel expression is also ready. A ready transition  $t$  under marking  $M$  with occurrence  $\alpha$  is fireable.
- (5) The firing of ready transition  $t$  with channel expression  $R_c(t)$  under  $M$  with  $\alpha$  returns the marking  $M'$  defined by  $M'(p) = M(p) - \bar{L}(p, t): \alpha \cup \bar{L}(t, p): \alpha$  for  $p \in P$ .  $M[t/\alpha > M'$  denotes the firing of  $t$  with occurrence  $\alpha$  under marking  $M$ .

- (6) For a marking  $M$ , the set  $[M>$  of markings reachable from  $M$  is the smallest set of markings such that  $M \in [M>$  and if  $M' \in [M>$  and  $M'[t/\alpha>M''$  then  $M'' \in [M>$ , for some  $t \in T$  and occurrence mode  $\alpha$  (note: concurrent transition firings do not produce additional new reachable markings);
- (7) An execution sequence  $M_0 T_0 M_1 T_1 \dots$  of  $N$  is either finite when the last marking is terminal (no more enabled transition in the last marking) or infinite, in which each  $T_i$  is an execution step consisting of a set of non-conflict firing transitions;
- (8) The behavior of  $N$  is the set of all execution sequences starting from the initial marking.

Note: the effect of information flows after an interaction has been reflected in the arc label expressions. For example, the value of variable  $x$  in an input channel command can affect the new marking  $M'$ .

For simplicity, the channel expression is defined in a one-to-one and unidirectional fashion. That is, each firing of a transition  $t$  sends/receives one message token to/from a single agent net. However, the communication channel can repeatedly fire as long as there are enough tokens in the input place.

### 3.3 Net Representations and Semantics

This section demonstrates typical patterns of net structures and their semantics in two-level nested PrT nets. These patterns and semantics are based on the following assumptions.

- (1) An agent net models the behavior of an autonomous agent, which has a computation logic that is independent from other agents;

- (2) All agent nets communicate through the mediator net; that is, there is no direct communication between agent nets;
- (3) All agent nets should be modeled prior to the mediator net; that is, it is assumed that all agent roles and their responsibilities have been identified before a multi-agent system can be built;
- (4) The mediator agent net is a coordination model accommodating agent nets;
- (5) Pre-existing agent nets can be used as token templates in the mediator net and repeatedly instantiated during model execution; however, instantiated agent nets from the same template are independent from each other; that is, they are different instances with different agent identification.
- (6) An agent net cannot conserve agent tokens, while a mediator agent net must conserve pre-defined agent net tokens.

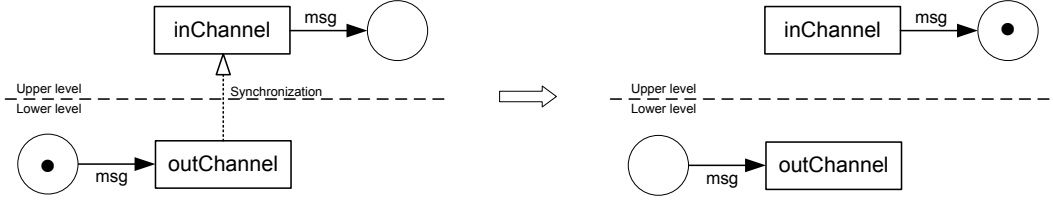
### 3.3.1 Agent Nets

The agent nets at the lower level cannot conserve agent tokens; that is, only data tokens are allowed in agent nets. Therefore, syntactically, there is no difference between the net representation of an ordinary PrT net and the net representation of an agent net in nested PrT nets. Semantically, however, there is a non-empty set of transitions model communication channels in an agent net.

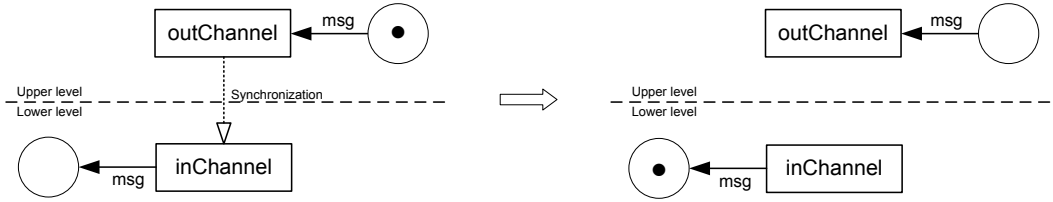
*Definition 3.3.1* An agent net is a PrT net with net structure  $N = (P, T, F)$ , where:

- $T = T_u \cup T_c$ ;
- $T_u$  is the set of non-communication transitions such that,  $\forall t \in T_u. (R_c(t) = \lambda)$ , where  $\lambda$  denotes an empty channel expression;

- $T_c$  is the set of communication channels such that,  $T_c \neq \emptyset$  and  $\forall t \in T_c. (R_c(t) \neq \lambda)$ ;
- $\forall p \in P. (\varphi(p) = DATA\_TYPE)$ , where  $DATA\_TYPES$  is the set of token types that define data tokens.



(a) pro-activeness



(b) reactivity

Figure 6. (a) Pro-activeness of an agent net; (b) reactivity of an agent net.

For example, Figure 6(a) shows a communication between nets at different level in which an input communication channel in the upper level net (the mediator net) generates a message token, while an output communication channel in the lower level net (the agent net) consumes a message token (Figure 6(a)). The channel expression of transition *outChannel* in the agent net can be defined as  $R_c(outChannel) = S!msg$ , where  $S$  is the *id* of the mediator net. On the other hand, some input channel of the agent net can be defined as  $S?msg$ , which may generates a data token that is sent from mediator net  $S$  (Figure 6(b)). The net representation for a communication channel is no difference from a

regular transition. However, since the constraint formula is augmented with a channel expression, the semantics of a communication channel is different from a regular transition.

During an execution, an agent net is *proactive* if it initiates an event (Figure 6(a)); and, it is *reactive* if it performs some action in response to an external event (Figure 6(b)).

### 3.3.2 The Mediator Agent Net

The mediator agent net at the upper level models the global process in a multi-agent system. A mediator agent net is a special type of agent net that conserves agent net tokens in addition to data tokens. Syntactically, the net representation of a mediator agent net is the same as an ordinary PrT net. Semantically, there is a non-empty set of transitions that model communication channels in a mediator agent net. In addition, each communication channel in the mediator agent net associated with a pair of input and output places that hold agent net tokens. A *mediator agent net* models the cooperation among agent nets. It is formally defined as follows.

*Definition 3.3.2* A *mediator agent net* is a PrT net with net structure  $S = (P, T, F)$ , where

- $T = T_u \cup T_c$  is the set of possible activities in a cooperation process;
- $T_u$  is the set of non-communication transitions such that  $\forall t \in T_u. (R_c(t) = \lambda)$ , where  $\lambda$  denotes an empty channel expression;
- $T_c$  is the set of communication transitions such that  $T_c \neq \emptyset$  and  $\forall t \in T_c. (R_c(t) \neq \lambda)$ ;
- $P = P_d \cup P_a$  is the set of places, where  $P_d$  denotes the set of places that hold data tokens, and  $P_a$  denotes the set of places that hold agent tokens;  $P_d \cap P_a = \emptyset$  and  $P_a \neq \emptyset$ ;

- $\forall p \in P_a. (\varphi(p) = IDENTIFICATION \times AGENT\_NET)$ , where
  - *IDENTIFICATION* is a unique number or name for identifying an agent net;
  - $AGENT\_NET \in AGENT\_TYPES$  where *AGENT\_TYPES* is the set of pre-defined agent nets that are valid in the MAS model;

Based on the above definitions, there are some essential patterns and limitations for the net representation of a mediator agent net. The patterns and limitations are introduced as follows.

### A. Net Patterns

#### (1) *Output information to an agent net*

In the mediator agent net, a transition with an output channel command is enabled when there are message tokens and agent tokens in its input places. Figure 7 shows a net structure like such. An output information flow occurs when the transition fires. The firing of an output communication channel sends the message token to the agent net that has the net identification indicated in the channel expression (channel name). That is, the firing of an output communication channel consumes a message token.

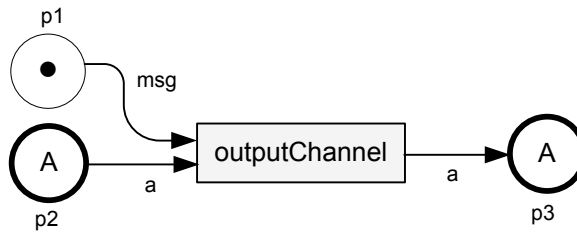


Figure 7. Net representation of an output communication channel.

For example, place  $p2$  in Figure 7 holds agent net tokens, while place  $p1$  holds message tokens. Let us assume the token type definition of  $p2$  is  $\varphi(p2) = INTEGER \times$

$AGENT\_NET$ , where the first element of the token is an agent id number and the second element is an agent net with the type  $AGENT\_NET$  defined in  $Spec$ . The output channel expression can be defined as  $R_c(outputChannel) = a[1]!msg$ , which means “send information token  $msg$  to the agent net with the  $id$  number contained in the first element of variable “ $a$ ”. After firing transition  $outputChannel$ , the agent net that received the information token is sent to output place  $p3$ .

(2) *Input information from an agent net*

A transition with an input channel command is enabled when there are agent tokens in the input place. Figure 8 shows a net structure like such. An input information flow occurs when the transition fires. The firing of an input communication channel received a message token from the agent net that has the net identification indicated in the channel expression (channel name). That is, the firing of an input communication channel generates a message token in  $p3$ .

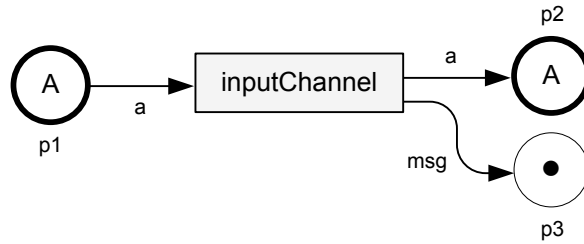


Figure 8. Net representation of an input communication channel.

For example, place  $p1$  and  $p2$  in Figure 8 holds agent net tokens, while place  $p3$  holds message tokens. Let us assume the token type definition for  $p1$  and  $p2$  is  $\varphi(p1) = \varphi(p2) = INTEGER \times AGENT\_NET$ , where the first element of the token is an agent id number, and the second element is an agent net with the type  $AGENT\_NET$  defined in

*Spec.* The input channel expression can be defined as  $R_c(inputChannel) = a[1]?msg$ , which means ‘input information token  $msg$  from the agent net with the  $id$  contained in the first element of variable  $a$ ’. After firing transition  $inputChannel$ , a message token is generated and put into place  $p3$ . The agent net that sent the information token is moved to place  $p2$ .

### (3) *Agent Autonomy*

In a global process view, an agent net is an independent computation entity that has its own thread of control over its own internal states. Therefore, an agent net is executing autonomously without a centralized control, and its behavior is invisible at the upper level net. The way that a transition in an agent net fires without the intervention of the mediator agent net is called *agent autonomy*.

### (4) *Transportation*

A transition firing moves an agent net token from an input place to an output place is called *agent transportation*. Agent transportation models the mobility of agent nets. For example, in Figure 9, an agent is transported from location  $p1$  to location  $p2$ . Agent transportation does not involve communication. That is, transition  $t$  is not a communication channel.

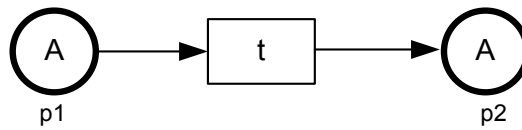


Figure 9. Transportation.



### (5) *Choice*

A global process is composed of a set of activities (or tasks) in which multiple agents are involved. An agent is designed to accomplish a certain task. On the other hand, an agent may have more than one capability to participate in different activities. For example, a gas producer may produce regular gas or premium gas. The option for an agent to participate in different activities constitutes a *choice*. For example, in Figure 10(a), an output message is sent to an agent net through either output channel  $t1$  or  $t2$ ; that is, an agent net is instantiated to participate in either activity  $t1$  or  $t2$ . In Figure 10(b), an input message is received from an agent net through either input channel  $t3$  or  $t4$ ; that is, an agent net can trigger either activity  $t3$  or  $t4$ .

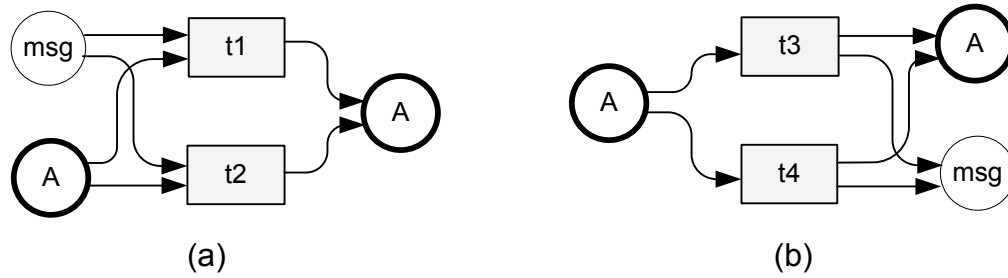


Figure 10. Choice; (a) output channels; (b) input channels.

### (6) *Synchronization*

A transition firing that synchronizes agent nets is called *synchronization*. For example, in Figure 11, two agent nets (A and B) are synchronized when transition  $t$  fires. Synchronization simply synchronizes agent nets, and does not involve agent communications.

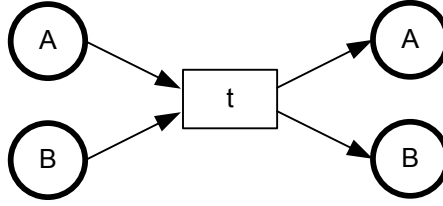


Figure 11. Synchronization.

(7) *Instantiation*

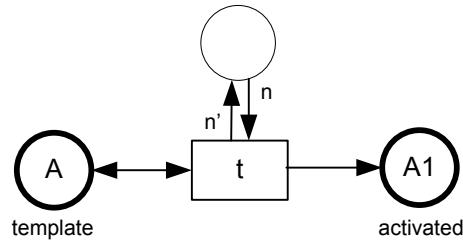


Figure 12. Instantiation.

Agent nets are pre-existing entities in the upper level net. An agent net can be a template and instantiated during model execution. A transition firing that instantiates an agent net with a unique identification is called *instantiation*. For example, in Figure 12, agent net *A1* is instantiated with an id number *n* after transition *t* fired. A template agent net is not an active token, while an activated agent net is an active token. An instantiation does not involve agent communication.

(8) *Cooperation*

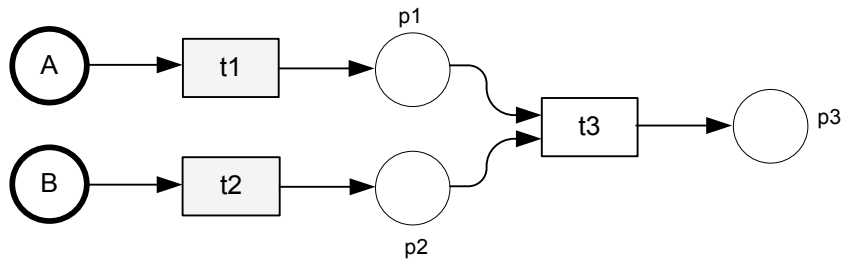


Figure 13. Cooperation.

Agents cooperate in a global process. For example, in Figure 13, agent net  $A$  and agent net  $B$  are sending back results through communication channel  $t1$  and  $t2$ , respectively. The results are stored in place  $p1$  and  $p2$ , and are combined through the firing of transition  $t3$ . A final result is put in  $p3$ .

#### (9) Coordination

Agents share resources in a global process. A mediator agent net coordinates for resource sharing. Figure 14 shows an abstract net for coordinating resources. For example, in Figure 14, the agent net in place  $p1$  can send a request for some resource. An agent net that has sent a request is moved to place  $p2$  and waits for the requesting resource. Transition ‘retrieve’ and ‘Resource\_out’ model the constraint for dispatching the resource in place  $p4$ .

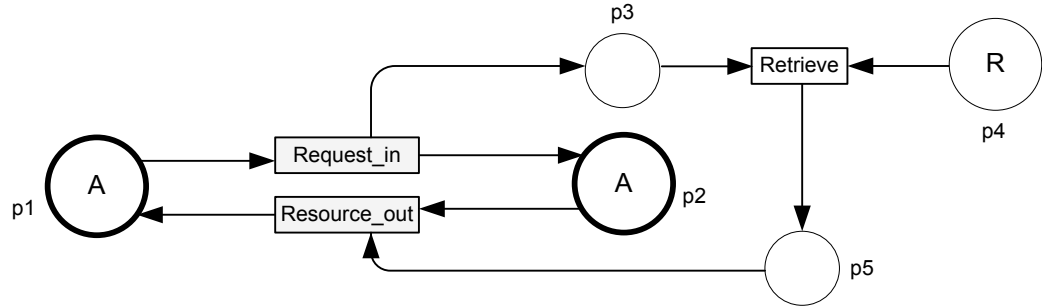


Figure 14. A control of resources.

### B. Restriction

An activated agent net cannot be replicated since an activated agent net is an independent process with a unique process id (agent identification). For example, in Figure 15, the firing of transition  $t$  replicates agent net  $A$ . The replication results in two agent nets with a duplicate id, which will lead to incorrect system behavior.

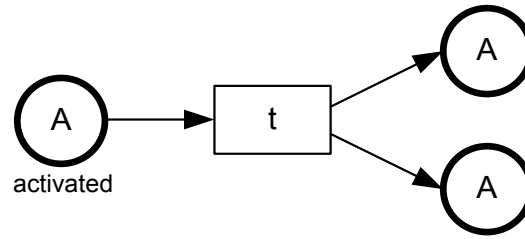


Figure 15. Replication.

## CHAPTER 4

### A METHODOLOGY FOR MODELING MULTI-AGENT SYSTEMS

The formalism introduced in chapter 3 provides a theoretical foundation for the representation of MAS. There are three different models involved at different stages of a MAS modeling process: (1) the *conceptual model*, which is a product of system requirement elicitation; (2) the *formal MAS model*, which is the nested PrT net model based on the conceptual model; and, (3) the *concrete model*, which is the transformed PROMELA model for verification. A modeling methodology is considered as a collection of methods that enable designers to transform instances of the model at one stage into the model at the next stage. This chapter introduces a modeling methodology developed to support the construction of MAS models based on two-level nested PrT nets. An agent-oriented modeling technique is employed and infused with aspect-oriented concepts. The idea is to modularize the common features of agents into aspects, which can be flexibly woven into agent nets based on requirements. Since this study focuses on formal specification, how to elicit domain specific agent roles and their associated responsibilities at the requirement analysis stage is outside the scope of this study. It is assumed that, at the point of constructing abstract models for the system, the agent roles and their associated functionalities have been explicitly identified.

The rest of this chapter is organized as follows. Section 4.1 introduces the concept of agent-oriented modeling. Section 4.2 formally defines an aspect orientation technique for the modularization of an agent net. Section 4.3 demonstrates a method for constructing a single agent net. Section 4.4 defines the coordinator for coupling an agent net and the

mediator agent net, and demonstrates a method for constructing a mediator net by composing relevant coordinators. Section 4.5 concludes the chapter by comparing this study with other related works.

#### **4.1 Agent-Oriented Modeling**

The traditional approach for building an intelligent agent program in the Artificial Intelligence (AI) community suggests that an agent program is a function that maps agent percepts to actions while updating its internal states [32]. There are *reflex* agents, *goal-based* agents and *utility-based* agents where each type of agent programs exhibits various degree of intelligence. Thus, the process of making decisions by searching for solutions based on agent knowledge is central to the design of agent programs. However, in this study, the objective is to formulate a recipe for the abstraction of complex systems based on agent-oriented concepts prior to implementation. Therefore, the detail of how to implement the search algorithm that solves a problem efficiently using an agent is not the focus here.

The agent-oriented concept adopted in this study is based on the widely recognized definition in the Agent-oriented Software Engineering (AOSE) [8, 31, 71] community. An *agent* is defined as follows.

*Definition 4.1.1* An *agent* is a distributed computation entity that is situated in some environment and capable of flexible, autonomous actions in that environment in order to meet its design objective.

The above definition highlights several important features with regard to an agent-oriented design: (i) an agent is a distributed computation entity designed to solve a

specific problem; (ii) an agent is embedded in an environment and can proactively get information and react to affect that environment; and, (iii) an agent is autonomous and self-contained in controlling its own states; it performs the action of its best interest.

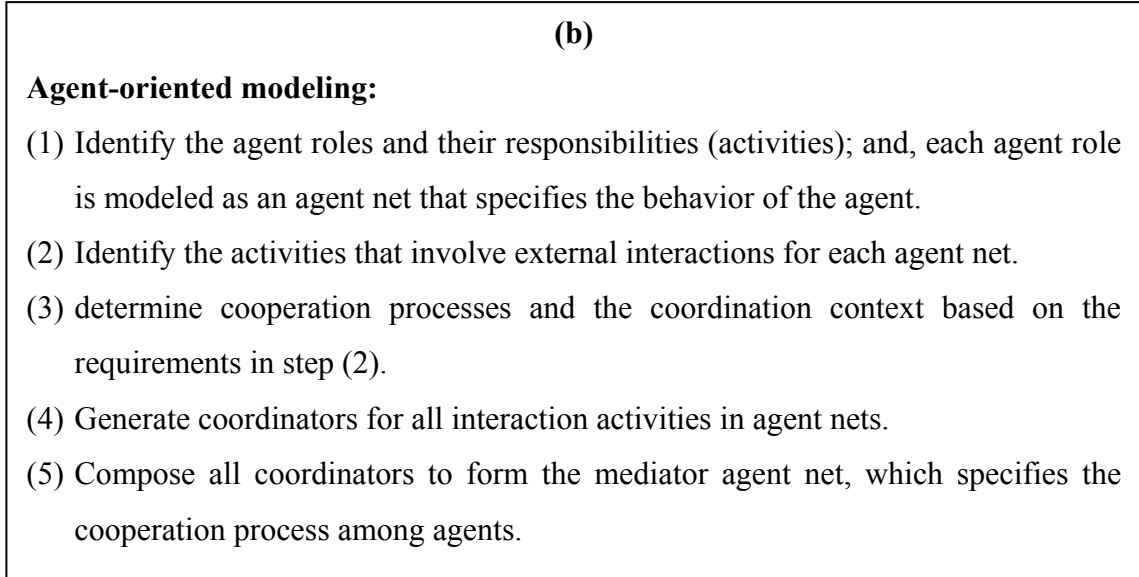
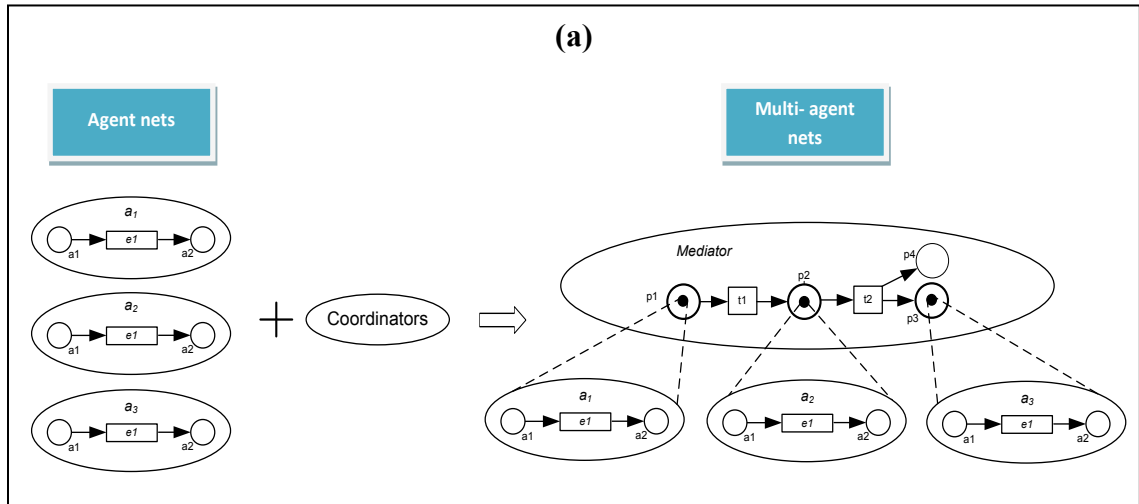


Figure 16. (a) Agent-oriented modeling; (b) essential modeling steps.

In this study, agent-oriented modeling is considered as an approach for constructing abstract models based on the agent concept defined in *Definition 4.1.1*. Therefore, the following major components are supported:

- (1) Agent models can be built individually to meet the requirement of their design objectives;
- (2) An agent model can be constructed with essential features, such as autonomy, proactiveness, reactivity and sociality.
- (3) A coordination model can be constructed to model the cooperation process among agents; and
- (4) Agent models can be coordinated and interact through the coordination model.

Figure 16(a) shows the idea of an agent-oriented modeling approach. First, agent nets are built individually; and next, they are coupled through coordinators. The resulting model is a two-level nested PrT net describing a system model with the MAS architecture. The modeling steps can be summarized into five major steps as shown in Figure 16(b).

#### **4.2 Aspect Orientation of the Internal Structure of an Agent Net**

There are different concerns when designing different types of agents [35]. For example, the concern for designing mobile agents is mobility, and the concern for designing task agents is collaboration. Various concerns are considered as *features* of an agent model. In multi-agent systems, agents are usually designed to solve different problems. That is, each agent has a different action model with respect to its design objective; however, it also may shares common features with the others. For example, an



agent that consumes resources may share an identical plan of getting authorization for using resources.

A variety of agent types substantially increases the complexity of agent design. One of the techniques to manage complexity is modularity. Aspect-oriented programming (AOP) [21] is one of the techniques applied at the implementation stage for modularity. The idea of AOP is to wrap crosscutting concerns into *aspects*, which are desired properties that can be woven into functional components. In recent years, aspect-oriented concepts were introduced into an early stage of system design to address the modularity of abstract models; for example, aspect-oriented modeling for MAS in [48, 49].

In this study, the concept of *aspects* as modular *features* is adopted to address the complexity of building agent models with different internal structures. First, the features shared among agents are identified and specified as aspects. Next, an agent net can be constructed by weaving desired features into the fundamental action model. As a consequence, agent nets are adaptable for different features, thus are more manageable. Figure 17 shows the conceptual model of an aspect-oriented agent net that is composed of the fundamental action model and various aspects.

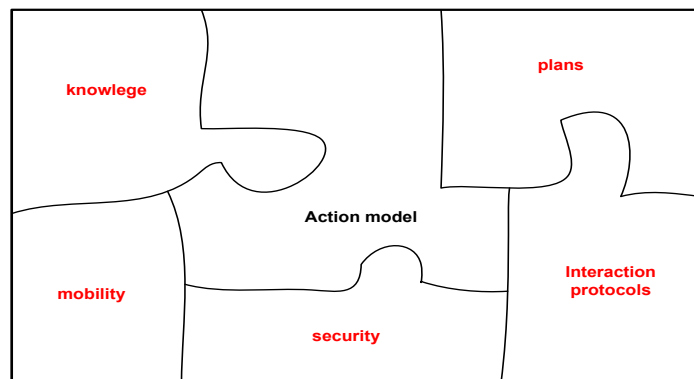


Figure 17. A conceptual model for an aspect-oriented agent net.

