Goal-oriented Modeling for Intelligent Agents and their Applications

Shen Zhiqi

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2005

Acknowledgement

I would like to express my deepest thanks to my supervisor, my mentor, Professor Robert Gay, for providing the guidance, enthusiastic encouragement and invaluable advice throughout the course of the research. At many stages of this research I benefited from his advice, particularly so when exploring new ideas. His encouragement in my research always inspires me to continue more in-depth research. His indispensable help contributed enormously to the production of this thesis.

I wish to express appreciation to my previous research director, Dr. Low Hwee Boon and other colleagues at I^2R (Institute for InfoComm Research) Singapore, for their support and encouragement during my PhD research. I would also like to thank the research opportunity offered by MDSI, Canada, from which I found that software agent is becoming pervasive in industry.

I would like to acknowledge the research opportunity granted by Information Communication Institute, School of EEE of NTU. I also wish to thank the Director and other colleagues from Managed Computing Competency Centre of School of EEE, NTU, for providing me various facility supports to carry out this research. Their help are gratefully acknowledged.

Special thanks to Dr. Zhang Sijian and Dr. Hong Tao who provided me valuable comments on earlier versions of this thesis; and to Dr. Li Xiang from SIMTech who

offered me a lot of information in one of my case studies and shared with me her research experiences.

I would like to take this opportunity to express my sincere appreciation to my wife for her love and support. One of the best experiences that we lived through in this period was the birth of our son Kevin and our daughter Karen, who provided an additional and joyful dimension to our life mission. Especially, I would like to give my heartfelt thanks to my parents, parents in law, my brothers and sisters whose silent love enabled me to complete this research work.

Last but not least, I would also like to thank all my friends who have individually helped me in one way or the other throughout these years.

Table of Contents

List of Figures

List of Tables

Summary

Agents are being advocated as a next generation technology for engineering complex, distributed systems. It is predicated that in 10 years time, most new software and developments will contain embedded agent systems. In view of multi-agent system as a *new software engineering paradigm*, in this research we study agents and multi-agent systems from a software engineering perspective. Despite the significant progress in the field of agent research and development, to date, there is still a lack of widespread development and deployment of agent systems and multi-agent systems. Agents are goal-oriented, which necessitates a shift in modelling paradigm, from object-oriented modeling to goal-oriented modeling. Given this new landscape, this thesis presents a novel goal-oriented (GO) modeling approach for modeling the complex goals of agents and for engineering agent-based systems in various application domains.

There have been many research efforts on characterizing agents, but little work has been done for characterizing the goal of agents. To model the complex goals of agents, a characterization of an agent's goal with different properties such as composite goal, fuzzy goal, partial goal, sub goal, atomic goal, etc. is given in the thesis. The goal characterization is an important part for conceptual modeling, analysis and evaluation of the goals of agents. To represent various properties of agent's goal and to model the dynamic goal relationships in a dynamic goal pursuing process, a goal-oriented agent model method, namely Goat Net, is proposed. Goal Net not only enables the agent to

manage the composite goals but also incorporates flexible action selection strategies for goal pursuing as well as provides the quantitative measurement of goal achievements.

As an agent goal model, Goal Net enables the agents to present both behavior autonomy and goal autonomy. The autonomy currently used in agent literature is referable to the behavior autonomy (as opposed to the goal autonomy) with the assumption that the goal of an agent is implicitly defined in agent behaviors. In a dynamic changing environment, agents must be able to autonomously decide their future goals. Goal Net facilitates the dynamic goal/action selection through anytime algorithms and various learning/reasoning mechanisms.

In addition to an agent goal model, Goal Net also serves as a goal-oriented requirement and modeling tool, and a multi-agent identification, organization and coordination model. As a new software-modeling tool, Goal Net assists in all the phases of the life cycle for development of agent-based applications and supports decomposition and re-combination.

A multi-agent system development framework, namely, Multi-agent Development Environment (MADE) that supports GO methodology is developed. MADE bridges the gap between agent mental models and agent implementation; facilitates the integration with other services; and highly improves the knowledge reuse and code reuse in a wide variety of agent-oriented development environments.

Case studies on agent-oriented e-forecasting and agent-oriented e-learning in a grid environment are conducted and reported in this thesis, which show that the proposed goal-oriented approach is not only promising but also practical.