Other advantages of an aspect-oriented agent model include net model reusability and flexibility. In terms of model analysis, an aspect-oriented agent model is amenable for incremental analysis by gradually weaving additional aspects or, by weaving different combinations of aspects.

#### 4.2.1 Specifying Aspects

To specify aspects, several terms from AspectJ [22] are used here; however, they are given different meanings for aspect weaving. These terms are defined as follows.

*Definition 4.2.1* An *aspect* is a modular specification, including an *advice*, a set of *pointcuts* and a set of *join points*.

*Definition 4.2.2* An *advice* is a PrT net, which specifies the common behavior shared by multiple agent nets.

*Definition 4.2.3* A *pointcut* specifies a weaving point in the *advice*; it can be a place or a transition.

*Definition 4.2.4* A *join point* specifies a weaving point in the *target* net; it can be a place or a transition.

A specification table is defined to specify an *aspect*. The table includes the essential information defined above. First, the *aspect name* is identified. Second, the *advice* to be woven is defined, including the advice name, the net structure, the semantic definitions and the *pointcuts*. Third, the *names* of target nets and their *joint points* are specified. The *pointcuts* from the advice and the *joint points* from the target nets are the matching points for aspect weaving. The specification is shown in Figure 18(a). An *aspect weaving* is considered as the process of composing two net structures into a single net structure by

connecting the *advice* to the target net specified in an aspect. Syntactically, two nets are connected together by specifying weaving points. Semantically, the semantics defined for weaving points have to be consistent before and after aspect weaving. Note that the semantic definitions of the advice can be delayed until it has been woven. In addition, it is assumed that there is no duplicate name in the woven net. That is, it is assumed that duplicate names with regard to net element definitions of the woven net have been properly resolved.

Let  $N$  be the target agent net, and  $A$  be the aspect specification in which *advice\_name* denotes the advice,  $PC$  denotes the set of pointcuts in *advice\_name*,  $JP$  denotes the set of join points in  $N$ , and  $R$  denotes the set of weaving relations for net  $N$ . A weaving specification  $N: R$  defined in aspect  $A$  weaves *advice\_name* in  $A$  into  $N$  based on the weaving relations specified in  $R$ . A weaving relation  $r$  in  $R$  is a binary relation  $(pointcut_i \rightarrow join\_point_i)$ , where  $pointcut_i \in PC$  and  $join\_point_i \in JP$ ; that is,  $r$  specifies a pair of matching points for an aspect weaving.

An aspect specification is shown in Figure 18(a). An aspect weaving process is shown in Figure 18(b).

There are two possible kinds of join points, namely transition join point and place join point. A place join point is considered as a place where an aspect of alternate choice can be added or, as a place that can hold the tokens generated from an aspect of some extended behaviors. A transition join point is considered as a point where an aspect of some concurrent behaviors can be added or, where an aspect of additional enabling conditions can be added.

**Aspect: A;**

**Advice:** advice\_name;

$P = \{pointcut_1, p1, p2\}; T = \{t1, t2\}; F = \{(pointcut_1, t1), (t1, p1), (p1, t2), (t2, p2)\};$

$\phi(p1) = \phi(p2) = \phi(pointcut_1) = TYPE;$

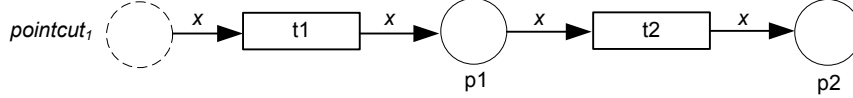
$R(t1) = R(t2) = \lambda;$

$L(pointcut_1, t1) = L(t1, p1) = L(p1, t2) = L(t2, p2) = x;$

$M_0(p1) = M_0(p2) = M_0(pointcut_1) = \{ \};$

**Pointcut:**  $pointcut_1 [, pointcut_2, pointcut_3, \dots];$

**N:**  $pointcut_1 \rightarrow join\_point_1 [, pointcut_2 \rightarrow join\_point_2, pointcut_3 \rightarrow join\_point_3 \dots];$



(a)

**Aspect weaving:**

- (1)  $\forall r \in R$  such that, for all incoming arcs  $(x, pointcut_i)$  and outgoing arcs  $(pointcut_i, x)$  defined in  $F$ , replace  $(x, pointcut_i)$  with  $(x, join\_point_i)$  and  $(pointcut_i, x)$  with  $(join\_point_i, x)$ ; and, replace  $L(x, pointcut_i)$  with  $L(x, join\_point_i)$  and  $L(pointcut_i, x)$  with  $L(join\_point_i, x)$ .
- (2) Discard  $pointcut_i$  such that  $P = P - pointcut_i$ , where  $P$  is the set of places in the advice.
- (3) Weaving advice\_name into  $N$  such that:  $N.P = N.P \cup P$ ;  $N.T = N.T \cup T$ ;  $N.F = N.F \cup F$ , where  $N.P$  is the set of places in target net  $N$ , and  $N.T$  is the set of transitions in target net  $N$ .

(b)

Figure 18. (a) An aspect specification table; (b) an aspect weaving process.

In addition to the previous example, some weaving patterns are generalized and shown in Figure 19, where (a), (b) and (c) are patterns of *transition join point* since the weaving point is at a *transition*; and, the patterns in (d), (e) and (f) are *place join point* since the weaving point is at a *place*. Patterns (a) and (d) are similar to *after advice* in

AOP; (b) and (e) are *before advices*; and, (c) and (f) are *around advices* that can be added as an explicit control of the net. Intuitively, during a weaving process, a transition join point must be connected with a place in the advice and a place join point must be connected with a transition in the advice. This is to ensure the correctness of the syntax and static semantics of a woven net.

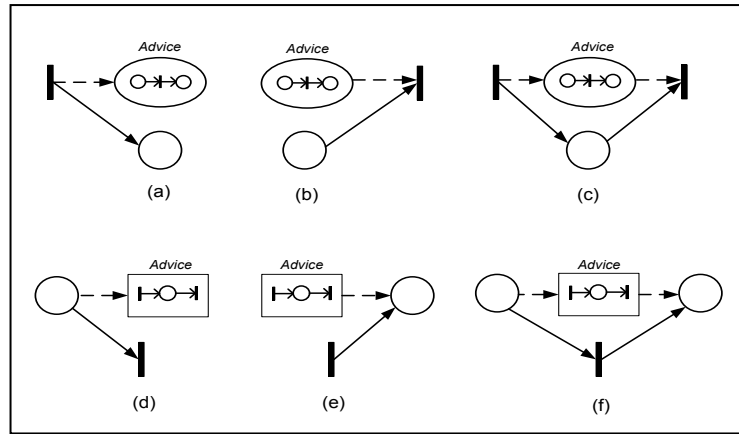


Figure 19. Weaving patterns.

### 4.3 Modeling a Single Agent Net

An agent net specifies the behavior of a distributed computation entity in the MAS context. It describes an agent's behavior without a centralized control. That is, it has the control over its own internal states. For modeling agent communications, channel commands are instrumented in transition constraints to specify external message exchanges. It is assumed that agent nets are communicating within the same context that has been defined in the semantic domain *Spec*, and are interpreting the information that is relevant to their computations based on their best interest. Furthermore, token types defined for exchanging messages between agent nets must be consistent. As a result, the messages exchanged among agent nets are in a simpler structure, and are only relevant to

the behavior of associated agent nets. This principle avoids the redundancy of irrelevant details and unnecessary complexity with regard to the interpretation of messages. The modeling steps for an agent net with communication channels can be summarized in Figure 20.

- (1) Identify the actions engaged for each agent and represent each action as a transition within an agent net; that is, define  $T$ .
- (2) Identify the pre-conditions and post-conditions for each transition in  $T$ ; and, add associated input places, output places, arcs and transition constraint for all transition  $t$  in  $T$ , respectively; that is, define  $P$ ,  $F$  and inscription  $ins$ .
- (3) Identify the transitions with external interactions in the net, and add associated channel commands; that is, define  $T_c$  in  $T$ , and  $\forall t \in T_c. (R(t) = R_u(t) \wedge R_c(t))$ , where  $R_c(t) = n!e \mid n?x$ ;  $n$  is the identification of the mediator agent net.
- (4) Draw an *interaction elicitation table* (IET), which contains four columns:
  - (i) all  $t$  in  $T_c$  identified in step (3),
  - (ii) the direction of the information flow associated with  $t$ ,
  - (iii) the exchanging information associated with  $t$ , and
  - (iv) the elicited channel command associated with  $t$  based on (i), (ii) and (iii).

Figure 20. Modeling steps for an agent net with communication channels.

#### 4.3.1 Modeling Examples

A Gas Station scenario is used as an example to demonstrate the construction of an agent net.

##### *A Gas Station Scenario*

For the operation of a gas station, there are gas consumers, gas suppliers, the bank and the gas station itself. Gas consumers pump gas for their cars in order to accomplish

their plans, while gas suppliers produce gas in order to supply gas to the gas station. The bank provides banking services, including credit card authorizations that allow customers to make transactions when pumping gas. The gas station provides an environment for these activities. There are pumping stations in the gas station where the cars entering the station can park and pump the gas. The gas pumping process includes four major steps: (1) the consumer who is driving a car and entering the station can park the car at one of the pumping stations that are available; (2) after parking at one of the pumping stations, the consumer must slide their credit card first in order to get the authorization for pumping gas; (3) if authorized, the consumer can start pumping gas with the choice of regular or diesel gas; and, (4) the consumer finished pumping and left the station.

Based on the above scenario, there are five different agent roles: (1) regular gas consumer, who uses a vehicle to commute between home and school; if the car is out of gas, he goes to the gas station and pumps regular gas; first, he needs to find an available pumping station, and then to slide his credit card in order to pump gas; (2) diesel gas consumer, who uses a vehicle to transport goods between the factory and the store; if it is out of gas, he goes to the gas station and pump diesel gas; he also needs to find an available pumping station first, and then slides his credit card in order to pump gas; (3) gas producer, who produces both regular and diesel gas based on orders; (4) the bank, which provides the credit card transaction service that checks credits and reports credits; and, (5) the gas station, which is served as the mediator agent that provides the global view of the gas pumping process in which multiple agents are engaged in.

*Example 1:* Modeling an agent net with communication channels.

Let us take the *diesel gas consumer* as an example to build the agent net based on the steps in Figure 20.

Step (1) define  $T$ ;

- According to the scenario, let  $T = \{ToStore, ToGasStation, Go, ToFactory, Park, SlideCard, PumpGas, Deny\}$ ;

Step (2) define  $P$ ,  $F$ , and  $ins = (\varphi, R, L, M_0)$ ;

- Let  $P = \{factory, store, GasStation, ToPump, standby, pumped, CreditCard\}$ ;
- $\varphi(factory) = \varphi(store) = \varphi(GasStation) = CAR \times INTEGER$ , where the first element of the data token is the car type and the second element indicates the condition of the gas tank (1 represents full tank, 0 otherwise), and  $CAR = \{sedan, truck\}$ ;  $\varphi(standby) = \varphi(ToPump) = \varphi(pumped) = CAR \times INTEGER \times INTEGER$ ;  $\varphi(CreditCard) = INTEGER$  denoting credit card numbers;
- $R$ : see Appendix A1;
- Let  $M_0(factory) = \{<truck, 0>\}$  denoting that in place *factory* there is a truck with empty tank,  $M_0(credit\_card) = \{<1>\}$  stores the credit card number, and all other places in  $P$  such that  $M_0(p) = \emptyset$ ;
- $F$  and  $L$  are defined accordingly.

Step (3) define  $T_c$ ;

- Based on the scenario,  $T_c = \{Park, SlideCard, PumpGas, Fail\}$ , while  $T_u = \{ToStore, ToGasStation, Go, ToFactory\}$ ; and let the identification of the mediator net be  $S$ ;



- The *interaction elicitation table* (IET) with four columns can be drawn based on step (1) ~ (3) to generate channel commands associated with  $T_c$ ; the table is shown in Table 2. Column  $R_c(t)$  contains the elicited channel commands, which can be added to associated transition constraints.

Table 2. Interaction elicitation table.

$t \in T_c$	Directions	Exchanging information	$R_c(t)$
Park	input	pumping station number	$S?st$
SlideCard	output	car type, credit card number and pumping station number	$S!<car, cr, st>$
PumGas	input	diesel gas	$S?g$
Deny	input	invalid credit card number	$S?cr$

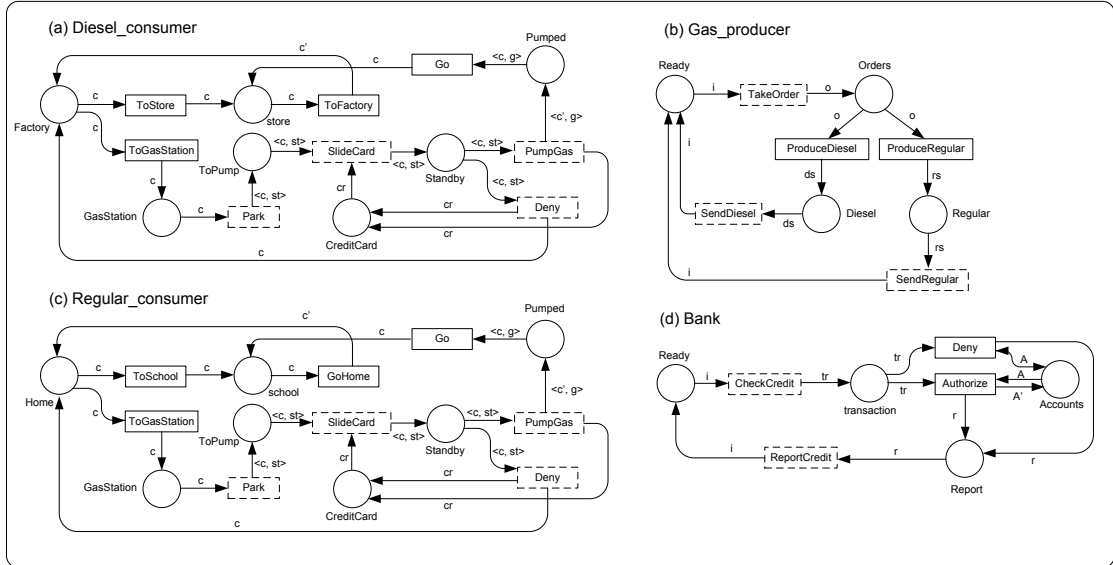


Figure 21. Agent nets in the gas station scenario.

Other agent nets can be built accordingly. The agent nets are shown in Figure 21, respectively. The rectangles with dashed line in Figure 21 denote the transitions involving external interactions. Detailed semantic definitions for the agent nets in Figure 21 are given in Appendix A1.

*Example 2: Aspect orientation and aspect weaving.*

From the above example, Diesel\_consumer agent and Regular\_consumer agent share the common behavior of pumping gas since they are gas consumers. Therefore, the pumping gas process can be modularized as an *aspect*. The specification of a *Pumping gas* aspect is shown in Figure 22(a). While the *Pumping gas* aspect is shared by two different agent nets to enforce an explicit control for transportation, the *Car wash* aspect shown in Figure 22(b) is an extended feature for gas consumer agents.

For example, let Figure 23(a) be the basic action model for a Regular\_consumer agent that regularly commutes between home and school. Let us assume that we want to add an explicit control to make sure the transportation only happen when the tank is full. In this case, the control can be done by weaving the *Pumping gas aspect* into its basic action model. Let us further assume that a Regular\_consumer agent is given an additional option to wash the car when in the gas station. In this case, the agent plan can be extended by weaving the *Car wash aspect* into its basic action model. The woven net is shown in Figure 23(b).

The weaving process based on Figure 18(b) is as follows.

Step (1) Replace arc (pcut\_1, ToGasStation) in *adv1* with (home, ToGasStation), and arc

(Go, pcut\_2) with (Go, School).

Step (2) Remove pcut\_1 and pcut\_2 from *P*.

Regular\_consumer. $P$  = Regular\_consumer. $P \cup adv1.P$ ;

Regular\_consumer. $T$  = Regular\_consumer. $T \cup adv1.T$ ;

Regular\_consumer. $F$  = Regular\_consumer. $F \cup adv1.F$ ;

Step (3) Add semantic definitions for the woven Regular\_consumer net.

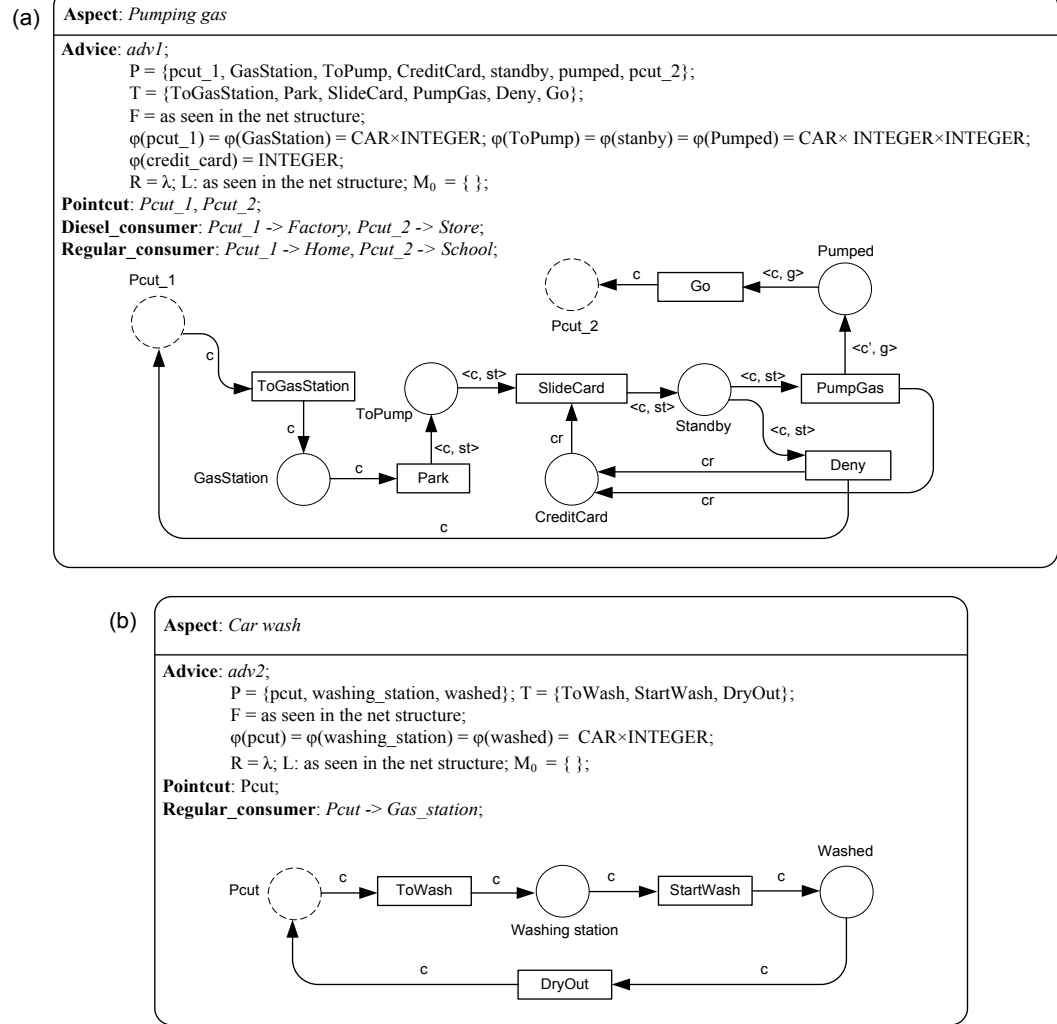


Figure 22. (a) The Pumping gas aspect; (b) the Car wash aspect.

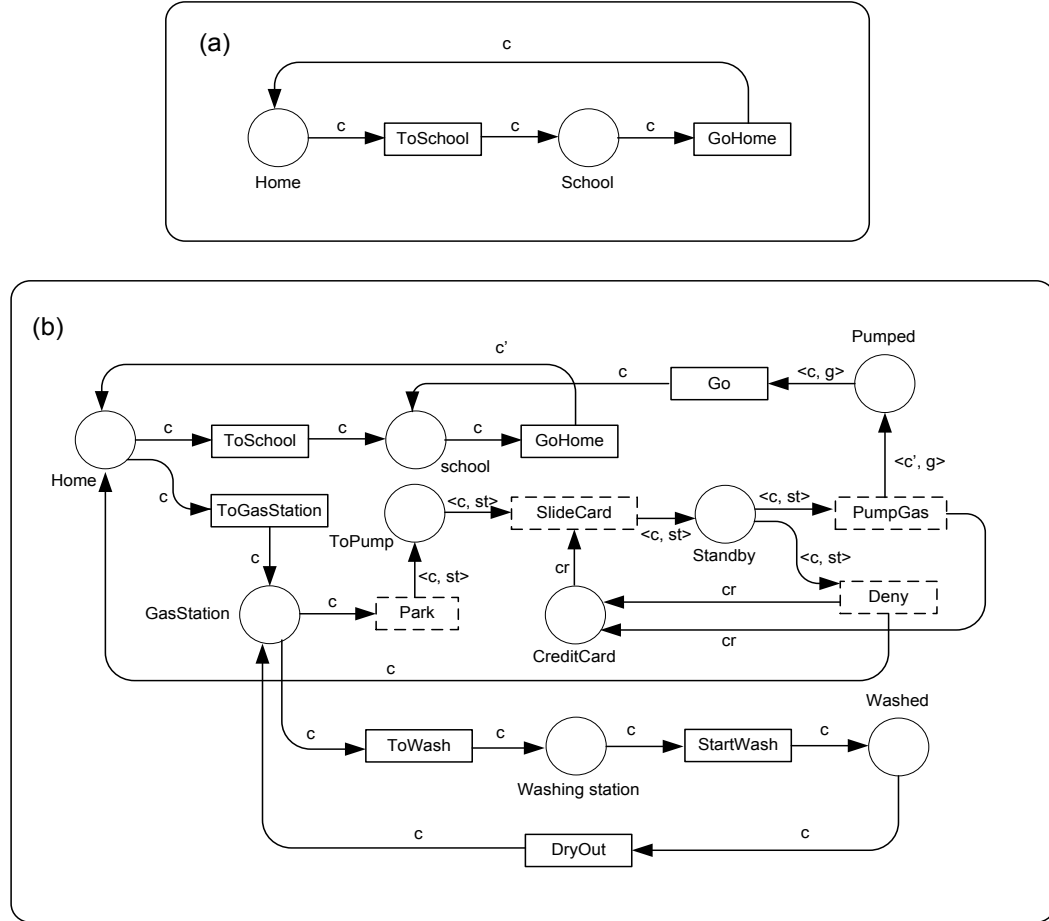


Figure 23. (a) The basic net for a Regular\_consumer agent; (b) the woven net.

#### 4.3.2 Constructing an Agent Net with the BDI Model

There are several concrete structures for agent models, such as *logic based* agents, *reactive* agents and *belief-desire-intention* (BDI) agents. However, the BDI model [59] has been widely adopted [33, 40, 59, 60, 80, 81] to build concrete agent models with rational behaviors. *Beliefs* refer to the information that an agent has about the situated environment; *desires* are agent goals that an agent would like to achieve; *Intention* are the choices with commitment [82]. The main idea of the BDI model is to explicitly represent beliefs, desires and intentions as the internal structure of an agent model to address agent autonomy [16].

A BDI model can be nicely modeled using PrT nets [83]. The *beliefs* are considered as internal states, which are markings of the net; the *desires* are goals, which are a set of reachable markings with respect to some initial marking; and the *intentions* are choices to reach the goals with respect to current marking. That is, an agent net is a plan that includes a set of transition sequences that can reach some goals with respect to some markings. For example, John's traveling plan is shown in Figure 24. John is currently at Miami and intends to go to Los Angeles. Los Angeles is a *goal*. Nevertheless, there are two paths available from current location Miami to Los Angeles, namely: (1) Miami-Houston-Los Angeles, and (2) Miami-Atlanta-Los Angeles. Thus, the set of transition sequences  $\sigma = \{\sigma_1, \sigma_2\}$ , where  $\sigma_1 = M_0[t1 > M_1[t3 > M_2]$ ,  $\sigma_2 = M_0[t2 > M_1[t4 > M_2]$  and the set of reachable markings  $[M > = \{M_0, M_1, M_2\}$ .  $[M >$  denotes the *beliefs*,  $\sigma$  denotes the *agent plan*,  $M_2$  denotes the *desire*, and transition sequences  $\sigma_1$  and  $\sigma_2$  denote the *intentions*. The net structure in Figure 24 exhibits the non-determinism that addresses the autonomy of path selection to reach the agent goal (Los Angeles).

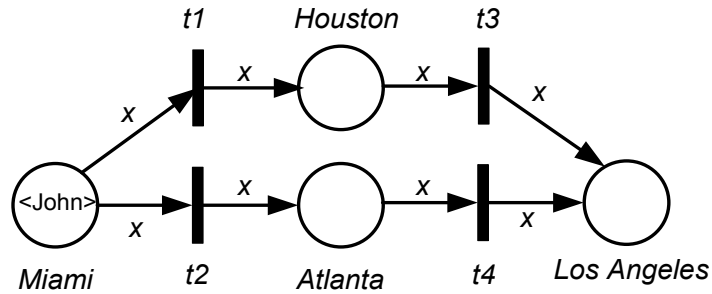


Figure 24. John's traveling plan.

Let  $\sigma$  be the set of transition sequences with respect to marking  $M_0$ , and  $[M >$  be the set of markings reachable from  $M_0$ . A BDI PrT net can be formally defined as follows.

*Definition 4.3.1* An *agent plan* is a net structure  $N = (P, T, F)$ .

*Definition 4.3.2* The *beliefs* of an agent net  $A$  with respect to an agent plan  $N$  is the set of reachable markings  $[M_0 >$  with respect to  $M_0$ .

*Definition 4.3.3* A goal of an agent is a goal state  $M_g$  of an agent plan  $N$ , where

- $M_g$  is a member of  $[M_0 >$ ;
- The set of goals  $M_G = \{M_{g1}, M_{g2}, \dots, M_{gn}\}$  is called the *desires* of an agent plan  $N$ , where  $M_G \subseteq [M_0 >$ .

*Definition 4.3.4* An *intention* is an execution sequence  $\sigma_i = M_0 t_1 M_1 t_2 \dots M_{gi}$ , where

- $\sigma$  is the set of possible execution sequences of agent plan  $N$ , and  
 $\sigma = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$ ;
- $\sigma_i \in \sigma$ ; and
- $M_0$  is the initial state and  $M_{gi}$  is the goal state.

A BDI PrT net can be constructed by the following steps:

- (1) Identify predicates.
- (2) Identify transitions.
- (3) Define each intention by connecting relevant predicates and transitions.
- (4) Combine all intentions to form the agent plan.

Given an initial marking describing initial beliefs, the agent plan can be checked if the goals are reachable.

#### 4.4 Coordination Modeling

Agents are autonomous and heterogeneous computation entities that are usually built independently. Therefore, one of the major concerns of designing MAS is the coherence of overall system. The coherence of the system usually relies on some mediator, which is employed for task sharing and resource sharing among agents. The process of task sharing [2] is considered as agent *cooperation*. The activities of coupling agents during the cooperation process are considered as agent *coordination*. Agent *communications* are two-way message exchanging activities to facilitate the coordination within an agent community. As such, a mediator agent net is employed to model the cooperation and coordination among agent nets (see the conceptual model in Figure 4). The idea is to coordinate individually modeled agent nets to constitute the global process.

Let  $S$  be a mediator agent net as defined in Definition 3.3.2. The mediator agent net  $S$  can be constructed by establishing a set of *coordinators* for coupling agent nets. A *coordinator* in the mediator agent net manages either an input or output information (resources) associated with an interaction activity of an agent net. A coordinator is formally defined as follows.

*Definition 4.4.1* A coordinator  $C$  is a PrT net with net structure  $(P_c, t_c, F_c)$  where:

- $P_c = P_d \cup P_a$ , where  $P_d$  is the set of places holding data tokens and  $P_a \neq \emptyset$ ;
- $P_a$  is the set of places holding agent net tokens and  $P_d \neq \emptyset$ ;
- $t_c$  is a communication transition such that  $R_c(t_c) \neq \lambda$ ; and
- $F_c$  is the flow relation indicating the information flows associated with transition  $t_c$ .

#### 4.4.1 Coupling Agent Nets

Figure 25 shows two different coordinators in which Figure 25(a) is an input channel coordinator, while Figure 25(b) is an output channel coordinator. Coordinators for all agent nets can be built by drawing a *coordination elicitation table* (CET). The CET provides an intermediate step to build the mediator agent net.

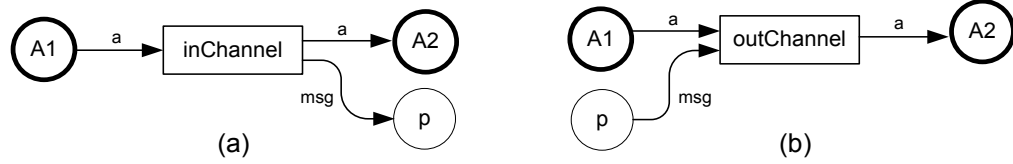


Figure 25. (a) An input channel coordinator; (b) an output channel coordinator

*Definition 4.4.2* A *coordination elicitation table* (CET) is a seven-columned table, where:

Column 1.  $N.R_c(t)$  is the communication constraint of transition  $t$  in an agent net  $N$ ;

Column 2.  $C.t_c$  is the *communication* transition  $t_c$  of the coordinator  $C$ ;

Column 3.  $C.R_c(t_c)$  is the *communication* constraint of transition  $t_c$  of the coordinator  $C$  derived from  $N.R_c(t)$ ;

Column 4.  $C.P_d$  is the set of places holding data tokens associated with  $t_c$  of  $C$ ;

Column 5.  $C.P_a$  is the set of places holding agent tokens associated with  $t_c$  of  $C$ ;

Column 6.  $C.\bullet t_c$  is the preset of  $t_c$ ;

Column 7.  $C.t_c\bullet$  is the post set of  $t_c$ .

The drawing steps of a CET are shown in Figure 26(a). Each entry in the CET represents a coordinator accommodating an interaction with an agent net. Therefore, each



entry constitutes either an input channel coordinator (Figure 25(a)) or an output channel coordinator (Figure 25(b)). A coordinator in the mediator agent net manages either an input or output information (resources) associated with an interaction activity of an agent net. The mediator agent net  $S$  is generated by merging all coordinators in the CET. The generating steps are shown in Figure 26(b).

*Example 3: Modeling the mediator agent net (the gas station).*

First, let us look at an example to build coordinators using the steps in Fig. 26(a) for the Diesel\_consumer agent net shown in Figure 21(a).

Step (1) Based on Table 2, there are four entries of  $R_c(t)$  in Diesel\_consumer agent net:

$S?st, S!<car, cr, st>, S?g, S?cr$ ; thus, the first column of the coordination elicitation table is filled with these entries.

Step (2) For each entry in the first column, the transition  $t_c$  is added for coupling.

Step (3)  $R_c(t_c)$  of coordinator  $C$  can be defined based on column one pairing with  $R_c(t)$  of agent net  $N$ .

Step (4) Define  $P_a$  and  $P_d$  based on Step (3).

Step (5) Define  $F$  based on column  $R_c(t_c)$ .

Table 3 shows the CET in which each entry produces one coordinator. Thus, there are four coordinators for the Diesel\_consumer agent net. Each coordinator is transformed into a net structure according to table 3. These coordinators are shown in Figure 27(a).

<b>(a) Generating the coordinators by a coordination elicitation table (CET):</b>						
(i) For each agent net $N = (P, T, F)$ in the multi-agent nets $MAS$ , list $R_c(t)$ for all $t$ in $T$ in the first column of the table.						
(ii) Add associated transition $t_c$ in the second column for paring with each $R_c(t)$ listed in the first column.						
(iii) Define $R_c(t_c)$ in the third column by the following rules:						
if $N.R_c(t) = S?msg$ , then $C.R_c(t_c) = N!msg$ ;						
if $N.R_c(t) = S!msg$ , then $C.R_c(t_c) = N?msg$ ;						
(iv) Add a place $p$ to $P_d$ for holding $msg$ , an input place $A1$ and an output place $A2$ to $P_a$ for holding net $N$ ; if necessary, add other places for holding data tokens.						
(v) Determine $\bullet t_c$ and $t_c \bullet$ based on the following rules;						
if $C.R_c(t_c) = N?msg$ , then $\bullet t_c = \{A1\}$ , $t_c \bullet = \{A2, p\}$ , $F = \{ (A1, t_c), (t_c, A2), (t_c, p) \}$ ;						
if $C.R_c(t_c) = N!msg$ , then $\bullet t_c = \{A1, p\}$ , $t_c \bullet = \{A2\}$ ; $F = \{ (A1, t_c), (t_c, A2), (p, t_c) \}$ ;						
<b>(b) Generating the mediator agent net from CET:</b>						
(i) Merge all coordinators by removing redundant places and redirecting the arcs to associate transitions.						
(ii) Add local transitions and places to $S$ based on the requirements for modeling the cooperation process.						
(iii) Define $\varphi$ , $R$ , $L$ and $M_0$ for $S$ accordingly.						

Figure 26. (a) The steps for generating CET; (b) the steps for generating the mediator agent net.

Table 3. The coordination elicitation table.

$N.R_c(t)$	$C.t_c$	$C.R_c(t_c)$	$C.P_d$	$C.P_a$	$C. \bullet t_c$	$C. t_c \bullet$
$S?st$	Park	$N!st$	{pumping_stations}	{in_station, parked}	{pumping_stations, in_station}	{parked}
$S!<car, cr, st>$	Pay	$N?<car, cr, st>$	{transactions}	{parked, waiting}	{parked}	{waiting, transactions}
$S?g$	PumpDiesel	$N!g$	{authorized, diesel_gas}	{waiting, pumped}	{authorized, diesel_gas, waiting}	{pumped}
$S?cr$	Fail	$N!cr$	{authorized}	{ waiting, parked}	{authorized, waiting}	{parked}

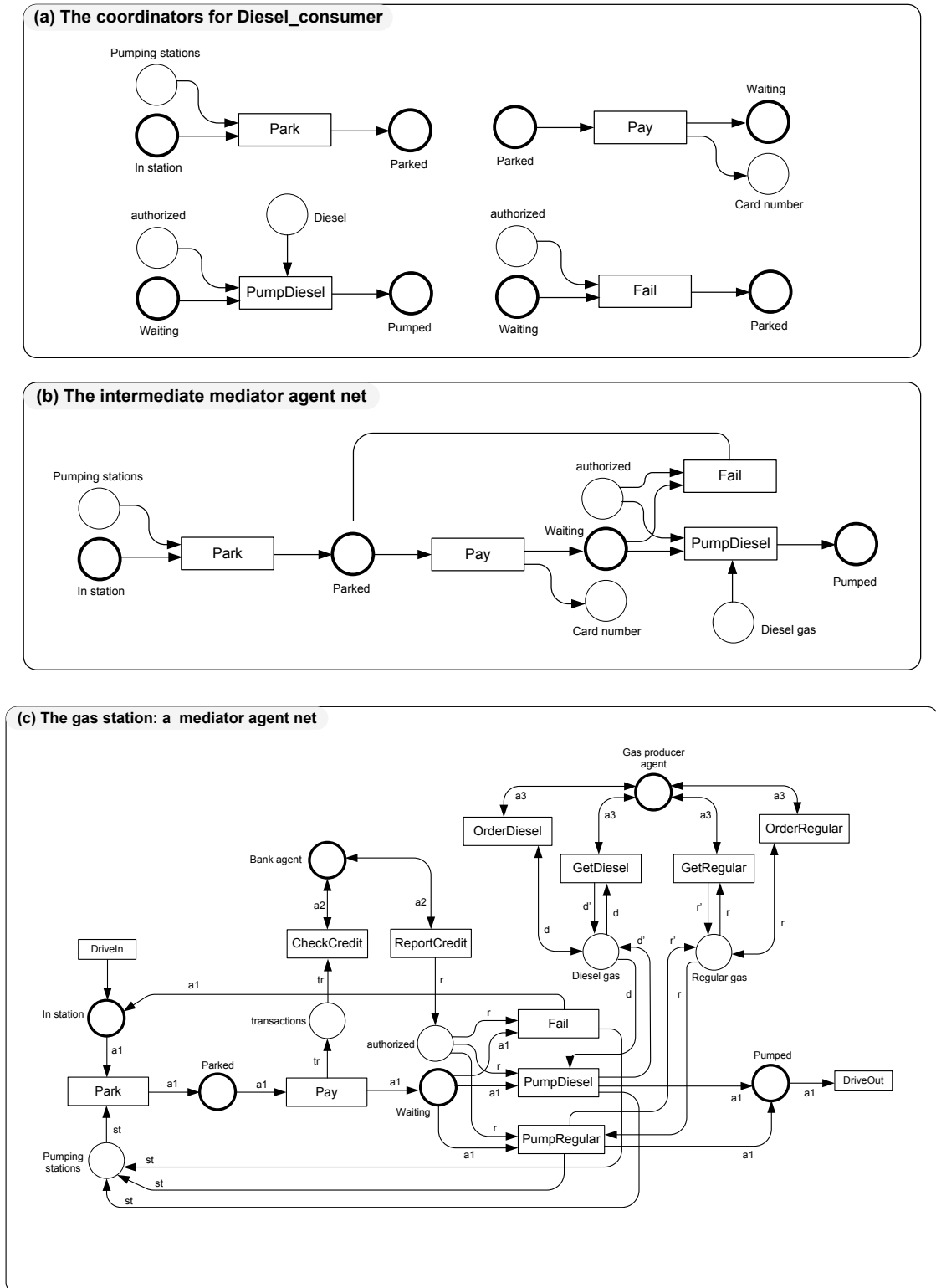


Fig. 27. (a) The coordinators of the Diesel\_consumer agent net; (b) the intermediate net; (c) the mediator agent net.

Next step is to merge all coordinators to form a single net structure based on the steps in Figure 26(b). The coordinators shown in Figure 27(a) are merged by the following steps.

Step (1) Merge all coordinators in Table 3; Figure 27(b) is the intermediate mediator agent net by merging all coordinators in Figure 27(a); coordinators for other agent nets in Figure 21 can be built and merged in the same manner.

Step (2) Merge all coordinators for all agent nets result in the mediator net shown in Figure 27(c). Two boundary transitions “drive\_in” and “drive\_out” are added to denote the entering and leaving of agent nets.

Step (3) Define the semantic definitions for the net (see Appendix A2).

#### **4.5 Summary**

In an effort to ease the construction of formal MAS models, this chapter presents a systematic approach for modeling a single agent net with an aspect-oriented approach, and for modeling the mediator agent net to coordinate agent nets.

Although there were several research works incorporated aspect-oriented concepts into PrT nets, their focus was not on MAS modeling. For example, in [84, 85], aspect-oriented concepts were used to address security concerns based on PrT nets. Security concerns were modeled as aspects and woven into a base net to generate a secured PrT net model. These works were not related to MAS modeling and were limited to security issues.

Among research works based on UML for modeling MAS, the work in [86] proposed an aspect-oriented agent architecture based on UML (Unified Modeling Language). In

this work, essential agent concerns were separated from functional components and modeled as aspectual components. In [48], model roles were defined to model aspects. In [49], a meta-modeling framework was defined to include aspect-oriented concepts. A crosscutting composition mechanism was provided to compose agent models and aspects. These work [48, 49, 86] were based on informal methods and focused on guidelines and steps for the aspect orientation of an agent program's internal structure. Their models did not support formal analysis.

## **CHAPTER 5**

### **A METHOD FOR ANALYZING FORMAL MAS MODELS**

Given that there is no existing method for the analysis of nested PrT nets, a model transformation technique is developed in this dissertation research to transform a MAS model with a two-level nested PrT net structure into a PROMELA program in SPIN (Simple PROMELA Interpreter) [38]. As a consequence, a full line of functionalities in SPIN can be utilized for the analysis of formal MAS models.

Among research works for the analysis of high-level nets, there were two major approaches adopted: (1) simulation-based model analysis, and (2) model checking. The most commonly-used tool for the first approach is the CPN (Colored Petri Nets) tool [79]. CPN tool is a well-developed simulation tool based on Colored Petri nets. However, this dissertation research aims at modeling checking formal MAS models. Therefore, the model checking approach is adopted. There are two renowned model checkers: (1) SMV (Symbolic Model Verifier) [15], and (2) SPIN [23]. SMV is a tool for checking whether or not a finite-state system satisfies specifications given in CTL (Computation Tree Logic) [87]. SMV has been very successful in verifying hardware systems, however suffers from the state-explosion problem in verifying software systems. SPIN is a model checking tool based on the partial order reduction method, which is aimed at reducing the size of the state space needed to be explored. The SPIN verifier checks abstract models written in PROMELA that if a given PROMELA model satisfies the claims given in LTL (Linear Temporal Logic) [39]. SPIN is a well-suited tool for this dissertation research based on two main reasons: (1) it is a well-developed tool for model checking concurrent and

asynchronous software systems; and (2) it supports the modeling of asynchronous process interactions which is an important feature for studying multi-agent systems. Therefore, the idea is to transform the formal MAS model into a PROMELA model that can be analyzed by the model checker SPIN.

Previous works in model translation from Petri nets to executable models generally fall into two major categories: (1) translation of Petri nets to high-level programming languages [72, 73, 74, 75]; and (2) translation of Petri nets to the meta-language supported in simulation tools [76, 77, 78, 79]. The authors in the first category attempted to use Petri nets as a central means during a model-driven system engineering process [88], and to generate an implementation dependent prototype from an implementation independent model. The works in the second category, however, focused on the validation of system design in critical aspects prior to implementation. The model transformation in this dissertation research aims at providing a method for verifying the proposed two-level nested PrT nets. Thus, a set of translation rules are explicitly defined for model transformation. The transformation technique provides a foundation for further automation in developing the tool for analyzing nested PrT nets.

The rest of this chapter is organized as follows. Section 5.2 introduces PROMELA and its semantics engine. Section 5.3 elaborates the translation rules for model transformation. Section 5.4 provides the proof of correctness regarding the translation. Section 5.5 presents a translation example from a disaster mitigation system. Section 5.6 demonstrates a method for analyzing the transformed PROMELA model using SPIN. Section 5.7 draws the conclusion.

## 5.1 The Target Language PROMELA

The specification language used in SPIN is called PROMELA, in which the focus is on specifying the controls, rather on the computations, of distributed systems. The program structure and semantics engine of PROMELA are briefly introduced in the following sections.

### 5.1.1 The Program Structure

A PROMELA model is constructed from three basic types of objects: (1) processes, which define the behaviors of distributed entities; (2) data objects, which define the variables for keeping information; and, (3) message channels, which model the exchange of information between processes. Figure 28 shows a generic PROMELA program structure. The detailed syntax and grammar rules of PROMELA can be found in [38].

```
#define MAX_TOKENS 10
mtype = { .... }           /* defines global data objects */
typedef TYPE { .... }
inline FUNC { ... }         /* defines macros */
chan CHANNEL = [0] of { .. } /* define global message channel */

active proctype MAIN_PROCESS ( ) {
    /* statement sequence */
}

proctype PROCESS1 ( ) { /* statement sequence */ }
proctype PROCESS2 ( ) { /* statement sequence */ }
.....
```

Figure 28. A generic PROMELA program structure.



### 5.1.2 The Semantics

In a PROMELA program, each declaratory *proctype* defines a process. Before model execution, each process is transformed to a *FSA* (finite state automaton) describing the execution sequences of that process. A *FSA* is a tuple  $(S, s_0, L, T, F)$  [38], where

- $S$  is a set of states denoting the possible points of control within a *proctype*;
- $s_0 \in S$ , is a distinguished initial state;
- $T$  is a set of transition relations denoting the flow of controls, and  $T \subseteq (S \times L \times S)$ ;
- $L$  is a set of labels that link each transition in  $T$  with a specific basic statement that defines the executability (pre-conditions) and the effect (post-conditions) of that transition; only six basic statements are allowed as valid labels: *print*, *receive*, *send*, *assignment*, *assertion* and *expression*, where *print* and *assignment* statements are unconditionally executable; and,
- $F$  is a set of final states, and  $F \subseteq S$ .

The global behavior of a concurrent system described by a PROMELA program is obtained by computing an asynchronous interleaving product of automata. The resulting system behavior is also represented by an automaton. In the initial system state, all processes are in their initial state, and all data objects are set to their initial values. The semantics engine in SPIN executes a PROMELA model in a step by step manner. In each step, one executable basic statement (transition) is selected out of the transitions in all active processes. If more than one statement is executable, any one of them can be selected randomly (non-determinism). Depending on the system state, any statement in a SPIN model is either *executable* or *blocked*; that is, if a process reaches a point where no

executable transition left to be executed, it is simply blocked. On the other hand, as long as there are executable transitions, the semantic engine repeatedly selects one of them at random and executes it. The execution of a transition is to apply the *effect* (post-conditions) defined in that transition. As a result, system variables, local variables, and the contents of channels may be modified. By simulating the execution of a PROMELA model, a large directed graph including all reachable system states is generated. Figure 29 shows the operational model of the PROMELA semantics engine.

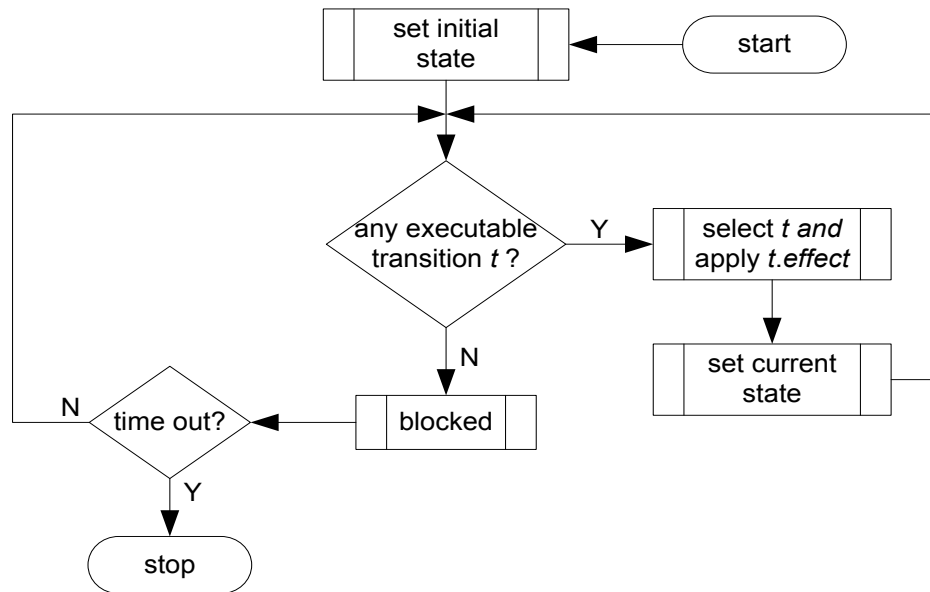


Figure 29. The operational model of the PROMELA semantics engine.

## 5.2 A Translation Method

The translation for model analysis is aimed at providing a method for analyzing the proposed two-level nested PrT nets. Therefore, the overall translation principles and assumptions are discussed as follows.

(1) *No embedded C codes.*

Since the objective of this work is to provide a means to model and verify the properties of abstract models at the system design stage, the principle is to build a smaller sufficient abstract model and to avoid possible redundancies. Despite the full description power of C language, no embedded C code is considered in this translation.

(2) *Restricted PrT nets.*

PrT nets are very expressive given that: (i) there is no explicit definition regarding the limitation of the quantity of tokens in a place; (ii) the sorts and their operations are implicitly defined in the semantic domain *Spec*; (iii) testing enabling conditions of transitions and instantiation of tokens are implicit; and, (iv) the firing sequences can be infinite as long as there are sufficient tokens. On the other hand, a PROMELA model is an executable program, in which the quantity and types of data objects are bounded and the execution is finite. Therefore, due to tractability, the expressive power of PrT nets needs to be restricted by limiting the sorts and the quantity of tokens in each place. As a result, the nested PrT nets to be translated have a finite state space such that model executions will terminate appropriately.

(3) *Interleaving semantics within a net entity.*

For simplicity, it is assumed that all transitions fire immediately after the guard conditions are evaluated to be true, and the firings are interleaving given that this restricted semantics does not affect the verification of state-based properties.

(4) *Communication channels are communicating in a one-to-one and unidirectional fashion.*

Broadcasting is not considered in this translation; however, it can be done through an appropriate setup of a loop-statement construct for multiple communications.

(5) *Machine analysis.*

This translation aims at the verification of the proposed two-level nested PrT nets in order to facilitate the machine analysis of multi-agent systems prior to implementation.

(6) *Correctness of translation.*

To justify the correctness of this translation, a translation from a two-level nested PrT net to a PROMELA program is said to be (i) *complete*, if all the net entities and net elements in a nested PrT net are faithfully translated to a set of non-overlapping statements in the target PROMELA program; (ii) *consistent*, if the target PROMELA program preserves the dynamic semantics of the nested PrT net; and, (iii) *correct*, if the translation is complete and consistent.

### 5.2.1 Translation Rules

The nested PrT net to be translated is called a *multi-agent net*. Each net entity in a multi-agent net is translated to a process in the target PROMELA program. As a result, a process describes the behavior of an agent, which is an autonomous entity and capable of interacting with the others. Each net element of a net entity is translated to a set of non-overlapping compound statements within a process to address the semantics of the net entity. The static semantics of a net entity is defined by the inscription  $ins = (\varphi, L, R, M_0)$ , while the dynamic semantics is defined by the transition enabling and firing rules in Chapter 3. Therefore, there are net entity translation rules and net element translation rules. The mapping relations are summarized in Table 4.

Table 4. Mapping relations of net elements to PROMELA objects.

PrT net elements	PROMELA objects
Net	Process
Place	Array data object
Transition (event)	Guard conditions → statement sequence
Communication channels	Message channels

#### A. Net entity translation rule

Let  $MAS$  be a set of PrT nets specifying a multi-agent system, including a mediator agent net and multiple agent nets. Each member in  $MAS$  is translated to a process object in the target PROMELA program. The rule for translating net entities in  $MAS$  to their counterparts in the target PROMELA program is called net entity translation rule. The rule is defined as follows.

**Rule e.1:** For every net entity  $N \in MAS$ , add a process **proctype** [active]  $N( ) \{ \}$  to the target PROMELA program, where  $N$  is a unique process name.

For example, let  $MAS = \{S, agent_1, agent_2\}$ , the translated skeleton program in PROMELA is as follows.

```

active proctype  $S( ) \{ statement\ sequence \}$ 
proctype  $agent_1(argument\_list) \{ statement\ sequence \}$ 
proctype  $agent_2(argument\_list) \{ statement\ sequence \}$ 

```

The keyword *active* in the declaratory statement denotes an immediate instantiation of the process when the program starts to run. The system process is required to be instantiated immediately as soon as the program starts to run, while agent processes are not. Instead of declaring as active processes using the declaratory keyword *active*, agent processes can be dynamically instantiated during model execution by the statement “*run agent<sub>i</sub>(argument\_list)*”, in which the name of the agent process and arguments representing the initial marking are instantiated through the operator *run*. The maximum number of active processes in PROMELA is 255; that is, if  $|MAS| = n$ , then  $2 \leq n \leq 255$  given that *MAS* has at least two members and at most 255 members.

## B. Net element translation rules

Net inscription  $ins = (\varphi, L, R, M_0)$  specifies the static semantics of a net entity  $N = (P, T, F)$  with respect to the semantic domain *Spec*. The translation rules for translating net elements *P*, *T*, *F* and inscription *ins* are called net element translation rules.

### B.1. Place Transition Rules

In PrT nets, a place *p* in *P* is a predicate denoting a relation among individuals. Thus, tokens in *p* are instantiations of individuals. An arc label in *L* specifies the variable extension of a place *p* to which the arc is connected. A consistent substitution of the labeled variables on an arc is an instantiation of a particular token in *p*. For example, a token  $\langle a_1, \dots, a_k \rangle$  in place *p* is a substitution of arc label  $x = (x_1, \dots, x_k)$ .

To translate a place *p* to its counterpart in the target PROMELA program, the translation strategy is to declare an array data object for *p* for holding tokens. Since the instantiation of tokens when executing the PROMELA program is simply by selecting the

element directly from the array data object, the translation of the associated labeled variable is omitted to avoid redundancy. In PrT nets, the token type of a place  $p$  is defined by  $\varphi(p) = SORT$ , where  $SORT$  is valid data types defined in the semantic domain  $Spec$ . In this translation,  $SORT$  is restricted to the basic data types in the target language PROMELA [38]. Due to tractability, the maximum number of elements for each array data object has to be predefined. In addition, an index is required for each array data object to keep track of token deposits.

In PrT nets, tokens deposited in a place do not have specific orders, and their instantiations are implicit. However, for model execution, token instantiations have to be explicitly defined. For simplicity, the strategy is to let the index of the array data object always points to the tail of the array. As a result, removing or adding a token is always happening at the tail of the array data object. For example, an array  $p$  has three elements  $p[0], p[1], p[2]$ ; assuming that the index is pointing at the third element  $p[2]$ , which means the third element  $p[2]$  is null and is the first available slot to deposit a token. On the other hand, if a token is to be instantiated, then  $p[1]$  will be instantiated (removed) since it is the first available token next to the null element  $p[2]$ . For translating initial marking  $M_0(p)$ , the array data object representing  $p$  is initialized to desired values at the time of declaration.

In summary, place translation rules are defined based on the following principles:

- (1) For all  $p$  in  $P$ ,  $p$  and its associated labeled variable are translated to an array data object in the PROMELA program.
- (2) The data type for array data object  $p$  is limited to the basic data types supported in PROMELA.

- (3) For each array data object  $p$ , define the maximum number of elements for  $p$ , and define an integer variable as the index of  $p$ .
- (4) Removing or adding a token is at the tail of array data object  $p$ .
- (5) Places that hold agent tokens are not explicitly translated due to two reasons: (a) agent tokens (agent nets) have been covered in the entity translation rule; (b) the identification (process id) of an agent token is given at runtime since each agent token run as an independent process in the PROMELA program.

Place translation rules are summarized in Table 5.

Table 5. Place translation rules.

Rule	Elements associated with a $p$ in $P$	Declarative PROMELA statements
$p.1$	$p$	<b>#define</b> MAX_p = MAX; <b>place_p</b> p[MAX_p]; <b>int</b> p_idx = TAIL;
$p.2$	$\varphi(p) = sort_1 \times \dots \times sort_n$	<b>typedef</b> place_p { $sort_1$ $x_1$ ; ... ; $sort_n$ $x_n$ };
$p.3$	$M_0(p)$	p[MAX_p] = INITIAL_VALUES;

For example, let  $\varphi(p) = int \times int$ ; that is, the sort of place  $p$  is a Cartesian product of two integers. Assuming that place  $p$  holds a maximum of four tokens, the translated PROMELA statements are as follows.

```
#define MAX_p = 4
typedef place_p { int x; int y }
place_p p[MAX_p] = 0;
int p_idx = 0;
```



## B.2. Transition translation rules.

The set of transitions  $T$  specifies the events that can change the marking of a PrT net. For all  $t \in T$ , there exists a constraint formula  $R(t)$  that defines the pre-conditions for enabling  $t$  and post-conditions after firing  $t$ . The set of all possible firing sequences with respect to some initial marking defines the dynamic behavior of a PrT net. Based on the dynamic semantics of PrT nets, (1) an enabled transition may not fire immediately; (2) the firing of an enabled transition is atomic; and (3) enabled transitions can fire non-deterministically and concurrently. In this translation, however, it is assumed that an enabled transition fires immediately and the transition firings are interleaving. These assumptions do not affect the verification of state-based system properties such as safety and liveness properties, thus are adequate for this study. Transition translation rules are defined based on the following principles:

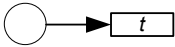
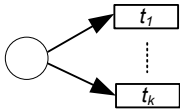
- (1) All transitions in a net are compositional in a *do...od* loop. Within the loop, transitions in *conflict* are specified by a selection construct *if ... fi*. As a result, the executability of transitions within the loop will be repeatedly checked in which the selection of transitions for execution is non-determinism based on PROMELA's semantics engine.
- (2) The constraint formula  $R(t)$  of a transition  $t$  is translated to a associated PROMELA construct in the form: *precondition-statements*  $\rightarrow$  *post-condition-statements*, where the executability of  $t$  is based on the '*precondition-statements*' and the firing of  $t$  is an atomic execution of *post-condition-statements* in PROMELA.
- (3) The universal quantifier  $\forall$  and existential quantifier  $\exists$  in  $R(t)$  are translated into *do...od* loop statements since they involve checking all or part of the elements in an array data object.

(4) The channel command specified in  $R(t)$  is translated to a rendezvous channel with buffer size 0 in PROMELA to address synchronous interactions among agents. The system channel that is shared by all agents is declared as a global message channel to be used for exchanging messages among processes. In addition, a local channel in each process is declared to receive messages from the mediator net through the global message channel.

Transition translation rules are summarized in Table 6.

A constraint formula  $R(t)$  of  $t$  is composed of  $R_u(t) \wedge R_c(t)$ , where  $R_u(t)$  is a non-communication constraint and  $R_c(t)$  is a communication constraint;  $R_c(t) = \emptyset$  if transition  $t$  is not a communication channel. A constraint formula  $R(t)$  is translated to a PROMELA construct in the form: *precondition-statements- $t$*   $\rightarrow$  *post-condition-statements- $t$* . In PROMELA, a separator ‘;’ is usually used for the separation of sequential composition of statements and declarations; it is not a statement terminator. The right arrow sign ‘ $\rightarrow$ ’ is a separator as well, and not a logical implication. For program readability, however, the arrow sign ‘ $\rightarrow$ ’ is used instead of the ‘;’ sign for separating pre-condition statements and post-condition statements. Precondition statements are guard statements, which may include relational statements (expressions with relational operators  $<, >, \leq$  and  $\geq$ ), equality statements, inequality statements, or a compound statement of the above statements connecting by *logical and* operator ‘&&’ or *logical or* operator ‘||’. For example, if the pre-conditions for transition  $t$  are defined as:  $x \neq 1 \wedge (y = 0 \vee z > 1)$ ; the translated PROMELA statement is  $x != 1 \ \&\& \ (y == 0 \ || \ z > 1)$ . The evaluation result of the guard statements is either *true* or *false*.

Table 6. Transition translation rules.

Rule <i>t.1.</i> Communication channel translation rule	
Channel	Interactions (synchronization)
<p>(1) declare global message channel:  <b>chan</b> <math>S = [0]</math> of { MSG, chan};  (2) declare local message channel in the system net:  <b>chan</b> <math>a\_id</math>;  (3) declare local message channel in an agent net:  <b>chan</b> <math>me = [0]</math> of { MSG };</p>	<pre> <b>typedef</b> MSG {     ..... }  (1) agent net initiate communication:     <b>MSG</b> msg;      <math>S!start(me);</math> /* agent send */     <math>S?msg(a\_id);</math> /* the system receive */ */ (2) system response:     <b>MSG</b> msg;      <math>a\_id!msg;</math> /* system send */     <math>me?msg;</math> /* agent receive */ </pre>
Rule <i>t.2.</i> Structural translation rule	
<p><b>Step 1:</b> for each non-conflicting transition <math>t \in T</math> such that <math>R(t) = pre \wedge post</math>, <math>R(t)</math> is translated into an atomic statement:  :: <b>atomic</b> { <math>pre\text{-}statements\text{-}t \rightarrow post\text{-}statements\text{-}t</math> }</p> 	<p><b>Step 3:</b> for <math>k</math> conflicting transitions <math>t_1 \dots t_k</math> such that <math>R(t_i) = pre_i \wedge post_i</math></p>  <p>Translated PROMELA statements:  :: <b>atomic</b> { guard-condition <math>\rightarrow</math>  <b>if</b> :: <math>pre\text{-}statements\text{-}t_1 \rightarrow post\text{-}statements\text{-}t_1</math>  :: <math>pre\text{-}statements\text{-}t_2 \rightarrow post\text{-}statements\text{-}t_2</math>  .....  :: <math>pre\text{-}statements\text{-}t_k \rightarrow post\text{-}statements\text{-}t_k</math>  <b>fi</b> }</p>
<p><b>Step 2:</b> Compose all atomic statements into a <b>do...od</b> construct:  <b>do</b>  :: <b>atomic</b> { <math>pre\text{-}statements\text{-}t_1 \rightarrow post\text{-}statements\text{-}t_1</math> }  :: <b>atomic</b> { <math>pre\text{-}statements\text{-}t_2 \rightarrow post\text{-}statements\text{-}t_2</math> }  .....  :: <b>atomic</b> { <math>pre\text{-}statements\text{-}t_n \rightarrow post\text{-}statements\text{-}t_n</math> }  <b>od</b></p>	

Rule <i>t.3</i> . Constraint formula with channel command translation rules		
<b>(i)</b> $R(t) = pre \wedge c?msg \wedge post$ translated statement: :: <b>atomic</b> { statements- <i>t</i> ; $c?msg$ → $post$ -statements- <i>t</i> }	<b>(ii)</b> $R(t) = c?msg \wedge post$ translated statement: :: <b>atomic</b> { $c?msg$ → $post$ -statements- <i>t</i> }	<b>(iii)</b> $R(t) = pre \wedge post \wedge c!msg$ translated statement: :: <b>atomic</b> { $pre$ -statements- <i>t</i> → $post$ -statements- <i>t</i> ; $c!msg$ }
Rule <i>t.4</i> . Constraint formula translation rules		
Components	PROMELA statements	
preconditions <b><i>pre</i></b> (a logical formula containing operators =, ≠, <, >, ≤, ≥, ∧, ∨)	(1) relational statements with <b>relational-operators</b> <, >, ≤ or ≥ : $op1$ <b>relational-operator</b> $op2$ ; (2) statements with <b>equality or inequality operators</b> == or !=: $op1 == op2$ ; or, $op1 != op2$ (3) logical statement with <b>logical-operators</b> && or   : $relational-statement-1$ <b>logical- operators</b> $relational-statement-2$	
universal quantifier ∀ in $R(t)$ (e.g., $\forall x \in X. (pre \wedge post)$ )	<b>int</b> p_idx = 0; <b>do</b> :: p_idx < MAX_p && ! $pre$ → break :: p_idx < MAX_p && $pre$ → p_idx++ :: p_idx ≥ MAX_p → $post$ ; break <b>od</b>	
existential quantifier ∃ in $R(t)$ (e.g., $\exists x \in X. (pre \wedge post)$ )	<b>int</b> p_idx = 0; <b>do</b> :: p_idx < MAX_p && $pre$ → $post$ ; break :: p_idx < MAX_p && ! $pre$ → p_idx++ :: p_idx ≥ MAX_p → break <b>od</b>	
post-conditions <b><i>post</i></b> (a logical formula containing ∧ or ∨)	a statement sequence including valid expressions in PROMELA	

A global message channel  $S\_CHAN$  is shared by agent processes to send messages to the system process; local message channel  $a\_id$  in the system process is used to send messages to agent processes; and, local message channel  $me$  in agent processes is used to receive message from the system process. That is, channels ' $a\_id$ ' and ' $me$ ' are matching pairs of message channels. Note that if an input channel is used as part of the pre-conditions and is not a sole pre-condition, then the input channel has to be put at the end of a set of precondition statements. The reason for this is the executability of a rendezvous message channel depends on the other matching message channel based on the semantics engine of PROMELA. It will cause an error if an input channel is put in the middle of a conjunction of guard statements. In addition, a separator ';' has to be used instead of *logical and* operator '&&' between the input channel and its previous guard statements. This precaution is to make sure that the original semantics of the net is translated correctly. For example, let  $agent_1 = (P, T, F)$  be an agent net, where  $P = \{p1, p2\}$ ;  $\varphi(p_1) = \varphi(p_2) = int$ ;  $T = \{t1, t2\}$ ;  $F = \{(p1, t1), (p1, t2), (t1, p2), (t2, p2)\}$ ;  $L(p1, t1) = L(p1, t2) = x$ ;  $L(t1, p2) = L(t2, p2) = z$ ;  $R(t1) = x > 3 \wedge me?y \wedge z = x - y$ ;  $R(t2) = x \leq 3 \wedge me?y \wedge z = x + y$ ;  $M_0(p1) = \{5\}$ ;  $M_0(p2) = \emptyset$ ; the translated PROMELA statements by applying translation rule **t.3** are as follows.

```

chan me = [0] of { int }
do
  :: atomic {  $x > 3$ ;  $me?y \rightarrow z = x - y$  }
  :: atomic {  $x \leq 3$ ;  $me?y \rightarrow z = x + y$  }
od

```

The transition constraint  $R(t1) = x > 3 \wedge me?y \wedge z = x - y$ , where  $x > 3 \wedge me?y$  is the pre-conditions and  $z = x - y$  is the post-condition. The pre-conditions are translated to PROMELA statements as “ $x > 3; me?y$ ”, where two guard statements are separated by the separator ‘;’ instead of *logical and* operator ‘&&’; that is, the pre-conditions cannot be specified as “ $x > 3 \&\& me?y$ ”, which will cause an error.

If a constraint formula contains universal quantifier  $\forall$ , then the whole set of tokens need to be examined. For example,  $\forall x \in X. (pre \rightarrow post)$ , where every element of  $X$  has to be checked for pre-condition  $pre$ . The strategy is to use an index for the array object that contains the tokens, and examine each element for the pre-condition. As soon as an element is found to be false, then the formula is immediately evaluated as a *false*. On the other hand, if a constraint formula contains quantifier  $\exists$ , then only one element in the array object is needed to satisfy the pre-condition. A constraint formula containing quantifiers is translated to PROMELA statements by applying rule **t.4** in Table 6.

Let us look at a translation example for a transition constraint formula without channel command. Let a constraint formula  $R(t)$  be defined as  $pre \rightarrow post$  where  $pre$  is  $\exists p \in P. (p[1] = ra[4]) \wedge \forall r \in R. (r[1] \neq ra[3])$ , and  $post$  is  $R' = R \cup \langle ra[3], ra[4] \rangle \wedge ra'[4] = 'added'$ ; the translated PROMELA statements are shown in Figure 30.

### 5.2.2 Model Transformation

Let  $MAS$  be the set of PrT nets called multi-agent nets specifying a multi-agent system, and  $PROG$  be the target PROMELA program. The transformation steps can be defined as follows:

Step 1. Apply entity translation rule *e.1* to *MAS* such that, for all  $N \in MAS$ , add a process

proctype  $N()$  to the target PROMELA program *PROG*.

Step 2. For each process  $N()$  in *PROG*, add PROMELA statements by applying rule *p.1*~

*p.3* to all  $p \in P$  associated with net entity  $N$ .

Step 3. For each process  $N()$  in *PROG*, add PROMELA statements by applying rule *t.1*~

*t.4* to all  $t \in T$  associated with net entity  $N$ .

```

cnt = 0; p12_idx--;
do :: cnt < MAX_SESSION && pa[cnt].roleP ==
    p12[p12_idx].content → cnt = 0; break
  :: cnt < MAX_SESSION && pa[cnt].roleP !=
    p12[p12_idx].content → cnt++
  :: cnt >= MAX_SESSION →
    printf("Role %e denied !", p12[p12_idx].content);
    p12[p12_idx].content = error
od;
do :: cnt < MAX_ROLE && ra[cnt].user == p12[p12_idx].category
    → ra[ra_idx].roleR = p12[p12_idx].content;
    p12[p12_idx].content = updated; break
  :: cnt < MAX_ROLE && ra[cnt].user != p12[p12_idx].category → cnt++
  :: cnt >= MAX_ROLE && ra_idx < MAX_ROLE
    → ra[ra_idx].user = p12[p12_idx].category;
    ra[ra_idx].roleR = p12[p12_idx].content;
    p12[p12_idx].content = added; ra_idx++; break
  :: ra_idx >= MAX_ROLE
    → printf("Exceed maximum role");
    p12[p12_idx].content = error; break
od;

```

Figure 30. A constraint formula translation example.

### 5.3 Correctness of the Translation

The translation is under the assumption that every PrT net in the formal MAS model has finite tokens. In addition, an interleaving semantics is adopted in translating transition

firings. The correctness of the translation is justified based on translation *completeness* and behavior *consistency* between two models.

### 5.3.1 Completeness

*Definition 5.3.1* Given a multi-agent net  $MAS$  and a translated PROMELA program  $PROG$ , the translation from  $MAS$  to  $PROG$  is *complete* if all net entities and net elements in  $MAS$  are covered by associated language constructs in  $PROG$ .

#### (1) Completeness of net entity translation

**Lemma 1.** Given a multi-agent net  $MAS$  and a translated PROMELA program  $PROG$ , a net entity translation is complete if,  $\forall N \in MAS$  such that, there exists a process *proctype*  $N()$  in  $PROG$ .

**Proof.** Based on net entity translation rule *e.1*, a skeleton process is faithfully created for each net entity in  $MAS$ . Therefore,  $PROG$  covers all entities in  $MAS$ .

#### (2) Completeness of net element translation

**Lemma 2.** Given a net entity  $N$  in  $MAS$  and its associated process  $N()$  in  $PROG$ , where  $N = (P, T, F)$ , a net element translation of  $N$  is complete if, net elements  $P, T, F$  and their static semantics  $\varphi, L, R$ , and  $M_0$  are properly mapped to associated PROMELA statements in process  $N()$ .

**Proof.** Rule *p.1~p.3* and *t.1~t.4* cover all net element translations of net  $N$  given that (i) for all places  $p$  in  $P$  such that,  $\varphi(p)$ ,  $L(p, x)$  and  $M_0(p)$  are translated by applying rule *p.1~p.3*; and, (ii) for all  $t$  in  $T$  such that,  $R(t)$  is translated by applying rule *t.1~t.4*. Thus, the process  $N()$  covers all syntactic and static semantics definitions of a net entity  $N$ .



### (3) Completeness of multi-agent net translation

**Lemma 3.** Given a multi-agent net  $MAS$  and a translated PROMELA program  $PROG$ , a multi-agent net translation is complete if,  $\forall N \in MAS$ , there exists a process *proctype*  $N()$  in  $PROG$  such that,  $N()$  covers the syntactic and static semantics of  $N$ .

**Proof.** Given Lemma 2, the net element translation rules can be applied to all  $N$  in  $MAS$ . Thus,  $PROG$  is a complete translation of multi-agent net  $MAS$ .

#### 5.3.2 Consistency

The dynamic behavior of a PrT net depends on the initial marking, instantiation of tokens and the definition of transition constraints. That is, the variation of the initial marking and instantiation of data tokens result in different execution sequences at runtime based on the same transition constraints. Therefore, the behavior consistency of two models in the translation is justified based on the same initial marking.

*Definition 5.3.2* Given a multi-agent net  $MAS$  and a translated PROMELA program  $PROG$ ,  $MAS$  and  $PROG$  are *behavior consistent* if,

- (i) For every net entity  $N \in MAS$ , there is a process  $N()$  in  $PROG$  preserves the dynamic semantics of  $N$ .
- (ii) For every transition  $t$  in  $T$  of the net entity  $N \in MAS$ , there is an atomic language construct  $E$  in the process  $N()$  in  $PROG$  such that,  $E$  preserves the semantics of  $R(t)$ .

**Lemma 4.** Given a transition  $t$  with  $R(t) = pre_t \wedge post_t$  in  $N$ , and its associated language construct  $E = pre\_statements-t \rightarrow post\_statements-t$  in process  $N()$ ;  $E$  is semantically consistent with  $R(t)$  if, for a firing  $M[t/\alpha > M'$  in  $N$  there exists an execution  $s E s'$  in  $N()$  such that  $M \Leftrightarrow s$  and  $M' \Leftrightarrow s'$ .

**Proof.** If  $E$  is executable under state  $s$  in the process  $N()$ , then *pre-statements- $t$*  must be true; the execution of  $E$  under state  $s$  results in state  $s'$ . By rule **t.2~t.4**, (i) *pre-statements- $t$*  is a mapping of  $pre_t$  in  $R(t)$  implies  $M \Leftrightarrow s$ ; and, (ii) *post-statements- $t$*  is a mapping of  $post_t$  in  $R(t)$  implies  $M' \Leftrightarrow s'$ .

**Lemma 5.** Given a net entity  $N \in MAS$ , and a complete translation process  $N()$  in *PROG* from  $N$ ,  $N()$  is semantically consistent with  $N$  if, for any firing sequence  $\sigma$  in  $N$  starting from the initial marking  $M_0$ , there exists an execution sequence  $e$  starting from the initial state  $s_0$  in  $N()$ , such that  $\sigma \Leftrightarrow e$  when  $M_0 = s_0$ .

**Proof.**

- (1) By Lemma 4, a firing  $\sigma = M_0 t_0 M_1$  of transition  $t_0$  under the initial marking  $M_0$  in  $N$  implies that there is an execution  $e = s_0 E_0 s_1$  under the initial state  $s_0$  in  $N()$  such that,  $M_0 \Leftrightarrow s_0$  and  $M_1 \Leftrightarrow s_1$ ; thus,  $e$  is semantically consistent with  $\sigma$ .
- (2) By (1),  $e$  is semantically consistent with  $\sigma$  for the case  $\sigma = M_1 t_1 M_2$  and  $e = s_1 E_1 s_2$  such that,  $M_1 \Leftrightarrow s_1$  and  $M_2 \Leftrightarrow s_2$ . It immediately follows that  $e$  is consistent with  $\sigma$  for the case  $\sigma = M_0 t_0 M_1 t_1 M_2$  and  $e = s_0 E_0 s_1 E_1 s_2$ , since  $M_1, s_1$  are the derivatives of  $M_0, s_0$  and  $M_2, s_2$  are the derivatives of  $M_1, s_1$  respectively.
- (3) By (1) and (2), for an execution  $\sigma = M_0 t_0 M_1 t_1 M_2 \dots M_{k-1} t_{k-1} M_k$  starting from the initial marking  $M_0$  in  $N$ , there is an execution  $e = s_0 E_0 s_1 E_1 s_2 \dots s_{k-1} E_{k-1} s_k$  starting from the initial state  $s_0$  in  $N()$  such that,  $\sigma$  and  $e$  are consistent given that  $M_0 \Leftrightarrow s_0, M_1 \Leftrightarrow s_1, \dots$ , and  $M_k \Leftrightarrow s_k$ .

(4) Based on the same initial state, other executions are proved by (1), (2) and (3). Thus, given a net entity  $N$  and a complete translation process  $N()$  from  $N$ ,  $N()$  preserves the semantics of  $N$ .

By Lemma 1 through 5, the correctness of this translation approach transforming the formal MAS model to a concrete model in PROMELA is proved.

#### 5.4 A Translation Example

Figure 31 shows a formal MAS model specifying a disaster mitigation system called BCIN (Business Continuity Information Network) [89]. Based on our previous study [90], BCIN employs a role-based access control for system resources. Each role has a distinct behavior model. The BCIN net shown in Figure 31 is the formal model to visualize the control of resources in BCIN. The semantic definitions for BCIN net are shown in Appendix A4. BCIN net is transformed to a PROMELA program by applying the translation rules defined in the previous sections. The resulting BCIN net consisting of one system net and four agent nets, has 28 transitions (excluding boundary transitions), 31 places and 16 communication channels.

The translated PROMELA model contains 237 lines of codes, including five processes: (1) the system net; (2) supervisor role (S\_NET); (3) observer role (O\_NET); (4) primary contact role (P\_NET); and, (5) participant role (PR\_NET). The PROMELA model for the BCIN net is shown in Appendix B.

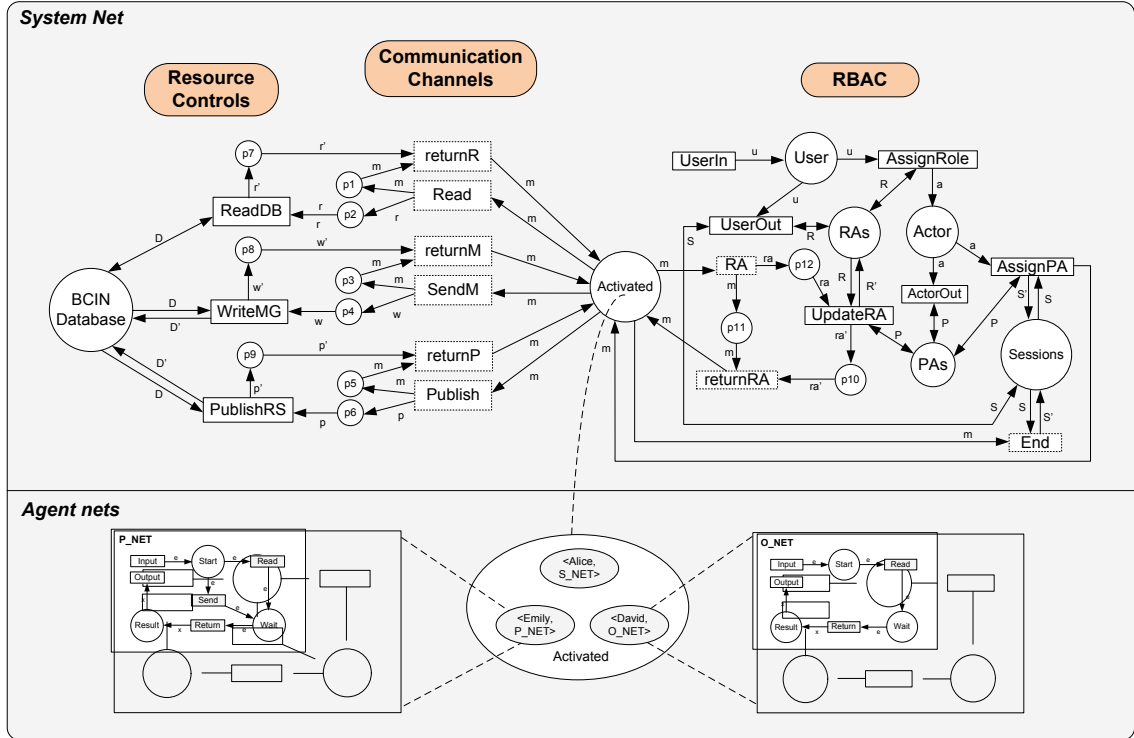


Figure 31. The BCIN net.

## 5.5 Model Analysis

Two types of correctness claims are considered in model checking the formal MAS model: (1) safety properties; and (2) liveness properties. A safety property defines an invariant that the model must always satisfy. A liveness property states that something good does happen on executing the model. Given a transformed PROMELA model, and correctness claims given in LTL, the model checker SPIN either proves that such claims are impossible or it provides examples of behaviors that match the claims. There are several ways to encode correctness claims in PROMELA. This section discusses how a transformed formal MAS model can be analyzed by the model checker SPIN.

### 5.5.1 Encoding System Properties in PROMELA

In SPIN, correctness claims are used to formalize erroneous system behaviors. The goal is to decide whether design requirements could possibly be violated [38]. There are six basic constructs in PROMELA to specify correctness properties:

- (1) *Basic assertions*; an assertion statement is a correctness claim, which asserts that the expression used in the statement cannot be false.
- (2) *End-state labels*; a meta-label indicating a valid end-state other than the default end-state of a process. A default end-state is at the end of the codes of a process.
- (3) *Progress-state labels*; a meta-label marking the execution of some statement; they can be used to detect a non-progress cycle.
- (4) *Accept-state labels*; they are used to find all cycles that do pass through at least one of those labels; and, there should not exist any execution that can pass through an accept-state label infinitely often.
- (5) *Never claims*; they are used to specify either finite or infinite system behavior that should never occur. A never-claim is checked at each execution step of the model execution.
- (6) *Trace assertions*; they are used to formalize statements about valid or invalid sequences of operations that processes can perform on message channels [38]. All channel names referenced in a trace assertion must be globally declared, and all message fields must be globally known constants or *mtype* symbolic constants. Traces on monitoring rendezvous channels can only capture the occurrence of the receive part of a handshake, not of the send part. Receive events on rendezvous channels can be monitored with trace assertions, but not with never claims.

Let us look at some examples using the above constructs to encode the correctness claims of the BCIN net [90]. Since BCIN net models the resource control among agent nets, there are several important properties need to be ensured:

- (a) A user cannot access the system without a valid role assignment.
- (b) A user enacting a role can only perform the actions pre-assigned based on permission assignments.
- (c) A user can only play a role at one time.
- (d) For each request received by the system net, there will be a response to the agent net eventually.

Property (a), (b) and (c) are considered as safety properties, which are required to be true at all times, and property (d) is considered as a liveness property. Safety properties can be encoded using basic assertions or never claims, while liveness properties can be encoded using meta-labels and trace assertions. These properties are translated into PROMELA codes as follows.

#### Property (a)

Although the constraint formula:  $\forall r \in R. (r[1] \neq u)$  defined in transition ‘*UserOut*’ (Appendix A4) enforces the elimination of invalid user tokens, a system invariant can be defined to check whether or not the BCIN model satisfies property (a). For example, the constraint formula states that if the user has not yet been assigned a role (a member of the set of role assignment  $R$ ), then the user cannot enter the system. Intuitively, all users in the system must be members of the set  $R$ . Therefore, the correctness claim can be defined as follows.

- `assert ( valid_user == ra[0].user || valid_user == ra[1].user || valid_user == ra[2].user )`

The assertion states that a valid user must be one of the users that defined in the array data object ‘*ra*’ that stores valid role assignments. All possible user identities need to be enumerated in order to check if this correctness claim is satisfied. An assertion statement is the only type of correctness property that can be checked in the simulation mode in SPIN. The execution will stop at the point where the assertion fails.

Another way to encode the correctness claim for this property is using a *never claim*. The property requires that a user cannot access the system without a valid role assignment; that is, at all time, users who are in sessions should be valid users. In the PROMELA model for BCIN net (Appendix B), the array data object ‘*session[act\_idx].sname*’ stores the user names in sessions. Therefore, when ‘*session[act\_idx].sname*’ is not null, it has to be one of the member of role assignments stored in the array data object ‘*ra*’. Let proposition *p* denotes users in sessions, and proposition *q* denotes valid users. The property can be encoded into a LTL formula  $\Box (p \rightarrow q)$ , where

- *p* is defined as: *session[act\_idx] != NULL*, and
- *q* is defined as:  

$$(session[act\_idx].sname == ra[0].user \parallel session[act\_idx].sname == ra[1].user \parallel session[act\_idx].sname == ra[2].user \parallel session[act\_idx].sname == ra[3].user)$$

SPIN provides a built-in translator that translates a LTL formula to never claim statements. A property can be encoded in a LTL formula first, and then translated into PROMELA codes using the translator provided. The PROMELA statements generated

from the above formula is shown in Figure 32, which results in the verification report in Figure 33.

#### Property (b)

In BCIN net, an actor is a valid user enacting a predefined agent role. An agent role is a role model with predefined permission assignments for accessing system resources. The constraint formula:  $\exists p \in P. (p[1] = a[2])$  defined in transition *AssignPA* (Appendix A4) makes sure that an associated role model is available to be activated. To assure this property, a correctness claim can be defined as follows.

- assert (actor.roleR == pa[0].roleP || actor.roleR == pa[1].roleP || actor.roleR == pa[2].roleP || actor.roleR == pa[3].roleP)

The assertion states that an agent role that an actor is enacting must be one of the predefined role models.

#### Property (c)

A session constraint is enforced by the formula  $\exists s \in S. (s = u)$  in the constraint definition of transition *UserOut*. In the PROMELA model, an assertion statement “assert (inSession == false)” asserts that a user can only have one session at a time.

#### Property (d)

The message channels in agent processes are declared as local message channels in the BCIN model. Since a trace assertion can only monitors global channels, this property can be checked simply using a progress label at a receiving message channel to make sure the agent receives the information from the server. For example, “*progress: me?m*” states that



it is impossible for the system to execute forever without also passing through the labeled states. In other words, the receive-message channel in an agent process should be visited infinitely often.

```
#define p (session[act_idx].sname != NULL)
#define q (session[act_idx].sname == ra[0].user || session[act_idx].sname ==
ra[1].user || session[act_idx].sname == ra[2].user || session[act_idx].sname ==
ra[3].user )

/*
 * Formula As Typed: [] (p -> q)
 * The Never Claim Below Corresponds
 * To The Negated Formula !([] (p -> q))
 * (formalizing violations of the original)
 */

never { /* !([] (p -> q)) */
T0_init:
    if
    :: (!((q)) && (p)) -> goto accept_all
    :: (1) -> goto T0_init
    fi;
accept_all:
    skip
}
```

Figure 32. The PROMELA codes for a never claim.

(Spin Version 5.2.4 -- 2 December 2009)

+ Partial Order Reduction

Full statespace search for:

never claim	+
assertion violations	+ (if within scope of claim)
acceptance cycles	+ (fairness disabled)
invalid end states	- (disabled by never claim)

State-vector 234 byte, depth reached 174, errors: 0

148 states, stored

33 states, matched

181 transitions (= stored+matched)

452 atomic steps

hash conflicts: 0 (resolved)

Stats on memory usage (in Megabytes):

0.035 equivalent memory usage for states (stored\*(State-vector + overhead))

0.262 actual memory usage for states (unsuccessful compression: 743.19%)

state-vector as stored = 1842 byte + 16 byte overhead

2.000 memory used for hash table (-w19)

0.305 memory used for DFS stack (-m10000)

2.501 total actual memory usage

pan: elapsed time 0.002 seconds

#endif

Figure 33. The verification report for the never claim.

In summary, safety properties can be checked through never claims and basic assertions, while liveness properties can be checked through never claims, progress-state labels and trace assertions. One important liveness property in formal MAS models is that an agent's request is eventually responded. This can be checked by using a progress-state label at the receive-message channel statement of an agent net. SPIN provides a translator for translating valid LTL formulas into never claims. Note that, liveness properties deal with infinite runs. An infinite run only happens in a finite system if the run is cyclic [38]. SPIN checks liveness properties by identifying acceptance cycles [38]. Table 7 summarizes the encoding rules.

Table 7. Encoding system properties

System Properties	PROMELA statement
<i>System invariant:</i> $\Box p$	<b>#define</b> p ..... <b>assert</b> (p)
<i>Properties expressed in LTL:</i> (1) $\Box (p \rightarrow q)$ (2) $\Box (p \rightarrow \Diamond q)$ (3) $\Diamond \Box p$ (4) $\Box (p \rightarrow \Diamond q)$ (5) Other valid LTL formula	<b>#define</b> p ..... <b>#define</b> q ..... <b>never</b> { ..... } 
<i>Liveness property:</i> an agent net eventually received the response	..... <b>progress:</b> me?msg; .....

### 5.5.2 Model Checking the MAS Model

Table 8 shows the transformation statistics from the BCIN net to the PROMELA model.

After generating the PROMELA model and desired correctness claims, the model is ready to be checked in SPIN. There are verification mode and simulation mode in SPIN. Usually, the model is checked in the verification mode first. If errors are found, execution trails can be examined step by step in the simulation mode.

Table 8. Statistics of model transformation.

The BCIN Net	The PROMELA Model
<ul style="list-style-type: none"> <li>▪ 5 individual nets (1 system net and 4 agent nets)</li> <li>▪ 31 places</li> <li>▪ 28 transitions (including 16 communication channels, excluding boundary transitions)</li> </ul>	<ul style="list-style-type: none"> <li>▪ 5 processes</li> <li>▪ 237 lines of codes</li> <li>▪ 328 execution trails generated (executing with some initial marking)</li> <li>▪ 2.501 Mbytes of memory used (500 states per second)</li> </ul>

The BCIN model is used as an example running under XSpin version 5.2.3 based on the following initial marking (initial state):

$M_0(\text{User}) = \{ \langle \text{Emily} \rangle \};$

$M_0(\text{RAs}) = \{ \langle \text{David, observer} \rangle, \langle \text{Alice, supervisor} \rangle, \langle \text{Emily, contact} \rangle \};$

$M_0(\text{PAs}) = \{ \langle \text{observer, O\_NET} \rangle, \langle \text{supervisor, S\_NET} \rangle, \langle \text{contact, P\_NET} \rangle, \langle \text{participant, PR\_NET} \rangle \};$

$M_0(\text{BCIN\_Database}) = \{ \langle \text{advisory, Wilma} \rangle, \langle \text{message, generator} \rangle \}.$

While the BCIN model is checked under the verification mode, agent nets are activated at different point in the PROMELA model to test the correctness of interactions between processes at runtime. In addition, unknown agent identities are also intentionally introduced at some points of the execution to test the exception handling of the PROMELA model. System properties (a), (b), (c) and (d) defined in the previous section are added in the PROMELA model and checked in both verification mode and simulation mode. While execution in verification mode, there was an error found as shown in Figure 34. By examining the simulation trails and sequence chart, the errors from the verification report and some subtle errors in the PROMELA model can be found and fixed. The interaction of agent processes in the corrected BCIN model is shown in Figure 35.

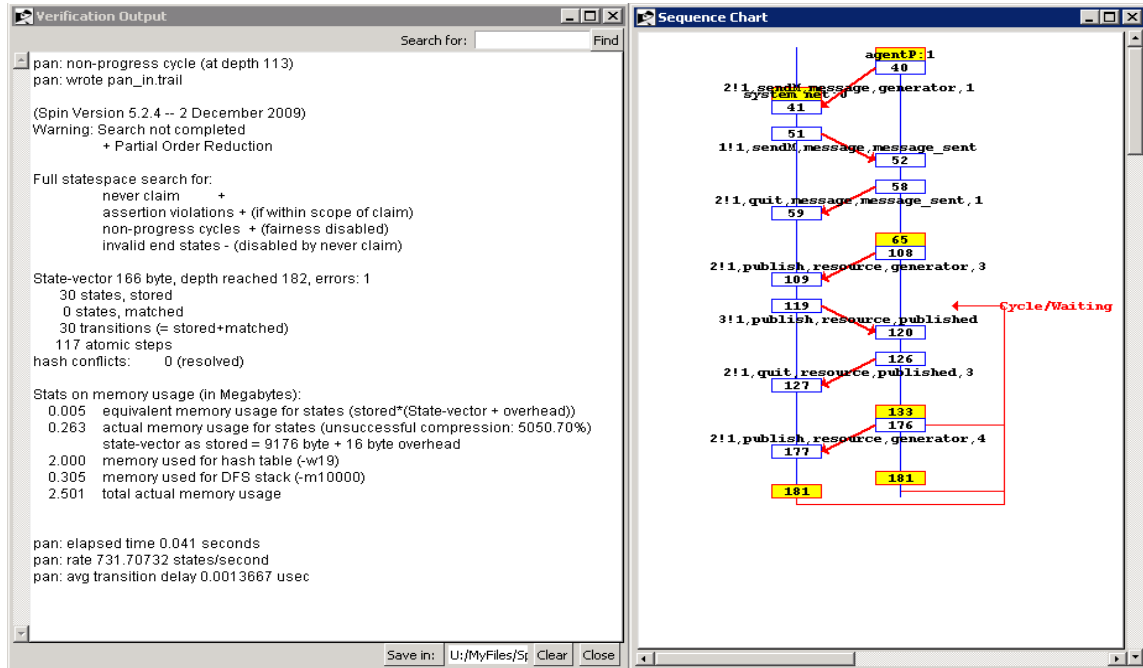


Figure 34. The verification output with non-progress cycle detected.

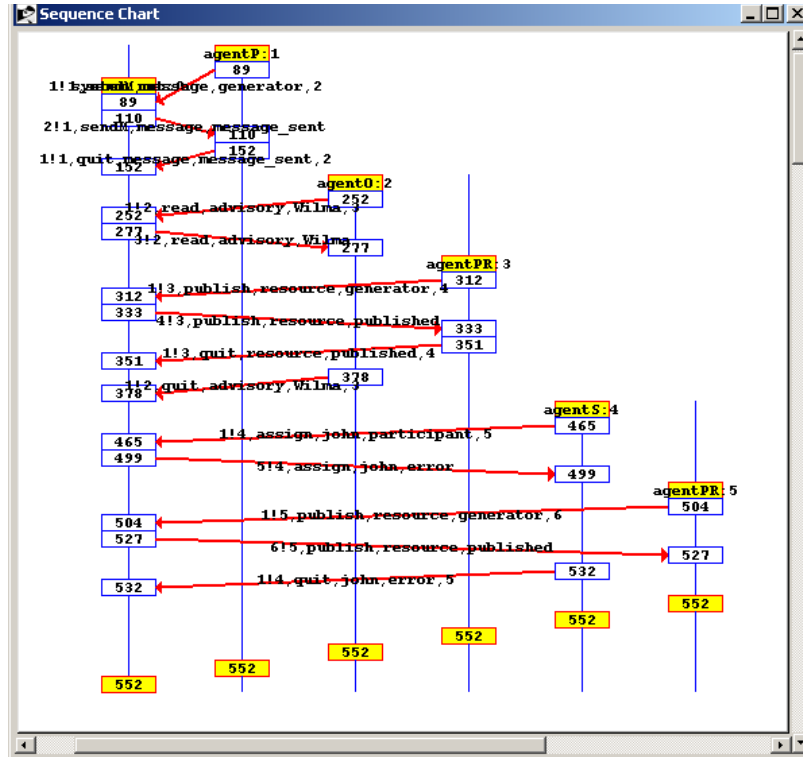


Figure 35. The sequence chart of agent interactions in a simulation run.

## 5.6 Summary

In this chapter, a methodology for analyzing formal MAS models is developed and presented. The objective is to provide a method to study the behavior of a given MAS model. The methodology is summarized as follows.

- (1) Translation rules are defined for model transformation from the formal MAS model to the PROMELA model, including the net entity translation rule that transforms a PrT net into a process skeleton in PROMELA, and net element translation rules that transform places and transitions into their associated PROMELA language constructs.
- (2) Given a nested PrT net model specifying MAS, an equivalent PROMELA model can be generated using the translation rules defined.

- (3) Given the transformed PROMELA model, the correctness claims can be encoded for model checking, including safety properties and liveness properties.
- (4) The PROMELA model can be executed in the verification mode for checking desired properties. If there are errors, the model can be diagnosed under the simulation mode by examining the execution trails generated during verification.

Based on this study, the transformed PROMELA model is fairly efficient in model analysis. However, there are two issues need to be considered for further automation of code generation from a formal MAS model to a PROMELA program. First, due to the complexity of transition constraints, manual modifications for transition translation may be needed after applying the translation rules. Second, since tokens at a place are structured data represented by an array data object, specifiers need to enumerate all possible substitutions to construct correctness claims.

The methodology presented in this chapter provides a foundation for tool development. The future work is to automate model analysis based on the transformation technique developed. The idea is to use nested Petri nets as the modeling tool for behavior models at the front-end, and the model checker SPIN as the analysis tool at the back-end.

## **CHAPTER 6**

### **CASE STUDY**

This chapter discusses case studies in three different application domains. Section 6.1 presents the case study of a wireless sensor network with mobile devices. Section 6.2 introduces the case study of an e-market. Section 6.3 elaborates the modeling of a business continuity information network for disaster mitigation.

#### **6.1 Wireless Sensor Network with Mobile Devices**

Recent wireless sensor network systems (WSN) integrate mobile devices to take advantage of the storage, communication ability and computation power of the mobile devices for gathering and sharing sensor data [34]. Since mobile devices constantly change their locations, the idea of sharing information through WSN with mobile devices is in a publish/subscribe pattern where the subscribers (some mobile devices) express the interest in some class of information and the publisher (some mobile device) picks up the information from a sensor at its current location and broadcasts the information to the subscribers. An information server is employed to manage group information of the mobile devices.

A key idea of the coordination is that mobile devices share common interest (sensor) data within the same group or among different groups. The coordination mechanism is a form of publish and subscribe and provides a paradigm for organizing multiple mobile devices through an interest-management server, in which groups of mobile devices are formed based on the location of their interested information. Mobile devices that issue queries for sensor data are subscribers. The server manages subscribers' information but



does not directly publish sensor data; that task is actually done by mobile devices using a multicast mechanism that sends the shared data to a group address given by the server. The sensor nodes in WSN are divided into clusters according to their geographic locations and the information of clusters is kept in the server. Each cluster, identified by a unique cluster id, has a cluster head and the sensor data within a cluster is periodically sent to the cluster head, which serves as a communication point with entities outside the cluster. The entities involved in communication behaviors have been identified and shown in Figure 36. The arrows denote the message flows.

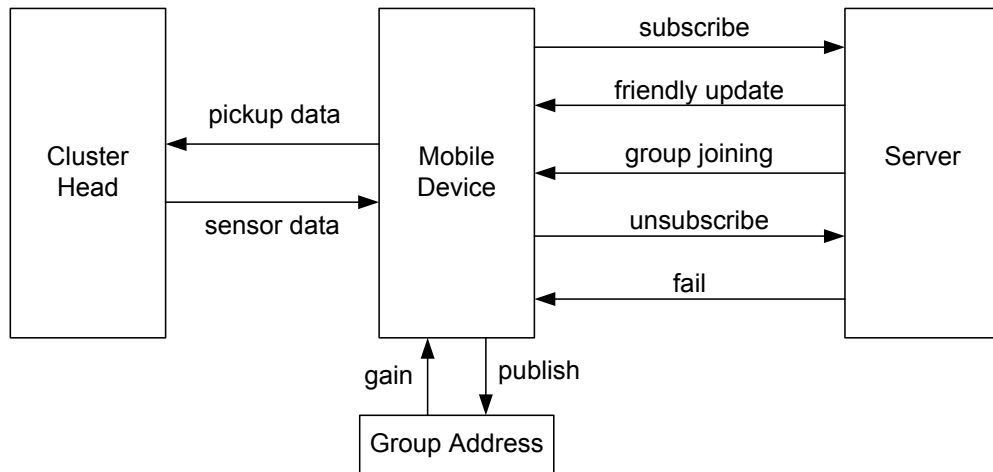


Figure 36. Message flows between the entities in a WSN with mobile devices.

The WSN with mobile devices [91] is used as an example and modeled using the two-level PrT nets defined in Chapter 3. Based on the communication relations in Figure 36, the mobile devices are modeled as agent tokens of a server net. A mobile device can issue a query (subscribe), pick up sensor data, publish shared data, gain shared data and

unsubscribe. The behavior models of a mobile device and the server net are discussed in the following sections.

#### 6.1.1 The Interest-Management Server Model

When the server receives a query request from a mobile device, there are some interactions between the server and the mobile device. The interaction scenarios are described as follows.

(1) The server checks whether or not there exists some common interest group with an interest at current location of the mobile device. If there is an existing common interest group at current location of the requesting mobile device, a message containing the group address is sent back to the mobile device. The message includes the instruction for 'friendly update', which is an action for the mobile device to pick up the sensor data from the relevant cluster head. After picked up the sensor data, the mobile device multicasts the data to the group address indicating in the message from the server.

(2) Followed by the friendly update, the server adds the requesting mobile device to the identified common interest group(s) that have the same query region as the subscribing mobile device. In the case that no existing cluster region covers the query region, a failed message is sent to the mobile device. The mobile device can unsubscribe after getting the result. When the server receives an unsubscribe message, it removes the information of the unsubscribing mobile device. Table 9 summarizes the actions of the server.

The following conventions have been used: (1) the model does not restrict the multiple subscriptions of the same group for the same mobile device; (2) it is assumed that the information required for multicasting is kept in the server, and the information is accessible during a multicast; (3) there is no action for a mobile device to temporarily

join a group to perform a friendly update; (4) the original query content is used as the unsubscribing message and assume that the group address described in the requirement is a unique and fixed value, such as a cluster id. Figure 37 shows the server's behavior model according to the information and control flows in Table 9.

Table 9. The server's actions.

<b>Actions</b>	<b>Pre-conditions</b>	<b>Post-conditions</b>
Subscribe	Mobile device sends a query	Query received
Instruct friendly update	Mobile device's current location exist common interest sharing group.	Sends a message to mobile device to instruct a friendly update at its current location.
Group Joining	Mobile device's query region is covered at least by one cluster	Adds the mobile device to the common interest sharing group (s) with the same query region and sends a joining message to the mobile device.
Fail	None of the cluster regions covered the query region.	Sends a fail message to the mobile device.
Unsubscribe	Received the unsubscribe message from a mobile device	Removes the information of the unsubscribing mobile device



*Transition Constraints:*

$$R(\text{subscribe}) = n?q \wedge m = md$$

$$R(f\_update) = \exists c \in C. (q[3] \in c[2]) \wedge c[4] \neq \emptyset \wedge q[2]!(c[2], c[3], q)$$

$$R(join) = \exists c \in C. (c[2] \cap q[4] \neq \emptyset) \wedge \forall c \in C. (c[2] \cap q[4] \neq \emptyset \wedge c'[4] = c[4] \cup q[2]) \wedge q[2]!(c[2], c[3], q)$$

$$R(fail) = \forall c \in C. (c[2] \cap q[4] = \emptyset) \wedge q[2]!'subscription\_failed'$$

$$R(unsubscribe) = n?q \wedge \forall c \in C. (c[2] \cap q[4] \neq \emptyset \wedge c'[4] = c[4] - q[2])$$

Note: transition *subscribe* is fireable when a matched output channel  $n$  in other net is ready. Transition *f\_update* is enabled if mobile device  $q[2]$ 's (an id) current location  $q[3]$  is in cluster region  $c[2]$  and a non-empty set  $c[4]$  of mobile devices subscribed to cluster region  $c[2]$ . After firing, the friendly update message is sent through the channel. Transition *join* add mobile device id  $q[2]$  to the set of subscribed mobile devices  $c[4]$ . If query region  $q[4]$  is not covered by any cluster region  $c[2]$  in  $C$ , transition *fail* will fire and a failing message is sent through channel  $q[2]$ . Transition *unsubscribe* removes the active query token and its subscribed information.

### 6.1.2 The Mobile Device Model

A mobile device communicates with the user, server, other mobile devices and cluster heads. First, a user may generate a query through the interface on the mobile device when he/she needs information. Then, the mobile device sends a query request message to the server for subscription. Various actions may be taken according to the responding message from the server. If a friendly-update instruction message is received, the mobile device has to pick up the sensor data from the cluster head at current location, and publishes the data to a group address indicating in the friendly-update message. Upon receiving the group-joining instruction message and by the expiration of Time-To-Live time frame, the mobile device may receive the data through other mobile devices.

Table 10. A mobile device's actions.

<b>Actions</b>	<b>Pre-conditions</b>	<b>Post-conditions</b>
input query	The user of the mobile device needs information	A query is generated containing query id, mobile id, current location, query region and Time-To-Live(valid time)
Subscribe	A query message is ready to send to the server.	Waits for the reply from the server.
friendly update	The server instructs a friendly update	Get the friendly update message and ready to pickup data from the cluster head in current region
join	Server has sent a group-joining message	(1) received sharing data from group members, or (2) picks up the sensor data directly by itself (3) performs direct injection to acquire the data and publishes
Fail	Server has sent a failed message	Outputs the error message to the user
Gain	Cluster data is published by group members	Got the data and ready to unsubscribe
Publish	Sensor data has been picked up.	Published the data and ready to unsubscribe
Direct pickup	Current location is inside the query region	Interact directly with cluster head to acquire sensor data
Direct injection	Current location is not in the query region and the valid time for the query is about to expire and no sharing data is available	Request sensor data from query region cluster head remotely through sensor network routing
Unsubscribe	Received the data and sends an unsubscribe message to the server	Outputs the data to the user

In other case, if the mobile device traveled to the same region with its query region before receiving the interested data through group sharing performed by the other mobile devices, it can pick up the sensor data itself and share the data with the members in the same group. If no such group sharing event happens when the time of a valid query is going to expire, i.e., no one in the same group ever reached the query region or no friendly update is performed, the mobile device will perform the ‘direct injection’, which is a way of getting the sensor data from the query region through sensor network routing. After received and published the data, the mobile device can unsubscribe from the server. The data received by the mobile device is properly extracted and displayed on the interface of the mobile device for the user. Table 10 shows the actions of a mobile device.

Based on the actions listed in Table 10, the behavior model of a mobile device is developed and shown in Figure 38. Figure 39(a) shows the group address multicast behavior model, and 39(b) shows the cluster head’s behavior model. It is assumed that a mobile device knows its location through the GPS (Global Positioning System) or other localization techniques. Place  $p8$  in Figure 38 keeps the mobile device’s id and its current location. Place  $p2$  in Figure 39(a) refers to the same database as place  $p3$  in Figure 38, which is assumed to be accessible during a multicast. In the model, the content of the responding message from the server contains the query information.

### **Semantic Definitions of Mobile Device’s PrT Model**

*Token Types:*

$QUERY = QueryID \times MobileID \times CurrentLocation \times Query\_region \times TimeToLive$

$\phi(p1) = QueryID \times Query\_region \times TimeToLive$

$\phi(p2) = \wp(QUERY)$

$\phi(p3) = \wp(Cluster\_region \times GroupAddress \times QUERY)$

$\phi(p4) = \wp(GroupAddress \times QUERY)$

$\phi(p5) = \wp(DATA \times GroupAddress)$

$\phi(p6) = \wp(QUERY \times DATA)$

$\phi(p7) = \wp(DATA)$

$\phi(p8) = MobileID \times CurrentLocation$

Note:  $QUERY$ ,  $Cluster\_region$  and  $Query\_region$  are of the same type as defined in the server's behavior model.  $QueryID$ ,  $MobileID$ ,  $CurrentLocation$ ,  $TimeToLive$ ,  $GroupAddress$  and  $DATA$  are predefined data types.

*Transition Constraints:*

$R(subscribe) = q[1] = r[1] \wedge q[2] = cl[1] \wedge q[3] = cl[2] \wedge q[4] = r[2] \wedge q[5] = r[3] \wedge S!q$

$R(f\_update) = S?(cr, ga, q)$

$R(fail) = S?d$

$R(join) = S?(cr, ga, q)$

$R(gain) = ga?d$

$R(publish) = s[2] = ga \wedge ga!s$

$R(direct\_pickup) = cl[2] \in cr \wedge cr!(q[2], ga)$

$R(direct\_injection) = cl[2] \notin cr \wedge \tau = q[5] - \varepsilon \wedge cr!(q[2], ga)$

$R(unsubscribe) = S!q$

$R(sensor\_data\_in) = cr?d$

Note:  $\varepsilon$  is a predefined and small constant value, which is determined by the designer to enforce the firing of the transition. Transition *input query* and *output data* are boundary transitions for user interface. Transition *positioning* is the boundary transition that has the function of obtaining the current location (coordinates) of the mobile device. Transition *subscribe* is enabled whenever there is token  $r$  available from the input place  $p1$  and a matched input channel  $S$  in the server net is ready. Transition *f\_update*, *join* and *fail* are



unconditionally ready for firing whenever output channel  $S$  in the server is ready to output messages. Transition  $gain$  is unconditionally ready for fire whenever output channel  $ga$  from group address multicast is ready to broadcast messages. Transition  $publish$  is ready for firing when sensor data  $s$  is received through transition  $sensor\ data\ in$  through input channel  $cr$ . Transition  $direct\ pickup$  is enabled if current location  $cl[2]$  is in query region  $cr$ . After firing, a message is sent through output channel  $cr$  to the cluster head to request data. Transition  $direct\ injection$  is enabled if current location  $cl[2]$  is not in the query region and the *time to live*  $q[5]$  is going to expire within  $\varepsilon$ . Transition  $unsubscribe$  is enabled if query  $q$  is finished and data  $d$  is available in  $p6$ . After firing, data  $d$  is output to  $p7$  and query  $q$  is sent through output channel  $S$  to the server to be removed from subscribed information.

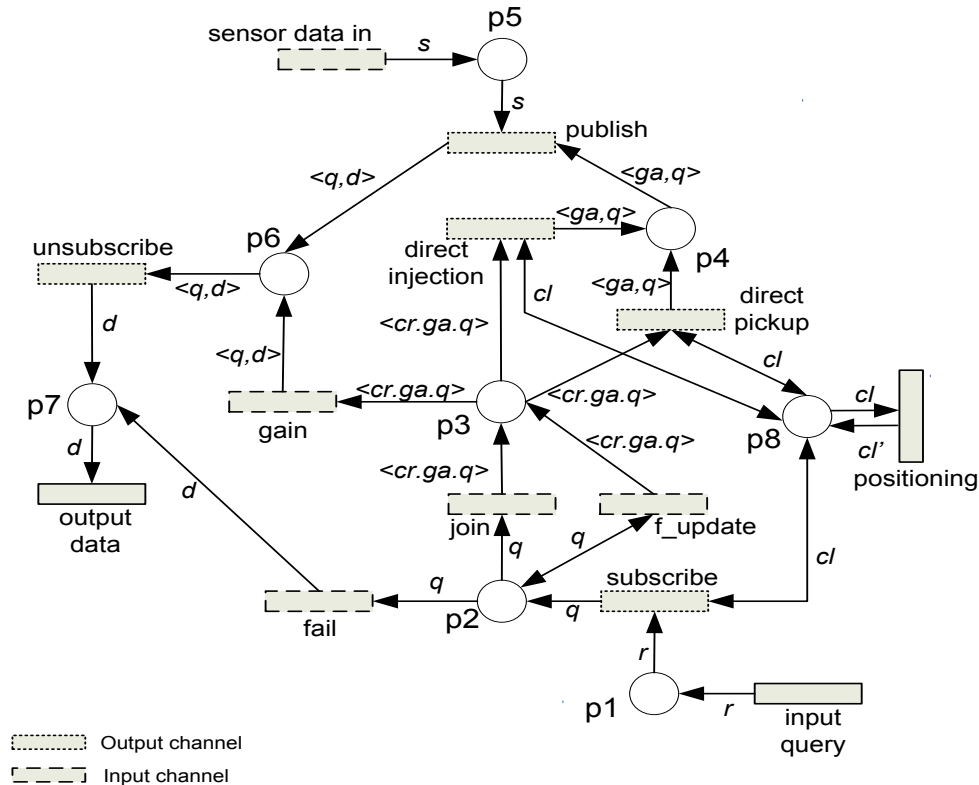


Figure 38. The mobile device's behavior model.

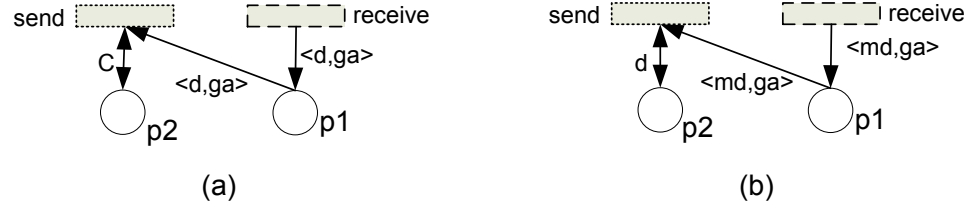


Figure 39. (a) Group address multicast (b) Cluster head data communication

### Semantic Definitions of Group Address Multicast's PrT Model

*Token Types:*

$$CLUSTER = ClusterID \times Cluster\_region \times GroupAddress \times \wp(MobileID)$$

$$\wp(p1) = DATA \times GroupAddress$$

$$\wp(p2) = \wp(CLUSTER)$$

*Transition Constraints:*

$$R(receive) = md \ ?(d, ga)$$

$$R(send) = \forall c \in C. (c[3] = ga \wedge \forall md \in c[4]. (md!d))$$

### Semantic Definitions of Cluster head Data Communication's PrT Model

*Token Types:*

$$MD = \wp(MobileID)$$

$$\wp(p1) = MD \times GroupAddress$$

$$\wp(p2) = DATA$$

*Transition Constraints:*

$$R(receive) = md \ ?(md, ga)$$

$$R(send) = md!(d, ga)$$

#### 6.1.3 A Query Request Scenario

Let us look at a simple scenario based on the behavior models defined in previous sections.

A campus WSN system with information-management server keeps the information of three cluster regions: library, cafeteria and gymnasium. The students on campus are usually interested in the conditions of these locations (for example, available space in the library). There are four students, John, Mary, Peter and Eric with their own mobile devices, and they are at different locations on campus. John and Mary are currently at the library; Peter is at the cafeteria and Eric is at the gym. Eric finished his workout and is heading to the cafeteria but wondering if the cafeteria is crowded or not. So Eric sends a query request through his mobile device to subscribe for information about the cafeteria. In the meantime, John and Mary finished their study at the library and are also heading to the cafeteria for lunch; they have sent query requests and subscribed for the information about the cafeteria. Peter has finished his lunch at the cafeteria and is heading to the gym. He subscribed the information for the gym. As such, two different common interest groups have been formed based on the interested regions: (1) John and Mary for the cafeteria, and (2) Peter for the gym. Eric's query request scenario is used as an example to demonstrate the interaction behaviors between Eric and the server. Let the current marking  $M_0$  of the server for the above scenario be the following:

$$M_0(p1) = \{\text{John, Mary, Peter, Eric}\}$$

$$M_0(p2) = \emptyset$$

$$M_0(p3) = \{(1, \text{library}, 1, \emptyset), (2, \text{cafeteria}, 2, \{\text{md1}, \text{md2}\}), (3, \text{gym}, 3, \{\text{md3}\})\}$$

$$M_0(p4) = \{(\text{John}, q1, \text{md1}, \text{library}, \text{cafeteria}, 10), (\text{Mary}, q1, \text{md2}, \text{library}, \text{cafeteria}, 10), \\ (\text{Peter}, q1, \text{md3}, \text{cafeteria}, \text{gym}, 10)\}$$

For simplicity, only Eric's mobile device net is demonstrated in which there are interactions with the server, the group address multicast and the cluster head. Initially,

only place  $p8$  has tokens, which stores the mobile ids and the current location of the mobile device. After Eric sent a query request through the mobile device's interface, a query request message containing the information of a query id, a mobile device id, the current location, the query region and a maximum waiting time for the query is generated. That is, there is a token  $q = \{q1, md4, gym, cafeteria, 10\}$  has been generated. Since the server is ready to input message through transition '*subscribe*', together with the transitions '*subscribe*' in the mobile device net, a synchronized communication occurs as a result of firing a pair of matching transitions. After firing the matching transitions, the server received the query request. That is, the token  $\{Eric, q1, md4, gym, cafeteria, 10\}$  is output to  $p2$  in both nets. This query simply says that the mobile device  $md4$  is interested in the information of the region *cafeteria*; and its current location is *gym*; and, this query is valid only within 10 seconds after the query is sent. The mobile device is ready to join a common interest group.

Next, the server checks if there exists a common interest group in mobile device  $md4$ 's current location *gym*. Refers to place  $p3$ , which holds the information of subscriptions and regions in the server net, there is a common interest group indicating cluster id '3', which is the region *gym*, linking a group address '3' and mobile device id ' $md3$ '. The server then responds with the friendly update message containing  $\{gym, 3\}$  through output channel ' $md4$ ', which is the communication channel to Eric's mobile device. The mobile device's input channel ' $f\_update$ ' simultaneously receives that message. Eric's mobile device then requests the sensor data through the communication channel '*direct pickup*' with the cluster head at the situated region. As soon as it got the response through '*sensor data in*' from the cluster head, the mobile device publishes the data to the group

address '3' through the firing of transition '*publish*' in Figure 38. The shared data is sent through a multicast, and the mobile device '*md3*', which is Peter, will receive the data. After instructing a friendly update, the server adds Eric's mobile device to the group of region '*cafeteria*' through a simultaneous firing of transition '*join*' in Figure 37 and Figure 38. Eric's mobile device received the group joining message containing  $\{cafeteria, 2\}$ , indicating that he has joined in the group address '2' in region *cafeteria*. After Eric joined the group, Mary happens to arrive at the cafeteria and gets the information of the cafeteria. She publishes the information for sharing to group address '2'. All the mobile devices in the same group, John and Eric, received the information of the cafeteria through input channel '*gain*'. After Eric received the data, he sends an unsubscribing message through transition '*unsubscribe*' with output channel *S* to server. Server receives the message and removes Eric's subscription information. The firing sequence of the mobile device net regarding Eric's query request is as follows.

$M_0 [input\ query > M_1 [subscribe > M_2 [f\_update > M_3 [direct\ pickup > M_4 [sensor\ data\ in > M_5 [publish > M_6 [join > M_7 [gain > M_8 [unsubscribe > M_9 [output\ data >$

The firing sequence of the server net regarding Eric's query request is:

$M_0 [subscribe > M_1 [f\_update > M_2 [join > M_3 [unsubscribe >$

The firing sequence of the group address multicasting net and the cluster head net regards to Eric's query request is:

$M_0 [receive > M_1 [send >$

The corresponding pairs of communication channels are listed in Table 11.

Let the firing sequence of Eric's request query be the sequence called *ReqSeq*, the transition firings of these nets be represented in the order of [server net, mobile device

net, group address multicast net, cluster head net], and  $\lambda$  be no transition firing in the net. The concurrently firing sequence *ReqSeq* of these nets regarding Eric's request query is as follows.

*ReqSeq* = [  $\lambda$ , *input query*,  $\lambda$ ,  $\lambda$  ], [ *subscribe*, *subscribe*,  $\lambda$ ,  $\lambda$  ], [ *f\_update* *f\_update*,  $\lambda$ ,  $\lambda$  ], [  $\lambda$ , *direct pickup*,  $\lambda$ , *receive* ], [  $\lambda$ , *sensor data in*,  $\lambda$ , *send* ], [  $\lambda$ , *publish*, *receive*,  $\lambda$  ], [ *join*, *join*,  $\lambda$ ,  $\lambda$  ], [  $\lambda$ , *gain*, *send*,  $\lambda$  ], [ *unsubscribe*, *unsubscribe*,  $\lambda$ ,  $\lambda$  ], [  $\lambda$ , *output data*,  $\lambda$ ,  $\lambda$  ]

Table 11. Communication channels.

Server	Mobile Device	Group Multicast	Cluster head
Subscribe(in)	Subscribe(out)		
F_update(out)	F_update(in)		
Join(out)	Join(in)		
Fail(out)	Fail(in)		
Unsubscribe(in)	Unsubscribe(out)		
	Publish(out)	Receive(in)	
	Gain(in)	Send(out)	
	Direct pickup(out)		Receive(in)
	Sensor data in(in)		Send(out)
	Direct injection(out)		Receive(in)

#### 6.1.4 Discussion

During this study, the applicability of two-level nested PrT nets is examined in modeling various aspects of an agent – autonomy, reactivity, pro-activeness, sociability,

and mobility. It is worth noting that, through this case study [91], I found the usefulness of formal modeling, which forces me to explicitly deal with all hidden assumptions and missing requirements. The messages and control flows between mobile devices and the server can be nicely modeled using two-level nested PrT nets. However, WSN involves low level communication mechanisms, such as the multicast. Although, the broadcasting of messages can be specified by a first order logic formula indicating all agent members, the details of low-level communications have to be abstracted away, such as identifying current coordinates (locations) of a mobile device.

## 6.2 E-Market

In the e-market study [36], an *Interaction Model* is defined to handle the coordination behaviors of a set of possible agent conversations in an e-market. A *Conversation* is an execution sequence, which is initiated by a requestor and ended with a successful commitment or terminated by a failure resulted from any participant that is engaged in the conversation. In a typical e-market, there are buyers and sellers engaging in some high-level negotiations for the goods.

Let us consider a simple conversation scenario at an e-market where seller and buyer auctioning goods. The conversation is in a format as: *sender: communicative act, message content* and *receiver*. An example of conversations is as follows.

*Seller:* request, 'sell book 30', broker

*Broker:* agree, 'posted book 30', seller

*Buyer:* request, 'buy book 25', broker

*Broker:* inform, 'sell book 25', seller

*Seller:* commit, 'commit book 25', broker

*Broker:* inform, 'buy book 25', buyer

*Buyer*: commit, ‘commit book 25’, broker

A higher level of abstraction is used to represent the message for demonstration purpose. The negotiation process about the transaction of a payment and the shipping detail between a seller and a buyer is abstracted away, since it is not relevant to the coordination behavior. The conversation starts from a request of a seller who wants to sell book for 30 dollars, the broker agreed with the request and posted the information. The buyer sends a request to the broker for buying the book. The broker informs the seller that there is someone wants to buy the book for 25 dollars and the seller agreed with the price. The broker informs the buyer and the deal is committed by the buyer.

First of all, there are three entities engaged in the conversation: a broker, a seller and a buyer. The broker is served as the coordinator thus modeled as the higher level host net, which provides the information service of auctioning goods. The buyer and seller are participants in the activity of auctioning goods, therefore modeled as agent nets at the lower level. According to the conversation scenario, the communicative acts in *verb* represent *actions*. These actions are transformed into transitions. For example, the seller has ‘*request*’ and ‘*commit*’ actions, which imply proactive and reactive behaviors, respectively. The actions should be linked to a message outgoing place. After received a message, a seller’s decision logic decides which action to be taken according to its local knowledge and policy. The seller’s, buyer’s and the broker’s interaction model are shown in Figure 40 and Figure 41, respectively .





Note: transition ‘*receive*’ and ‘*send*’ are used to input and output messages through communication channels. Transition *commit* is enabled when there is a previous request exists in place *p5* and a response message for that request is also available. If the message content cannot be identified, the message is discarded through transition ‘*fail*’. Transition ‘*set request*’ inputs message tokens from outside of the model. Transition ‘*reasoning*’ decides which action to be taken based on the received message. The initial marking  $M_0$  sets a message token with message id #1, sender agent id *a1*, book price 30 dollars, minimum acceptance price 25 dollars, and receiver id *s1*.

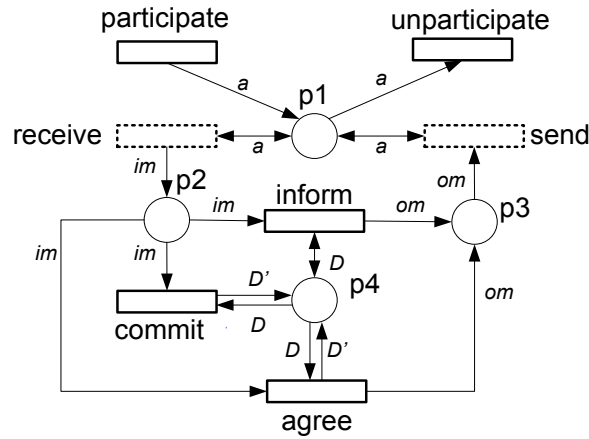


Figure 41. The broker's interaction model.

Token Types:

$$MESSAGE = MID \times SENDER \times ACT \times PRICE \times RECEIVER$$

$$DIRECTORY = MID \times SENDER \times ACT \times PRICE \times RECEIVER$$

$$\phi(p1) = AGENTNET$$

$$\phi(p2) = \phi(p3) = MESSAGE$$

$$\phi(p4) = DIRECTORY$$

Note: type *MESSAGE* and *DIRECTORY* are of the same type that is defined in the agent model. Type *AGENTNET* defines the net tokens.

*Transition Constraints:*

$$R(receive) = aid ? im$$

$$R(send) = aid = om[5] \wedge aid ! om$$

$$R(agree) = (\forall d \in D. (d[1] \neq im[1] \wedge D' = D \cup im) \wedge om[1] = im[1] \wedge om[2] = im[5] \wedge om[3] = 'posted' \wedge om[4] = im[4] \wedge om[5] = im[2])$$

$$R(inf\ orm) = \exists d \in D. (d[1] = im[1] \wedge om[1] = d[1] \wedge om[2] = d[5] \wedge om[3] = d[3] \wedge om[4] = d[4] \wedge om[5] = d[2])$$

$$R(commit) = \exists d \in D. (im[1] = d[1] \wedge im[3] = 'commit' \wedge D' = D - im)$$

Note: transition ‘*receive*’ and ‘*send*’ are used to input messages and output messages through channels respectively. Transitions ‘*participate*’ and ‘*unparticipate*’ allow agent nets enter and leave the system. Transition ‘*agree*’ sends a message for a successful posting back to agents. Transition ‘*inform*’ notifies agents that there is a matched deal. When the deal is committed, the information is deleted from the directory through transition ‘*commit*’.

### 6.2.1 Discussion

In this case study, a higher level of abstraction is used to describe the negotiation mechanism to avoid unnecessary redundancy in message interpretation. However, e-market involves intensive context-based interactions, which result in a too fine-grained abstract model that introduced additional complexity for model analysis.

There are several major research issues to be solved. First, an agent’s decision logic decides the degree of autonomy and the behavior of how an agent should react to external events. The decision logic largely depends on the knowledge base of the agent. Thus, knowledge representation in a Petri net model is a challenging issue. Second, all entities have to speak the same language in order to understand each other and the content of the exchanged information, which is usually domain specific. Third, message exchange in

multi-agent systems is often asynchronous, i.e. agents may not need to respond immediately or wait for responses. However, there may still be some temporal dependency among agent tasks. Fourth, some of the methodologies for MAS design used an organization view; for instance, the work in [42]. It is possible for an agent to be assigned more than one role based on requirements.

### **6.3 Disaster Mitigation**

Disaster management has been one of the major application areas for multi-agent systems due to its socially significant nature. Heterogeneous agents are engaged in an emergency scenario [9, 10, 11, 12, 89, 92]. The major concerns are resource management and emergency response among distributed agents.

In this case study [90], the multi-agent modeling approach is applied to a Business Continuity Information Network (BCIN) [89], which is aimed to prepare private sectors for a rapid recovery after major disasters. Since BCIN employs a role-based access control, the agent-oriented approach is used to model the interactions between agent roles and the BCIN system. At current stage, BCIN system supports static information sharing, including (1) the advisories from public agencies, and (2) the resources provided by private sectors. Yet, the interdependencies between recovery plans and the resources from private sectors were not studied. The availability of resources is rather dynamic in the aftermath of a disaster, and affects the feasibility of recovery plans. The objective of this work is to provide a dynamic model to study the interdependencies of activities and the dynamics of resource consumptions in BCIN prior to deployment to ensure system dependability.

We consider BCIN system as a coordination model to accommodate multiple agents in a multi-agent system context. BCIN currently employs a role-based access control (RBAC) mechanism [93]. Some use case scenarios are as follows.

*Scenario 1:* David, a member from EOC (Emergency Operations Center), intends to get the most up-to-date information about Hurricane Wilma, which has been announced as a category one hurricane. The advisory for Wilma has been published and is available in BCIN.

*Scenario 2:* Alice, a supervisor from Hardware Depot for emergency response, intends to assign John as the primary person to publish the resources provided by their company.

*Scenario 3:* Eric, the supervisor of EOC, is going to publish a new advisory for an incoming hurricane.

*Scenario 4:* Emily, a primary contact for emergency response from Shop Mart, has entered BCIN system and read information about resources provided by other companies. She decided to send a message to Hardware Depot, which has three power generators available. Shop Mart needs the generators for frozen foods.

For simplicity, this case study only demonstrates the above scenarios instead of all possible scenarios. However, the above scenarios are typical examples in accessing information in the BCIN system. The actors and their associated actions are listed in Table 12 based on the above scenarios.

Here, an actor is an external entity that interacts with the system. Each actor plays a different role regarding system access. That is, each role has different permission assignments (PAs) [93], which describe the operations that a role can perform in the system. To this end, an actor is modeled as an agent net, and BCIN system as the system

net that accommodates agents. In the following sections, the steps for generating agent nets and the system net with communication channels are introduced. Modeling examples based on the above scenarios in the BCIN system are given as well.

Table 12. Actors and their associated roles and actions.

Actors	Roles	Actions
David	Company Observer	read information
Alice	Company Supervisor	assign roles
John	Company Participant	publish resources
Eric	EOC Supervisor	publish advisories
Emily	Primary Contact	send messages

### 6.3.1 Modeling Interactions in an Agent Net

The behaviors related to permission assignments are considered as an *interaction aspect* of an agent net. An interaction aspect addresses the sociality of an agent and is called a *role model*. Since the information stored in BCIN database is organized based on categories, a typical flow of event for an action in a role model is: (1) send a *request(action, x, criteria)* to trigger the system process *action* regarding category *x* based on the *criteria*, and (2) get the result of *action* from a *receive(action, x, result)*. A ‘*request*’ denotes an outgoing data flow to the system, while a ‘*receive*’ denotes an incoming data flow from the system.

Formally, a role model *RM* with the net structure  $(P_{RM}, T_{RM}, F_{RM})$  is a PrT net, which models the interaction aspect of an agent net. The set of permission assignments *PAs* of an *RM* defines the set of operations *OP*, where each  $op \in OP$  is either a *request(action, x, criteria)* or a *receive(action, x, result)* operation. A *request* operation involves an outgoing data flow to the system, and a *receive* operation involves an incoming data flow

from the system. The set of transition  $T_{op} \subseteq T_{RM}$  models the set of  $OP$  in an  $RM$ . As a result, a role model  $RM$  with communication channels can be generated based on the algorithm in Figure 42.

```

1: set  $T_{op} = \emptyset$ ;
2:  $\forall op \in OP$ 
    {  $t_{op} = action$ ;
       $T_{op} = T_{op} \cup t_{op}$ ;
      add  $\bullet t_{op}$  and  $t_{op} \bullet$  and associated arcs;
       $\forall p \in (\bullet t_{op} \cup t_{op} \bullet)$ , define  $\varphi(p)$ ;
      define  $R_u(t_{op})$ ;
      /*  $R_u(t_{op}) \in R(t_{op})$  is a non-communication constraint */
      label each arc of  $t_{op}$  with sort-respecting variables;
      if  $action \in request$  {
           $R_c(t_{op}) = S!x$ ;
          /*  $S$  is the system  $id$ ,  $x \in L(t_{op}, p)$  and  $p \in t_{op} \bullet$  */
           $R(t_{op}) = R_u(t_{op}) \cup R_c(t_{op});$  }
      if  $action \in receive$  {
           $R_c(t_{op}) = S?e$ ;
          /*  $S$  is the system net  $id$ ,  $e \in L(p, t_{op})$  and  $p \in \bullet t_{op}$  */
           $R(t_{op}) = R_u(t_{op}) \cup R_c(t_{op});$  }
    }

```

Figure 42. An algorithm for generating a role model.

*Example1: A PrT net for a supervisor role model.*

Let us look at an example to build a role model for a *supervisor* role by applying the algorithm in Figure 42. Based on Table 12, the operations permitted for a *supervisor* role are read information, publish advisories, and update role assignments. Therefore, the

*action* in  $request(action, x, criteria)$  is instantiated and results in  $request(Read, x, criteria)$ ,  $request(Publish, x, criteria)$ , and  $request(Assign, x, criteria)$ .

For simplicity, only action *Return* is used in operation  $receive(action, x, result)$ . As a result, the set of operations *OP* is defined as  $\{request(Read, x, criteria), request(Publish, x, criteria), request(Assign, x, criteria), receive(Return, x, result)\}$ . The denotation of an *op* is not restricted, however should be differentiated in terms of the direction of an information flow. For example, all operations in *OP* denoted as *request/receive* can be denoted as *output/input* instead. After *OP* is defined, a PrT net for the *supervisor* role can be generated based on the algorithm in Figure 42. The resulting net structure for a *supervisor* role is shown in Figure 43, where relevant semantic definitions are given in Appendix A3. Note that, two boundary transitions (with no input or output places) ‘*Input*’ and ‘*Output*’ are added to denote the interfaces of the role model for non-interaction aspects.

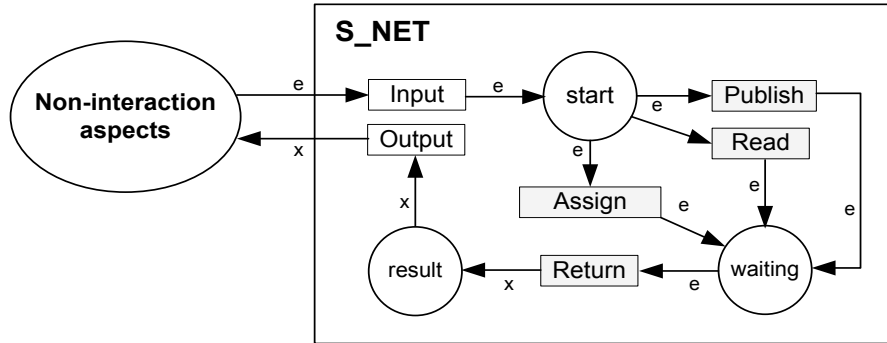


Figure 43. A role model for a supervisor role.

### 6.3.2 Modeling Interactions in the System Net

The system net has three major components in order to accommodate multiple agents; namely access controls, communications and resource controls. Access controls address



the security aspect of the system, while communications and resource controls address the coordination among agents.

#### *a. Modeling Role-based Access Control*

From a system view, each user has to be assigned a role for the purpose of access control. Based on Table 12, each user plays a different role. However, it is possible that different users play the same role and one user plays more than one role at the same time. Therefore, a *role* is considered as *a pattern of interactions* within the system. A role assignment happens at runtime, that is, a user becomes an actor enacting a role given by the system after successfully started a session. To this end, the strategy is to keep a net template for each distinct role in a repository. A net template is the behavior model of a role interacting with the system, and is activated while a valid user enacting the role. This strategy not only conforms to the agent-oriented design, but also allows the adaptation of interaction behaviors at runtime. Net templates can be built for every distinct roles based on the algorithm described in Figure 42. Activated role models are executing concurrently. In other words, the role models that are activated from the same template (users enacting the same role) have different markings (states).

Formally, a *RBAC* model is a tuple  $(RAs, PAs, Sessions, AssignRole, AssignPA)$ , where  $RAs$  is a set of  $U \times R$  relations of users  $U$  and roles  $R$ ,  $PAs$  is a set of  $R \times RM$  relations of  $R$  and role model  $RM$ ,  $Sessions$  is a set of user sessions represented by  $U \times R$ ,  $AssignRole: U \rightarrow R$ , is a function that maps a user in  $U$  to a role in  $R$ ,  $AssignPA: R \rightarrow RM$  is a function that maps a role in  $R$  to a role model  $RM$ . Each element in the *RBAC* model is mapped to an appropriate net element; for example,  $RAs$ ,  $PAs$ ,  $Sessions$  are modeled as places and the functions are modeled as transitions (see *RBAC* in Figure 31), where transition

*AssignRole* assigns a valid user a role based on predefined RAs, and transition *AssignPA* assigns PAs by activating an associated role model from net templates. Activated role models (net tokens) are kept in place *Activated* with their identifications. Note that boundary transitions *UserIn*, *UserOut*, *ActorOut* and *End* are added to generate or eliminate tokens. A session constraint [93] is enforced by the formula:  $\exists s \in S.(s = u)$  in transition *UserOut* to eliminate an invalid user token that has an active session in place *Sessions*. Relevant dynamic definitions can be found in Appendix A4.

*b. Modeling interactions and resource controls*

In the system net, a typical interaction sequence of an operation is: (1) an agent net sends the request for an operation (incoming information flow); (2) the system net performs the operation requested (resource controls); and, (3) the result is sent back to the agent net (outgoing information flow). The information flows involved in the above sequence are modeled by a pair of transitions with input/output channel commands to address the interactions with agent nets. Let the set of operations  $OP_s$  denotes the resource controls in a system net  $S$ ; a *RC* net with net structure  $(P_{rc}, T_{rc}, F_{rc})$  models the data and controls of an operation  $op \in OP_s$ , and there exists a pair of transition  $t_{in}$  and  $t_{out}$  model the input and output channel respectively, where  $t_{in} \in T_{rc}$  and  $t_{out} \in T_{rc}$ . The algorithm in Figure 44 describes the steps to build a *RC* net that models an operation  $op$ .

*Example 2: A PrT net for operation ‘read’ in system net.*

For simplicity, a transition *readDB* and a pair of input place and output place are used to model data access against BCIN database (resource controls). Input channel transitions ‘*Read*’ and output channel transition ‘*returnR*’ are added to address the data flows

from/to agent nets. The net structure for operation ‘*read*’ is shown in Figure 45. Note that a transition with a channel command in the system net has an input and an output places that hold agent net tokens, which can be instantiated for interactions. For example, places *Activated* and *p1* stores the agent net tokens to enable input and output channel transitions *Read* and *returnR* when interactions are desired. Detailed constraint definitions are elaborated in Appendix A4.

```

1: define  $P_{rc}$ ,  $T_{rc}$  and  $F_{rc}$  for operation  $op \in OP_s$ .
2: for all  $p \in P_{rc}$  define  $\phi(p)$ ;
3: for all  $t \in T_{rc}$  define  $R_u(t)$ ;
4: for all  $f \in F_{rc}$  define  $L$ ;
5: add an input channel  $t_{in}$  to  $T_{rc}$ ;
   define  $\bullet t_{in}$ ,  $t_{in} \bullet$  and associated arcs;
   define  $R_u(t_{in})$ ;
    $R_c(t_{in}) = a?x$ ; /*  $a$  is the agent net  $id$  */
    $R(t_{in}) = R_u(t_{in}) \cup R_c(t_{in})$ ;
6: add an output channel  $t_{out}$  to  $T_{rc}$ ;
   define  $\bullet t_{out}$  and  $t_{out} \bullet$  and associated arcs;
   define  $R_u(t_{out})$ ;
    $R_c(t_{out}) = a!e$ ; /*  $a$  is the agent net  $id$  */
    $R(t_{out}) = R_u(t_{out}) \cup R_c(t_{out})$ ;

```

Figure 44. The algorithm for generating a RC net.

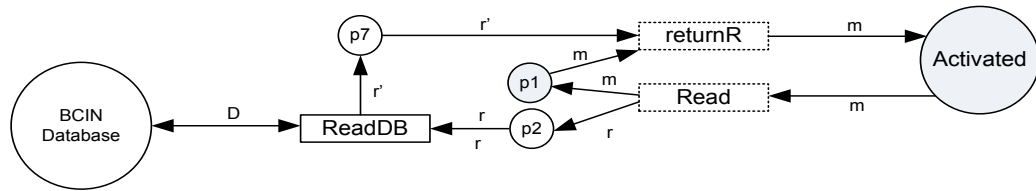


Figure 45. A resource acquisition operation.

### *The BCIN Net*

A BCIN net based on the operations in Table 12 is shown in Figure 31 by applying the algorithms described previously for modeling RBAC, communication channels and resource controls. The detailed semantic definitions for the net are given in Appendix A4.

The resource controls addressed are rather straightforward in this example for demonstration purpose. However, further policies for resource sharing during a global process can be modeled in two ways: (1) define further constraints (rules) in output channel transitions for dispatching resources; and/or (2) add an input place (or multiple input places) representing additional pre-conditions for the enabling of output channel transitions. Either way, the policies are enforced at the point of synchronization, which results in an outgoing information flow downward to an agent net.

#### 6.3.3 Soundness of BCIN Net

There are several important properties regarding information access in BCIN need to be ensured: (1) an invalid user cannot play a role in the system; (2) an actor cannot perform the operations that were not assigned; (3) a user cannot play more than one role at the same time; and, (4) for a request received from an agent net, there will be a response to the agent net eventually. The prove sketches for the above properties are given as follows.

*Property (1):* The constraint formula:  $\forall r \in R. (r[1] \neq u)$  defined in transition ‘*UserOut*’ (Appendix A4) enforces the elimination of invalid user tokens, where  $R$  is the set of valid role assignments.

*Property (2):* The constraint formula:  $\exists p \in P. (p[1] = a[2])$  defined in transition *AssignPA* make sure that an associated role model is available to be activated. Since role models are

predefined based on PAs, user behaviors are well controlled. The constraint formula  $\forall p \in P. (p[1] \neq a[2])$  defined in transition '*ActorOut*' eliminates the user token that has not yet been assigned any permissions.

*Property (3):* A session constraint is enforced by the formula  $\exists s \in S. (s = u)$  in the constraint definition of transition *UserOut*; that is, if a user is already in *Sessions*, it is considered as an invalid user and is eliminated.

*Property (4)* The semantic definitions for a *RC* net constructed by applying algorithm given in Figure 44 has to be carefully defined to make sure that if there is a request token *rq* in *p* such that  $p \in t_{in} \bullet$ , there will be a result token *rs* in *p* such that  $p \in \bullet t_{out}$ , and the agent  $id \in rq$  equals to the agent  $id \in rs$ . For example, in Figure 45, there will be an agent token in place *p1* and a request token in place *p2* with the same agent *id* if transition *Read* fired. Since the constraint definition of transition *ReadDB* (Appendix A4) encompasses exception handling, there will be a result token sent to place *p7* eventually.

#### 6.3.4 Discussion

This work provides a dynamic model to study the interdependencies of activities and the dynamics of resource consumptions in BCIN. Yet, further studies for the interdependencies between the recovery plans and resources from private sectors are needed, since the availability of resources is rather dynamic in the aftermath of a disaster, and affects the feasibility of the recovery plans from private sectors. Although, it is difficult to obtain the detail information regarding the recovery plans from private sectors, recovery plans from private sectors are essential for further studies of the dynamics of agent behaviors during a disaster scenario.

## CHAPTER 7

### CONCLUSION

To address the representation and analysis of complex systems with the MAS architecture, this dissertation research developed a framework that provides a comprehensive methodology for modeling and analyzing MAS based on model-oriented formal methods. An important element of the framework is modularity, which is fulfilled by an agent-oriented modeling methodology incorporated with aspect-oriented concepts. As a result, the MAS model is a modular composition of multiple agent nets, and the individual agent net is a modular composition of agent features. The advantages of a modular system model includes: (i) *adaptability* for future extensions; (ii) *reusability* of modular components; (iii) *conciseness* of the model; and (iv) *compositionality* for incremental analysis.

The underlying formalism of the framework is PrT nets, which is adapted and infused with agent-oriented concepts for modeling MAS. A key idea is to support the modeling of the dynamic structure in MAS. The nested PrT nets defined in this study facilitate the modeling of a hierarchical MAS architecture, which can be changed dynamically. Nested PrT nets can be checked by the model checker SPIN. SPIN is an excellent tool for model checking logical correctness of distributed software systems. This study takes advantage of the functionalities in the model checker SPIN to verify nested PrT nets. Given a nested PrT nets describing the behavior of a MAS, a PROMELA program can be generated through the model transformation technique developed in this study for model checking.

The contributions, limitations and future works based on this dissertation research are discussed in the following sections.

## **7.1 Contributions**

The framework presented in this dissertation provides a systematic approach for modeling and analyzing complex systems within the MAS context. Given a multi-agent system with a set of agent roles, their behavior models can be constructed and analyzed following the methodology provided in this dissertation. Compared to other works based on Petri nets([24, 25, 26, 27, 28, 29, 30] in Table 1), this dissertation research provides a more comprehensive framework featured a well-defined formal model with a dynamic structure, agent communication notations, agent coordination modeling, a comprehensive modeling methodology, and the tool support for machine analysis. Major contributions of this dissertation research are summarized as follows.

- (1) Developed a process model based on PrT nets to address the modeling of multi-agent systems.
  - Support agent-oriented modeling in which the system model is a modular composition of multiple agent nets;
  - Support the modeling of a dynamic MAS architecture in which the formal MAS model is a two-level nested PrT net;
  - Support the modeling of dynamic agent communications with the instrumentation of channel commands in transition constraints.
- (2) Developed a methodology for constructing formal MAS models.

- Provide an aspect orientation technique for constructing individual agent nets with different features;
  - Provide a technique for agent coordination modeling to ease the construction of formal MAS models.
- (3) Developed a model transformation technique for model checking formal MAS models.
- Define a set of translation rules that systematically transform nested PrT nets to PROMELA programs in SPIN for automatic model analysis.

## 7.2 Limitations

The framework supports the modeling of essential characteristics found in MAS [1, 4, 8, 94, 95]. Some limitations are discussed as follows.

- (1) The MAS model focuses on the interdependencies between agent nets, rather on the computations. Therefore, the detail of how to implement the search algorithm for solving a problem efficiently using an agent program [32] is outside the scope of this study.
- (2) The nested PrT nets defined in this study is limited to a two-level nested net structure under the assumption of a two-layered multi-agent architecture. However, it can be extended to a multi-level structure by defining the relations between mediator agent nets.
- (3) Agent communications are in a one-to-one and unidirectional fashion through channel commands instrumented in transition constraints. Transitions with channel commands can fire many times as long as there are sufficient tokens. Since channel names are agent identifications, broadcasting or one-to-many communication can be achieved



by defining the constraint formula and tokens. For example, given a set of agent identifications, send the same message token to all members of the set.

- (4) Agent communications involve high-level knowledge exchange in which the context of the knowledge is usually domain specific. Thus, the structure of the messages is difficult to be generalized in the abstract models. Therefore, in this work, the messages exchanged among agents are in a simpler structure that is only relevant to the behavior models. The idea is to focus on the control flows with regard to system resources, and to produce a smaller sufficient model for model checking. This approach is applicable to most MAS domains, such as control systems, workflow managements and disaster mitigations where the resource management and controls are the central issues. However, for application domains where intensive context-based interactions are involved, such as the e-commerce, the messages need to be handled with a different approach. For example, based on our e-market case study, there are certain patterns with regard to agent interactions. This can be dealt with by naming all possible interaction patterns and modeling each pattern as an aspect as defined in Chapter 4. That is, the interactions can be treated as agent features. Another way to handle content-rich messages is to use only one pair of input and output channels to handle incoming and outgoing messages uniformly in each agent net. As a result, the interpretation of the messages is a part of the agent behaviors.
- (5) The BDI model provides a theoretical foundation to build intelligent computer systems by explicitly modeling the mental states to mimic the reasoning behaviors of human. Based on the case study in [83], PrT nets can naturally model agent knowledge in terms of logical formulas. However, agent reasoning involves domain

specific ontology and inference techniques, which will result in a too fine-grained abstract model for model checking.

(6) There are limitations for model checking nested PrT nets:

- Given that nested PrT nets are very expressive, they need to be restricted in order to be properly translated into PROMELA models. That is, due to tractability, the sort of a place is restricted to the data types supported in PROMELA, and the quantity of tokens in the place is finite. As a result, the transformed concrete model to be analyzed has a finite state space such that the model execution in the model checker SPIN will terminate appropriately.
- For simplicity, all transitions in the PROMELA model fire immediately after the guard conditions are evaluated to be true, and the firings are interleaving given that this restricted semantics does not affect the verification of state-based properties.
- The properties can be checked are restricted within the scope of the model checker SPIN has to offer with XSpin version 5.2.3.

### **7.3 Future Works**

Based on the limitations described in the previous section, several research directions are possible.

(1) Extension of current framework.

Currently, the framework supports two-level nested PrT nets. It can be extended to a multi-level structure. An idea for this extension is to define the relation among mediator agent nets. That is, the mediator agent of an agent community serves as a proxy to other communities.

(2) More case studies.

Three different application domains have been studied in this dissertation research, namely wireless sensor network, disaster mitigation and e-market. However, there are other domains [1] need to be further investigated.

(3) Context and knowledge representation problems.

Agents are heterogeneous computation entities in the MAS context. Thus, each agent has its own computational logic. However, an essential element in MAS is coordination in which agents must communicate within the same context. The context of the high-level knowledge exchanged is considered as the precondition of an agent plan. That is, an agent reasons about the context of knowledge it currently has to decide whether or not a plan is feasible. How to define the meta-knowledge for agent communications is an interesting topic. For example, represent resources, policies and trust metrics among distributed agencies during an emergency response scenario.

(4) Tool development.

One of the key elements to increase the value of the proposed methodology is to develop a tool that provides a reasonable user interface for easy prototyping and analysis. This can be done in two directions.

- Integrate nested PrT nets with current tools. Recently, a tool based on SAM (Software Architecture Model) [77] has been developed to support the editing of two-level nested PrT nets. The translation rules defined for model transformation in this dissertation research provides a foundation for further tool development in model analysis by integrating SPIN as the model checker at the back-end.

- Develop a general-purpose tool for modeling and simulation of agent-based systems. Dynamic interactions in a multi-agent system are difficult to be directly observed. A hand-on tool for simulation will help. However, current state-of-the-art tools for agent-based modeling and simulation need to be investigated first.

## REFERENCES

- [1] M. Wooldridge: *An Introduction to Multiagent Systems*, J. Wiley, New York, 2002.
- [2] Edmund H. Durfee: *Distributed Problem Solving and Planning*, Lecture Notes in Computer Science, Vol. 2086, pp.118-149, 2001.
- [3] L. Baresi, E.D. Nitto, C. Ghezzi: *Toward Open-World Software: Issue and Challenges*. IEEE Computer 39(10): 36-43, 2006.
- [4] K. P. Sycara: *Multi-Agent Systems*, AI Magazine, Vol. 19, No. 2, pp.11-12, 1998.
- [5] M. Wooldridge and P. Ciancarini, *Agent-Oriented Software Engineering*, The State of The Art. Lecture Notes in Computer Science, Vol. 1957, 2001, pp. 1–28.
- [6] M. Wooldridge: *Agent-Based Software Engineering*, IEEE Proceedings on Software Engineering, pp. 26-37, 1997.
- [7] F. Bergenti, M.-P. Gleizes and F. Zambonelli: Eds. *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Volume 11. Springer-Verlag, 2004.
- [8] N. R. Jennings and M. Wooldridge, *Agent-Oriented Software Engineering*. Proceedings of the 9th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, 2000.
- [9] F. Fiedrich, *an HLA-based Multi-agent System for Optimized Resource Allocation after Strong Earthquakes*. In Proceedings of WSC 2006, Monterey, pp. 486-492.
- [10] H. Kitano, and S. Tadokoro, *RoboCup Rescue: A Grand Challenge for Multi-agent and Intelligent Systems*. AI Magazine Vol. 22, No. 1, 2001, pp. 39–52.
- [11] K. F.R. Liu, *Agent-Based Resource Discovery Architecture for Environmental Emergency Management*, Expert System with Applications, Vol. 27, No.1, July 2004, pp. 77-95.
- [12] X. Pan, C. Han, K. Dauber, K. H., *A Multi-Agent Based Framework for the Simulation of Human and Social Behaviors During Emergency Evacuations*, AI & Society, Vol. 22, No. 2, 2007, pp.113-132, Springer-Verlang.
- [13] J. M. Wing: *A Specifier's Introduction to Formal Methods*. IEEE Computer, Vol. 23, No. 9, pp. 8-24, 1990.
- [14] G. Barrett: *Model checking in practice: The T9000 Virtual Channel Processor*, IEEE Transactions on Software Engineering, Vol. 21, No.2, pp: 69–78. 1995.

- [15] E. M. Clarke, O. Grumberg and D. A. Peled: “Model Checking”, The MIT Press, 1999.
- [16] M. Wooldridge. Reasoning about Rational Agents. The MIT Press: Cambridge, MA, 2000.
- [17] H. J. Genrich, Predicate/Transition nets. Advances in Petri Nets 1986, pp. 207–247.
- [18] C. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.
- [19] R. Valk: Petri nets as token objects: An introduction to elementary object nets, Application and Theory of Petri Nets, Vol. 1420 of LNCS, pp. 1-25, June 1998.
- [20] R. Valk: Concurrency in communicating object Petri nets, Concurrent Object-Oriented Programming and Petri Nets, Lecture Notes in Computer Science, Berlin, 2000, Springer-Verlag.
- [21] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J. M. Loingtier, J. Irwin: Aspect-oriented programming. In Proceedings of the European Conference on Object-Oriented Programming (ECOOP). Springer-Verlag LNCS 1241, June 1997.
- [22] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. Griswold: Getting Started with AspectJ, Communication of ACM, Vol. 44, No. 10, pp. 59-65, 2001.
- [23] G. J. Holzmann: The Model Checker Spin, IEEE Transactions on Software Engineering, Vol. 23, No. 5, May 1997, pp. 279-295.
- [24] M. Kohler, D. Moldt, H. Rölke: Modeling the Structure and Behavior of Petri Net Agents, Application and Theory of Petri Nets 2001, LNCS Vol. 2075, pp: 224-241, 2001.
- [25] M. Kohler, H. Rolke, Modeling Mobility and Mobile Agents Using Nets Within Nets, Proc. of International Conf. on Application and Theory of Petri Nets, LNCS vol. 2679 (2003), 121-139.
- [26] D. Moldt, F. Wienberg: ”Multi-Agent Systems Based on Coloured Petri Net”, The Proceedings of the 18th International Conference on Application and Theory of Petri Nets”, LNCS 1248, pp.82-101, 1997.
- [27] D. Xu, J. Yin, Y. Deng and J. Ding: A Formal Architecture Model for Logical Agent Mobility. IEEE Transactions on Software Engineering. Vol. 29, No. 1, pp. 31-45, Jan. 2003.
- [28] J. Ding, P. J. Clarke, D. Xu, and X. He: A Formal Model-Based Approach for Developing an Interoperable Mobile Agent System, International Workshop on Agent-Oriented Software Development Methodology, 2005.

- [29] H. Xu and S. M. Shatz, A Framework for Model-based Design of Agent-oriented Software. *IEEE Transactions on Software Engineering*, 2003, pp. 15–30.
- [30] J. Lian, Sol M. Shatz,: A Modeling Methodology for Conflict Control in Multi-Agent Systems, *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)* 2008, Vol 18, No. 3, pp.263-303.
- [31] N. R. Jennings, K. Sycara, M. Wooldridge, “A Roadmap of Agent Research and Development”, *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 1, No. 1, pp. 7-38, 1998.
- [32] S. Russell, P. Norvig: *Artificial Intelligence: a Modern Approach*, Second Edition, Prentice Hall, 2003.
- [33] K. Sycara, K. Decker, A. Pannu: Distributed Inteligent Agents, *IEEE Expert*, Vol. 11, pp. 36-46, 1996.
- [34] E. Ekici, Y. Gu, D. Bozdag, “Mobility-based communication in wireless sensor networks”. *IEEE Communications Magazine*, Vol.44, no.7 (2006), 56-62.
- [35] H. S. Nwana: “Software Agents: An Overview”, *Knowledge Engineering Review*, Vol. 11, No. 3, pp. 205-244, 1996.
- [36] L. Chang, J. Ding, X. He, S. Shatz: A Formal Approach for Modeling Software Agents Coordination, *Comm. of SIWN*, Vol. 3, 2008, pp.58-64.
- [37] T. Murata, Petri Nets: Properties, Analysis and Applications, an invited survey paper, *Proceedings of the IEEE*, Vol.77, No.4 pp.541-580, April, 1989.
- [38] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley, September, 2003.
- [39] A. Pnueli, The Temporal Logic of Programs, *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, Providence, R.I., pp. 46-57, 1977.
- [40] D. Kinny, M. Georgeff and A. Rao, “A Methodology and Modeling Technique for Systems of BDI Agents,” *Proceedings of the Seventh European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, 1996.
- [41] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini: Tropos: An Agent-Oriented Software Development Methodology. In *Journal of Autonomous Agents and Multi-Agent Systems*. May 2004. Kluwer Academic Publishers.
- [42] F. Zambonelli, N. R. Jennings and M. Wooldridge: Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology*, Vol. 12, No.3, pp.417 – 470, 2003.

- [43] M. P. Singh, Synthesizing Coordination Requirements for Heterogeneous Autonomous Agents, *Autonomous Agents and Multi-Agent Systems*, Vol.3, No. 2. pp. 107-132, 2000.
- [44] N. Jennings: Coordination Techniques for Distributed Artificial Intelligence, in G. O'Hare and N. Jennings, Eds., *Foundation of Distributed Artificial Intelligence*, Sixth-Generation Computer Technology Series, pp. 187-210, 1996.
- [45] D. Gelenter and N. Carriero. Coordination Languages and Their significance. *Communications of the ACM*, 35(2):97–107, February 1992.
- [46] N. H. Minsky, T. Murata. On Manageability and Robustness of Open Multi-agent Systems. In C. Lucena, A. Garcia, A. Romanovsky, J. Castro, P. Alencar Editors, *Computer Security, Dependability, and Assurance*. LNCS 2940, February 2004.
- [47] R.G. Smith: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, *IEEE Transactions on Computers*, Vol. C-29, No.12, December 1980.
- [48] C. Silva, J. Araujo, A. Moreira, J. Castro, D. Penaforte, A. Carvalho: Towards an Aspect Oriented Modeling in Multi-agent Systems, *Proceedings of the III Brazilian Workshop on Aspect-Oriented Software Development (WASP 2006)*, October 17, 2006, Florianópolis, Santa Catarina, Brasil.
- [49] A. Garcia, C. Chavez, and R. Choren: An Aspect-Oriented Modeling Framework for Designing Multi-Agent Systems, *Lecture Notes in Computer Science*, Volume 4405/2007, pp. 35-50, Springer-Verlag, Berlin Heidelberg 2007.
- [50] B. Bauer: Extending UML for the Specification of Interaction Protocols, Submission for 6th the Call for Proposal of FIPA and revised version of FIPA-99.
- [51] B. Bauer, J. P. Muller, J. Odell: Agent UML: A Formalism for Specifying Multi-agent Interaction, LNCS Vol. 1957, pp. 109-120, Springer 2001.
- [52] J. Odell, H. V. D. Parunak, and B. Bauer: Representing Agent Interaction Protocols in UML. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering — Proceedings of the First International Workshop (AOSE-2000)*. Springer-Verlag: Berlin, Germany, 2000.
- [53] L. Cabac, D. Moldt: Formal Semantics for AUML Agent Interaction Protocol Diagram, LNCS Vol. 3382, pp. 47-61, Springer-Verlag 2005.
- [54] FIPA, Interaction Protocol Library Specification, 2000.
- [55] R. Cervenka and I. Trencansky: The Agent Modeling Language-AML: A Comprehensive Approach to Modeling Multi-Agent Systems, Birkhauser Verlag AG, 2007.



- [56] V. T. da Silva, R. Choren, C. J. P. de Lucena: Using the MAS-ML to Model a Multi-agent System, *Software Engineering for Multi-Agent Systems II, Research Issues and Practical Applications*, pp. 129-148, Springer 2004.
- [57] V. T. da Silva, A. F. Garcia, A. Brandão, C. Chavez, C. J. P. de Lucena, P. S. C. Alencar: Taming Agents and Objects in Software Engineering, *Software Engineering for Large-Scale Multi-Agent Systems, Research Issues and Practical Applications*, pp. 1-26, Springer 2003.
- [58] H. Barringer, M. Fisher, D. Gabbay, G. Gough and R. Owens, METATEM: A Framework for Programming in Temporal Logic, *Proceedings on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness. LNCS*, Vol. 430, pp.94-129.
- [59] A. S. Rao, M. Georgeff: Modeling Rational Agents within a BDI Architecture, *Proceedings of Knowledge Representation and Reasoning*, pp.473-484, 1991.
- [60] A. S. Rao, M. Georgeff: BDI Agents: From Theory to Practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, San Francisco, CA, June 1995.
- [61] X. He: PZ nets—A Formal Method Integrating Petri net with Z, *Information and Software Technology*, Vol. 43, No.1, pp. 1-18, 2001.
- [62] M. d’Inverno, D. Kinny, M. Luck, M. Wooldridge: A formal specification of dMARS, *Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages*, pp.155–176. Springer-Verlag, 1997.
- [63] M. P. Luck, N. Griffiths and M d’Inverno, From Agent Theory to Agent Construction: A Case Study. *Proceedings of the ECAI’96 Workshop on Agent Theories, Architectures, and Languages: Intelligent Agents III 1997*, Vol. 1193, Springer-Verlag: Heidelberg, Germany, pp. 49–64.
- [64] L. Cabac, M. Duvigneau, D. Moldt, H. Rolke: Modeling Dynamic Architectures Using Nets-Within-Nets, *Application and Theory of Petri Nets 2005, LNCS 3536*, pp.148-167, Springer-Verlag.
- [65] J. Lian and S. M Shatz, Potential arc: A Modeling Mechanism for Conflict Control in Multi-agent Systems. *Proceedings of the 4th Symposium on Design, Analysis, and Simulation of Distributed Systems (DASD-06) 2006*, pp. 467–474.
- [66] C. Hanachi and C. Blanc, “Protocol Moderators as Active Middle-Agents in Multi-Agent Systems,” *Autonomous Agents and Multi-Agent Systems*, Vol. 8, No. 2, pp. 131-164, 2004.
- [67] J. R. Celaya, A. A. Desrochers, R. J. Graves: Modeling and Analysis of Multi-agent Systems using Petri Nets, *Journal of Computers*, Vol. 4, NO. 10, October, 2009.

- [68] D. Xu, R. Volz, T. Loerger, J. Yen: Modeling and Verifying Multi-agent Behaviors Using Predicate/Transition Nets, Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, pp.193-200, 2002.
- [69] S. Christensen, N. Hansen: "Colored Petri nets extended with channels for synchronous communication", Proc. of International Conf. on Application and Theory of Petri Nets, (1994), 159-178.
- [70] O. Kummer: Simulating Synchronous Channels and Net Instances, In Jörg Desel, Peter Kemper, Ekkart Kindler, and Andreas Oberweis, editors, Forschungsbericht Nr. 694: 5. Workshop Algorithmen und Werkzeuge für Petrinetze, number Forschungsbericht Nr. 694, pages 73-78. Fachbereich Informatik, Universität Dortmund, 1998.
- [71] M. Wooldridge, N. Jennings: Intelligent Agents: Theory and Practice, The Knowledge Engineering Review, Vol. 10, No. 2, pp.115-152, 1995.
- [72] Y. Fu, Z. Dong, and X. He: "A Translator of Software Architecture Design from SAM to Java". International Journal of Software Engineering and Knowledge Engineering, vol. 17, no.6, 2007, 709-755.
- [73] X. He: "Translating Hierarchical Predicate Transition Nets into CC++ Programs", Information and Software Technology, vol.42, no.7, 2000, 475-488.
- [74] S. Lewandowski and X. He: "Generating Code for Hierarchical Predicate Transition Net Based Designs", Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE2000), Chicago, July, 2000, 15-22.
- [75] S. Philippi: Automatic Code Generation from High-Level Petri Nets for Model Driven System Engineering, The Journal of Systems and Software, Vol. 79, 2006, pp. 1444-1455.
- [76] G. Argote, P. Clarke, X. He, Y. Fu, and L. Shi: "A Formal Approach for Translating a SAM Architecture to PROMELA", Proc. of the International Conference on Software Engineering and Knowledge Engineering (SEKE08), San Francisco, July, 2008.
- [77] X. He and Y. Deng: A Framework for Developing and Analyzing Software Architecture Specifications in SAM. The Computer Journal, Vol. 45, No. 1, 2002, pp. 111-128.
- [78] X. He, H. Yu, T. Shi, J. Ding, and Y. Deng: "Formally Analyzing Software Architectural Specifications Using SAM", Journal of Systems and Software, vol.71, no.1-2, 2004, 11-29.
- [79] K. Jensen, L. M. Kristensen and L. Well: "Coloured Petri Nets and CPN Tool for Modelling and Validation of Concurrent Systems", International Journal on

Software Tools and Technology Transfer, vol. 9, no. 3-4, pp. 213-254, Springer Berlin, June, 2007.

- [80] K. Sycara, J. A. Giampapa, B. Langley, M. Paolucci: The RETSINA MAS, a Case Study, in SELMAS 2003, pp.232-250. Springer-Verlag.
- [81] K. Fischer, J. P. Muller, M. Pischel: A Pragmatic BDI Architecture, Intelligent Agent II, Vol. 1037, pp.203-218, Springer-Verlag, 1995.
- [82] P. R. Cohen, H. J. Levesque. Intention is choice with commitment. Artificial Intelligence, Vol. 42, pp.213–261, 1990.
- [83] L. Chang and X. He: "Towards Adaptable BDI Agent: a Formal Aspect-Oriented Modeling Approach", Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE09), Boston, July, 2009, pp. 189-193.
- [84] D. Xu, K. E. Nygard: Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets. IEEE Transactions on Software Engineering, Vol.32, No.4, pp.265-278, IEEE Press.
- [85] H. Yu, D. Liu, X. He, L. Yang, S. Gao: Secure Software Architectures Design by Aspect Orientation, Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, pp.47-55, 2005, IEEE Computer Society.
- [86] A. Garcia, U. Kulesza, C. Lucena: Aspectizing Multi-agent Systems: From Architecture to Implementation, SELMAS, LNCS Vol. 3390, pp.121-143, February 2005, Springer Berlin / Heidelberg.
- [87] E. A. Emerson, J. Srinivasan: Branching Time Temporal Logic, LNCS, Vol. 354, Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, pp.123-172, Springer Berlin / Heidelberg, 1989.
- [88] R. B. France, B. Rumpe: "Model-Driven Development of Complex Software: A Research Roadmap", Proc. of 2007 Future of Software Engineering, pp. 37-54.
- [89] K. Saleem, S. Luis, Y. Deng, S.C. Chen, V. Hristidis, T. Li: Towards a Business Continuity Information Network for Rapid Disaster Recovery, Proceedings of the 9th Annual International Conference on Digital Government Research, Partnerships for Public Innovation, Montreal, Canada, pp. 107-116, 2008.
- [90] L. Chang and X. He: "A Multi-Agent Model for a Business Continuity Information Network", Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering 2010 (SEKE10), San Francisco, CA, July, 2010.

- [91] L. Chang, X. He, J. Lian, and S. Shatz: "Applying a Nested Petri Net Modeling Paradigm to Coordination of Sensor Networks with Mobile Agents", Proc. of Workshop on Petri Nets and Distributed Systems 2008, Xian, China, June, 2008, pp.132-145.
- [92] N. Okada, E. Hideshima: A Petri Net Approach for Modeling Bottlenecks in the Restoration and/or Restructuring Process of Multiplex disaster-Damaged Urban Infrastructure Systems, Journal of Natural Disaster Science, Vol. 17, No. 2, 1995, pp.75-86.
- [93] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman: Role-Based Access Control Models. IEEE Computer Vol. 29, No.2, pp. 38-47, 1996.
- [94] M. P. Luck, P. McBurney, and Ch. Preist: Agent Technology: Enabling Next Generation Computing (A roadmap for Agent-Based Computing), AgentLink II, January 2003.
- [95] M. P. Luck, P. McBurney, J. Gonzalez-Palacios: Agent-based Computing and Programming of Agent Systems. LNCS 3862, pp. 23-37, Springer, 2006.

## APPENDIX A

### SEMANTIC DEFINITIONS

#### A1. Figure 21.

##### *Diesel\_consumer*

$P = \{factory, store, GasStation, ToPump, standby, pumped, CreditCard\};$   
 $T_c = \{Park, SlideCard, PumpDiesel, Deny\};$   
 $T_u = \{ToStore, ToGasStation, Go, ToFactory\};$   
 $\varphi(factory) = \varphi(store) = \varphi(GasStation) = CAR \times INTEGER;$   
 $\varphi(standby) = \varphi(ToPump) = \varphi(pumped) = CAR \times INTEGER \times INTEGER;$   
 $CAR = \{sedan, truck\}; \varphi(CreditCard) = INTEGER;$   
 $R(ToStore) = c[2]=1; R(ToGasStation) = c[2]=0; R(Go) = c[2]=1 \wedge g=1;$   
 $R(ToFactory) = \lambda; R(Park) = S?st; R(SlideCard) = S!<c[1], cr, st>; R(PumpDiesel)$   
 $= S?g \wedge c'[2]=1; R(Deny) = S?cr;$   
 $M_0(factory) = \{<truck, 0>\}; M_0(CreditCard) = \{<1>\}; M_0(store) = \emptyset;$   
 $M_0(GasStation) = \emptyset; M_0(ToPump) = \emptyset; M_0(standby) = \emptyset; M_0(pumped) = \emptyset;$   
 $F$  and  $L$  are as seen in the Figure 21(a).

##### *Gas\_producer*

$P = \{Ready, Orders, Diesel, Regular\};$   
 $T_c = \{TakeOrder, SendDiesel, SendRegular\};$   
 $T_u = \{ProduceRegular, ProduceDiesel\};$   
 $\varphi(Ready) = INTEGER; \varphi(Orders) = INTEGER;$   
 $\varphi(Diesel) = \varphi(Regular) = INTEGER;$   
 $R(TakeOrder) = S?o; R(SendDiesel) = S!ds; R(SendRegular) = S!rs;$   
 $R(ProduceRegular) = o=2 \wedge rs=1; R(ProduceDiesel) = o=1 \wedge ds=1;$   
 $M_0(Ready) = \{<1>\}; M_0(Orders) = \emptyset; M_0(Diesel) = \emptyset; M_0(Regular) = \emptyset;$   
 $F$  and  $L$  are as seen in the Figure 21(b).

### ***Regular\_consumer***

$P = \{home, school, GasStation, ToPump, standby, pumped, CreditCard\};$   
 $T_c = \{Park, SlideCard, PumpRegular, Deny\};$   
 $T_u = \{ToSchool, ToGasStation, Go, ToHome\};$   
 $\varphi(home) = \varphi(school) = \varphi(GasStation) = CAR \times INTEGER;$   
 $\varphi(standby) = \varphi(ToPump) = \varphi(pumped) = CAR \times INTEGER \times INTEGER;$   
 $\varphi(CreditCard) = INTEGER;$   
 $R(ToSchool) = c[2]=1; R(ToGasStation) = c[2]=0; R(Go) = c[2]=1 \wedge g=1;$   
 $R(ToHome) = \lambda; R(Park) = S?st; R(SlideCard) = S!<c[1], cr, st>;$   
 $R(PumpRegular) = S?g \wedge c'[2]=1; R(Deny) = S?cr;$   
 $M_0(home) = \{<sedan, 0>\}; M_0(CreditCard) = \{<2>\}; M_0(school) = \emptyset ;$   
 $M_0(GasStation) = \emptyset; M_0(ToPump) = \emptyset; M_0(standby) = \emptyset; M_0(pumped) = \emptyset;$   
 $F$  and  $L$  are as seen in the Figure 21(c).

### ***The bank***

$P = \{Ready, transaction, Accounts, Report\};$   
 $T_c = \{CheckCredit, ReportCredit\};$   
 $T_u = \{Deny, Authorize\};$   
 $\varphi(Ready) = INTEGER;$   
 $\varphi(transaction) = INTEGER \times CAR \times INTEGER \times INTEGER;$   
 $\varphi(Accounts) = INTEGER \times INTEGER;$   
 $\varphi(Report) = INTEGER \times CAR \times INTEGER \times INTEGER \times INTEGER;$   
 $R(CheckCredit) = S?cr; R(ReportCredit) = S!<cr, r>;$   
 $R(Authorize) = \exists a \in A. (tr[3] = a[1]) \wedge a'[2] = a[2] - 1 \wedge r[1]=t[1] \wedge r[2]=t[2]$   
 $\wedge r[3]=t[3] \wedge r[4]=t[4] \wedge r[5]=1;$   
 $R(Deny) = \forall a \in A. (tr[3] \neq a[1]) \wedge r[1]=t[1] \wedge r[2]=t[2] \wedge r[3]=t[3] \wedge$   
 $r[4]=t[4] \wedge r[5]=0;$   
 $M_0(Ready) = \{<1>\}; M_0(CardNumber) = \emptyset ;$   
 $M_0(Accounts) = \{<1,5>, <2,4>, <3,3>, <4,1>\}; M_0(Report) = \emptyset;$   
 $F$  and  $L$  are as seen in the Figure 21(d).

**A2. Figure 27(c).**

$P_a = \{in\_station, parked, bank\_agent, waiting, gas\_producer\_agent, pumped\};$   
 $P_d = \{pumping\_station, transactions, authorized, diesel\_gas, regular\_gas\};$   
 $T_c = \{Park, Pay, CheckCredit, ReportCredit, PumpDiesel, PumpRegular, Fail, GetDiesel, GetRegular, OrderRegular, OrderDiesel\};$   
 $T_u = \{drive\_in, drive\_out\};$   
 $\varphi(in\_station) = \varphi(parked) = \varphi(waiting) = \varphi(pumped) =$   
 $INTEGER \times CONSUMER\_AGENT;$   
 $\varphi(bank\_agent) = INTEGER \times BANK\_AGENT;$   
 $\varphi(gas\_producer\_agent) = INTEGER \times GAS\_PRODUCER\_AGENT;$   
 $\varphi(transactions) = INTEGER \times CAR \times INTEGER \times INTEGER;$   
// the three integers represent agent id, car type, credit card number and pumping station respectively;  
 $\varphi(authorized) = INTEGER \times CAR \times INTEGER \times INTEGER \times INTEGER;$   
// the four integers represent agent id, car type, credit card number, pumping station and credit report respectively;  
 $\varphi(pumping\_station) = INTEGER;$   
 $\varphi(diesel\_gas) = \varphi(regular\_gas) = INTEGER;$   
 $R(Park) = N!st; R(Pay) = N?<c, cr, st>\wedge tr[1]=a1[1]\wedge tr[2]=c\wedge tr[3]=cr\wedge tr[4]=st;$   
 $R(CheckCredit) = N!tr; R(ReportCredit) = N?r;$   
 $R(PumpDiesel) = a1[1]=r[1]\wedge r[2]=truck\wedge r[5]=1\wedge d'=d-1\wedge g=1\wedge N!g;$   
 $R(PumpRegular) = a1[1]=r[1]\wedge r[2]=sedan\wedge r[5]=1\wedge r'=r-1\wedge g=1\wedge N!g;$   
 $R(Fail) = r[5]=0\wedge N!cr; R(GetDiesel) = N?ds \wedge d' = d + ds;$   
 $R(GetRegular) = N?rs \wedge r' = r + rs;$   
 $R(OrderDiesel) = d < 1 \wedge N!1; R(OrderRegular) = r < 1 \wedge N!2;$   
 $R(drive\_in) = R(drive\_out) = \lambda;$   
 $M_0(in\_station) = \{<1, dieselConsumer>, <2, regularConsumer>\}; M_0(parked) = \emptyset;$   
 $M_0(waiting) = \emptyset; M_0(pumped) = \emptyset; M_0(bank\_agent) = \{<3, bank>\};$   
 $M_0(gas\_producer\_agent) = \{<4, producer>\};$   
 $M_0(pumping\_station) = \{<1>, <2>, <3>\}; M_0(credit\_card) = \emptyset;$

$M_0(diesel\_gas) = \{<10>\}; M_0(regular\_gas) = \{<10>\};$

$F$  and  $L$  are as seen in the Fig. 27(c).

### A3. Figure 43.

#### *The supervisor role*

$\varphi(start) = \varphi(waiting) = NAME \times ACTION \times CATEGORY \times CONTENT;$

$\varphi(result) = CONTENT;$

$R(Read) = e[1] = 'read' \wedge S!e;$

$R(Publish) = e[1] = 'publish' \wedge S!e;$

$R(Assign) = e[1] = 'assign' \wedge S!e;$

$R(Return) = S?r \wedge r[2] = e[1] \wedge x = r[4]$

### A4. Figure 31.

#### *The BCIN net*

$\varphi(Users) = NAME;$

$\varphi(RAs) = \wp(NAME \times ROLE);$

$\varphi(Sessions) = \wp(NAME);$

$\varphi(Actor) = NAME \times ROLE;$

$\varphi(PAs) = \wp(ROLE \times RM);$

$\varphi(Activated) = \varphi(p1) = NAME \times RM;$

$\varphi(BCIN\_Database) = \wp(CATEGORY \times CONTENT);$

$\varphi(p2) = \varphi(p7) = NAME \times ACTION \times CATEGORY \times CONTENT$

$\varphi(p3) = \varphi(p5) = \varphi(p11) = NAME \times RM;$

$\varphi(p4) = \varphi(p8) = \varphi(p6) = \varphi(p9) = \varphi(p12) = \varphi(p10) = NAME \times ACTION \times$   
 $CATEGORY \times CONTENT;$

$R(UserOut) = \exists s \in S. (s = u) \vee \forall r \in R. (r[1] \neq u);$

$R(AssignRole) = \exists r \in R. (r[1] = u) \wedge a = r;$

$R(AssignPA) = \exists p \in P. (p[1] = a[2]) \wedge m[1] = a[1] \wedge m[2] = p[2] \wedge S' = S \cup a[1];$

$R(ActorOut) = \forall p \in P. (p[1] \neq a[2]);$

$R(End) = m[1]?op \wedge op[1] = 'quit' \wedge \exists s \in S. (s = m[1]) \wedge S' = S \setminus s;$

$R(Read) = m[1]?op \wedge r = op;$



$$\begin{aligned}
R(ReadDB) &= (\exists d \in D. (d[1] = r[3]) \wedge r'[4] = d[2]) \vee (\forall d \in D. (d[1] \neq r[3]) \wedge r'[4] = \\
&\quad \text{'error'}); \\
R(returnR) &= m[1] = r'[1] \wedge m[1]!r'; \\
R(SendM) &= m[1]?op \wedge w = op; \\
R(WriteMG) &= D' = D \cup \langle w[3], w[4] \rangle \wedge w'[4] = \text{'message sent'}; \\
R(returnM) &= m[1] = w'[1] \wedge s!w'; \\
R(Publish) &= m[1]?op \wedge p = op; \\
R(PublishRS) &= D' = D \cup \langle p[3], p[4] \rangle \wedge p'[4] = \text{'published'}; \\
R(returnP) &= m[1] = p'[1] \wedge m[1]!p'; \\
R(RA) &= m[1]?op \wedge ra = op; \\
R(UpdateRA) &= (\exists p \in P. (p[1] = ra[4]) \wedge ((\forall r \in R. (r[1] \neq ra[3]) \wedge R' = R \cup \langle ra[3], \\
&\quad ra[4] \rangle \wedge ra'[4] = \text{'added'})) \vee (\exists r \in R. (r[1] = ra[3]) \wedge r'[2] = ra[4])) \wedge ra'[4] = \\
&\quad \text{'updated'})) \vee (\forall p \in P. (p[1] \neq ra[4]) \wedge ra'[4] = \text{'error'}); \\
R(returnRA) &= m[1] = ra'[1] \wedge m[1]!ra';
\end{aligned}$$

## APPENDIX B

### THE PROMELA MODEL FOR BCIN NET

```
#define NULL 0
#define MAX_SESSION 4
#define MAX_ACTIVATED 4
#define MAX_ROLE 4
#define MAX_DB 4
#define MAX_TOKENS 4

mtype = { read, publish, sendM, assign, quit };
mtype = { updated, added, error, message_sent, published}
mtype = { david, john, alice, emily, mary};
mtype = { observer, supervisor, contact, participant };
mtype = { advisory, message, resource };
mtype = { Wilma, generator};
mtype = { O_NET, S_NET, P_NET, PR_NET };

typedef MSG {
    pid agent_id;
    mtype action;
    mtype category;
    mtype content }

typedef ROLE {
    mtype user;
    mtype roleR }

typedef PA {
    mtype roleP;
    mtype net }

typedef IN_SESSION {
    pid sid;
    mtype sname }

typedef BCIN_DB {
    mtype DB_category;
    mtype DB_content }

chan S = [0] of { MSG, chan};

inline initial_marking(pl, p_pid, v1, v2, v3) { pl.agent_id = p_pid;
pl.action = v1; pl.category = v2; pl.content = v3 }
inline add_token(p_in, p_out) { p_out.agent_id = p_in.agent_id;
p_out.action = p_in.action; p_out.category = p_in.category;
p_out.content = p_in.content}
inline remove_token(p) { p.agent_id = 0; p.action = 0; p.category
= 0; p.content = 0 }

active proctype system_net()
{
```

```

chan aid;
MSG m, p12[MAX_TOKENS] = NULL, p2[MAX_TOKENS] = NULL,
p4[MAX_TOKENS] = NULL, p6[MAX_TOKENS] = NULL;
byte cnt, act_idx = 0, p12_idx = 0, p2_idx = 0, p4_idx = 0, p6_idx
= 0;

ROLE actor, ra[MAX_ROLE] = NULL;
IN_SESSION session[MAX_SESSION] = NULL;
ra[0].user = david; ra[0].roleR = observer;
ra[1].user = alice; ra[1].roleR = supervisor;
ra[2].user = emily; ra[2].roleR = contact;
byte ra_idx = 3;

PA pa[MAX_SESSION];
pa[0].roleP = observer; pa[0].net = O_NET;
pa[1].roleP = supervisor; pa[1].net = S_NET;
pa[2].roleP = contact; pa[2].net = P_NET;
pa[3].roleP = participant; pa[3].net = PR_NET;

BCIN_DB db[MAX_DB] = NULL;
db[0].DB_category = advisory; db[0].DB_content = Wilma;
db[1].DB_category = message; db[1].DB_content = generator;
byte db_idx = 2;
bool inSession = false;
mtype valid_user = NULL;

/* RBAC Begin */
mtype user_in = emily;

do
    /* UserOut */
    :: atomic { user_in != NULL -> cnt = 0;
do
    :: cnt < MAX_SESSION ->
        if :: session[cnt].sname == user_in -> printf("User %e in
session !", user_in); user_in = NULL; inSession = true; break

        :: session[cnt].sname != user_in -> cnt++
        fi
    :: cnt >= MAX_SESSION -> cnt = 0; inSession = false; break
od;
do
    :: inSession == false ->
        if :: cnt < MAX_ROLE && ra[cnt].user == user_in -> valid_user
            = user_in; user_in = NULL; break
        :: cnt < MAX_ROLE && ra[cnt].user != user_in -> cnt++
        :: cnt >= MAX_ROLE -> printf("No role assignment for %e
!",
            user_in); user_in = NULL; break
        fi;
od }

/* Assign Role */
:: atomic { valid_user != NULL -> cnt = 0;
do

```

```

:: cnt < MAX_ROLE ->
    if :: ra[cnt].user == valid_user -> actor.user =
        ra[cnt].user; actor.roleR = ra[cnt].roleR; valid_user =
        NULL; break
    :: ra[cnt].user != valid_user -> cnt++
fi;
:: cnt >= MAX_ROLE -> break
od }
/* ActorOut */
:: atomic { actor.user != NULL -> cnt = 0;
    do
        :: cnt < MAX_SESSION && pa[cnt].roleP != actor.roleR ->
            cnt++
        :: cnt < MAX_SESSION && pa[cnt].roleP == actor.roleR ->
/* Assign PA */
        if :: actor.roleR == supervisor -> session[act_idx].sid =
            run agentS(assign, john, participant)
        :: actor.roleR == observer -> session[act_idx].sid = run
            agentO(read, advisory, Wilma)
        :: actor.roleR == contact -> session[act_idx].sid = run
            agentP(sendM, message, generator)
        :: actor.roleR == participant -> session[act_idx].sid =
            run agentPR(publish, resource, generator)
        fi;
        session[act_idx].sname = actor.user; act_idx++; actor.user
        = NULL; actor.roleR = NULL;
        actor.user = NULL; actor.roleR = NULL; break
        :: cnt >= MAX_SESSION -> printf("Role %e not available !",
            actor.roleR); break
    od }

/* RBAC End */

/* Main Operation */
:: atomic { S?m(aid) ->
    /* Assign RA */
    if :: m.action == assign ->
        if :: p12_idx < MAX_TOKENS -> add_token(m, p12[p12_idx]);
            p12_idx++
        :: p12_idx >= MAX_TOKENS -> printf("Exceed maximum
            token deposits!")
        fi;
    /* UdateRA */
    cnt = 0; p12_idx--;
    do :: cnt < MAX_SESSION && pa[cnt].roleP ==
        p12[p12_idx].content -> cnt = 0; break
        :: cnt < MAX_SESSION && pa[cnt].roleP !=
        p12[p12_idx].content -> cnt++
        :: cnt >= MAX_SESSION -> printf("Role %e denied !",
            p12[p12_idx].content); p12[p12_idx].content = error
    od;
    do :: cnt < MAX_ROLE && ra[cnt].user ==
        p12[p12_idx].category -> ra[ra_idx].roleR =
        p12[p12_idx].content; p12[p12_idx].content =
        updated; break

```

```

:: cnt < MAX_ROLE && ra[cnt].user !=
    p12[p12_idx].category -> cnt++
:: cnt >= MAX_ROLE && ra_idx < MAX_ROLE ->
    ra[ra_idx].user = p12[p12_idx].category;
    ra[ra_idx].roleR = p12[p12_idx].content;
    p12[p12_idx].content = added; ra_idx++; break
:: ra_idx >= MAX_ROLE -> printf("Exceed maximum role");
    p12[p12_idx].content = error; break
od;
/* returnRA */
    aid!p12[p12_idx] ; remove_token(p12[p12_idx])
/* read */
    :: m.action == read -> user_in = alice;
    if :: p2_idx < MAX_TOKENS -> add_token(m,
p2[p2_idx]);
        p2_idx++
        :: p2_idx >= MAX_TOKENS -> printf("Exceed maximum
            token deposits!")
    fi;
/* readDB */
    cnt = 0; p2_idx--;
    do :: cnt < MAX_DB && db[cnt].DB_category ==
        p2[p2_idx].category -> p2[p2_idx].content =
            db[cnt].DB_content; break
        :: cnt < MAX_DB && db[cnt].DB_category !=
            p2[p2_idx].category -> cnt++
        :: cnt >= MAX_DB -> p2[p2_idx].content = error;
            break
    od;
/* returnR */
    aid!p2[p2_idx]; remove_token(p2[p2_idx])
/* sendM */
    :: m.action == sendM ->
    if :: p4_idx < MAX_TOKENS -> add_token(m, p4[p4_idx]);
        p4_idx++
        :: p4_idx >= MAX_TOKENS -> printf("Exceed maximum
            token deposits!")
    fi;
/* WriteMG */
p4_idx--;
    if :: db_idx < MAX_DB -> db[db_idx].DB_category =
        p4[p4_idx].category; db[db_idx].DB_content =
        p4[p4_idx].content; p4[p4_idx].content =
        message_sent
        :: db_idx >= MAX_DB -> printf("DB full !!");
        p4[p4_idx].content = error
    fi;
/* returnM */
    aid!p4[p4_idx]; remove_token(p4[p4_idx]); user_in = david
/* publish */
:: m.action == publish ->
    if :: p6_idx < MAX_TOKENS -> add_token(m, p6[p6_idx]);
        p6_idx++
        :: p6_idx >= MAX_TOKENS -> printf("Exceed maximum
            token deposits!")

```

```

        fi;
        /* publishRS */
        p6_idx--;
        if :: db_idx < MAX_DB ->
            db[db_idx].DB_category = p6[p6_idx].category;
            db[db_idx].DB_content = p6[p6_idx].content;
            p6[p6_idx].content = published
        :: db_idx >= MAX_DB -> printf("DB full !!");
            p6[p6_idx].content = error
        fi;
        /* returnP */
        aid!p6[p6_idx]; remove_token(p6[p6_idx])
        /* End */
    :: m.action == quit ->
        cnt = 0;
        do :: cnt < MAX_SESSION && session[cnt].sid ==
            m.agent_id -> act_idx--; session[cnt].sid =
            session[act_idx].sid; session[cnt].sname =
            session[act_idx].sname; session[act_idx].sid =
            NULL; session[act_idx].sname = NULL; break
        :: cnt < MAX_SESSION && session[cnt].sid !=
            m.agent_id --> cnt++
        :: cnt >= MAX_SESSION -> break
    od;
    fi }
od
}

proctype agentS( mtype i1, i2, i3) /* Supervisor S_NET */
{
    chan me = [0] of { MSG };
    MSG m, start, waiting, result;

    initial_marking(start, _pid, i1, i2, i3);
    S!start(me);
    add_token(start, waiting);
    remove_token(start);
    me?m;
    printf("operation %e %e %e", m.action, m.category, m.content);
    m.action = quit;
    S!m(me)
}

proctype agentO(mtype i1, i2, i3) /* Observer O_NET */
{
    chan me = [0] of { MSG };
    MSG m, start, waiting, result;

    initial_marking(start, _pid, i1, i2, i3);
    S!start(me);
    add_token(start, waiting);
    remove_token(start);
    me?m;
    printf("operaton %e successful!", m.action);
    m.action = quit;
    S!m(me)
}

```

```

proctype agentP(mtype i1, i2, i3)    /* Primary Contact P_NET */
{
    chan me = [0] of { MSG };
    MSG m, start, waiting, result;

    initial_marking(start, _pid, i1, i2, i3);
    S!start(me);
    add_token(start, waiting);
    remove_token(start);
    me?m;
    printf("operaton %e successful!", m.action);
    m.action = quit;
    S!m(me)
}

proctype agentPR(mtype i1, i2, i3)    /* Participant PR_NET */
{
    chan me = [0] of { MSG };
    MSG m, start, waiting, result;

    initial_marking(start, _pid, i1, i2, i3);
    S!start(me);
    add_token(start, waiting);
    remove_token(start);
    me?m;
    printf("operaton %e successful!", m.action);
    m.action = quit;
    S!m(me)
}

```

## VITA

### LILY CHANG

1977 — 1982	Department of Accounting and Statistics The Overseas Chinese Institute of Technology Taichung, Taiwan
1982 — 1986	Banker, Consumer and Small Business Banking The 7 <sup>th</sup> Commercial Bank (Now: Cathay United Bank) Taichung, Taiwan
1986 — 1988	M.S., College of Computing Sciences New Jersey Institute of Technology Newark, New Jersey
1988 — 1991	Senior Software Engineer, Department of Information Systems Tatung Co. Taipei, Taiwan
1997 — 1999	Director, Computing and Network Center Taipei College of Maritime Technology Taipei, Taiwan
1991 — 2005	Instructor, Computing and Network Center Taipei College of Maritime Technology Taipei, Taiwan
2005 — 2011	Doctoral Candidate in Computer Science Florida International University Miami, Florida

## PUBLICATIONS AND PRESENTATIONS

L. Chang, X. He and S. Shatz: “A Methodology for Modeling Multi-Agent Systems using Nested Petri Nets”, International Journal of Software Engineering and Knowledge Engineering, 2011.

L. Chang and X. He: "A Model Transformation Approach for Verifying Multi-Agent Systems", Proceedings of the Symposium on Applied Computing (SAC2011), Taichung, Taiwan, March, 2011.



L. Chang and X. He: "A Multi-Agent Model for a Business Continuity Information Network", Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE10), San Francisco, July 2010, pp. 657-663.

L. Chang and X. He: "Towards Adaptable BDI Agent: a Formal Aspect-Oriented Modeling Approach", Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE09), Boston, July, 2009, pp. 189-193.

L. Chang, J. Ding, X. He, S. Shatz: "A Formal Approach for Modeling Software Agents Coordination". Communication of SIWN, Vol. 3, 2008, pp. 58-64.

L. Chang, X. He, J. Lian, and S. Shatz: "Applying a Nested Petri Net Modeling Paradigm to Coordination of Sensor Networks with Mobile Agents", Proc. of Workshop on Petri Nets and Distributed Systems 2008, Xian, China, June, 2008, 132-145.

B. Wongsaraj, S. Graham, O. Wolfson, R. Steinhoff, A. Cary, L. Chang, A. Lee, A. Rodriguez, P. Singh, R. Haynes, T. Rush, A. Barreto, M. Adjouadi, N. Rish, "Native XML Database Management", Proceedings of The 3rd International Conference on Cybernetics and Information Technologies, Systems and Applications, CITSA, Orlando, FL, July, 2006.