Nomenclature

AI – Artificial Intelligence

AOBF – Agent-oriented Business Forecasting

AOSE – Agent-oriented Software Engineering

FNN – Fuzzy Neural Network

GO – Goal-oriented

IBF – Intelligent Business Forecasting

KQML – Knowledge Query Manipulation Language

MADE – Multi-agent Development Environment

MAS – Multi-agent System

OO – Object-oriented

Agent – a software entity that has the capability to act towards its goal autonomously in the specific or even unknown environment on behalf of users, other software or other agents

Goal – a desired state that an agent tries to reach

Goal Net – proposed composite state goal model

Behavior autonomy – the ability of autonomous action selection

Goal autonomy – the ability of autonomous goal selection

CHAPTER 1

INTRODUCTION

1.1 Motivation

There have been significant changes in the environment of software systems as well as user's expectations towards software systems in recent years. With the emergence of Internet and WEB/GRID technology [Foster, 2002], a large majority of software is moving to open, distributed, decentralized and integrated environments. Moreover, with the widespread of handheld computing devices, there is an increased demand for provision of various software services in a pervasive environment. Meanwhile, users expect to access various services (e-commerce, e-learning, e-banking, e-healthcare, e-forecasting, etc.) from anywhere, at any time and in any form [Guttman, 98].

Taking e-learning as a typical example of an application domain, dramatic changes have been made to e-learning systems. Huge investments are being put in e-learning world-wide in response to the skills shortage in a rapidly changing economy. The new

knowledge-based economy and increased demand for various educational/knowledge upgrade drive a movement from traditional training/education to providing personalized e-learning services "at anytime, from anywhere and in any form" [Silveira, 2002].

To adapt to the big changes of environment as well as to satisfy the increased expectations from the users, a software system needs to present some new characteristics [Zambonelli, 2003], such as: 1) it is always running, i.e., it runs 24 hours a day and 7 days a week (24/7), therefore, it could not be turned off for repairing or maintenance in a traditional way; 2) it is able to adapt to the dynamic changing environment; 3) it allows new software components to be plugged in, new services to be composed "on the fly"; 4) it is highly interactive, it must be ready to interact with users and help the users to achieve their objectives from anywhere, at any time and in any forms; 5) it should be light-weight for residing in a pervasive environment and for decomposing computing complexity; 6) it needs to be personalized, as different user may access the services from different environments with different objectives. For example, most of the current e-learning systems focus on information transmission and treat each learner equally. While it is useful, it is far from the individual user's own expectations, as different users have different learning goals with different knowledge skills and might learn in different learning environments.

With such a background, agent technology, especially multi-agent system (MAS) has received a great deal of attention from both research and industry in the last few years [Shen, 99; Kendall, 98a; Durfee, 96; Wooldridge, 99]. An agent is an autonomous software entity, which acts towards its goal on behalf of others. Agents have some characteristics such as being autonomous, pro-active, goal-oriented, adaptive, communicative, decentralized etc. that satisfy the new requirements for the software

systems. For instance, an agent can work 24/7. It is able to adapt itself in a changing environment. A multi-agent system that consists of interactive agents represents a new software engineering paradigm. A set of research papers has predicted that agent technology will have a major impact on future generations of software. [Kotz, 99; Faltings, 2000; Maes, 98; Kalakoka, 2000]. With the fast growing number of agent applications in various domains, the agent-oriented modeling approach has become significant for development of next generation of software applications [Zambonelli, 2003].

However, despite the significant progress in the field of agent research and development, to date, there is still a lack of widespread development and deployment of agent systems and multi-agent systems [Jennings, 98; Jennings, 2000; Jennings, 2001]. An important reason is that most of existing research efforts on agents are focused on studying agents from an AI (Artificial Intelligence) perspective (learning, reasoning, emotion, coordination etc.), or DAI (Distributed Artificial Intelligence) perspective (decentralization, mobility etc.). In view of multi-agent system as a ***new software engineering paradigm***, in this research we study agents and multi-agent system from a software engineering perspective and propose a novel goal-oriented method for engineering multi-agent systems in various application domains.

1.2 A Software Engineering Perspective of Agents

Research on agent technology can be traced back to the 1970s. At that time, agents mainly belonged to the domain of artificial intelligence (AI). The use of agents became frequent in the 1980s, especially in various expert systems. Since 1994, agent technology

has become one of the most important research areas in mainstream computer science [Genesereth, 94; Maes, 94; Franklin, 96; Nwana, 99]. The range of agent types being investigated has become much broader due to integration with other technologies e.g. distributed AI, embedded technology [Nwana, 99; Serrano, 2002]. It has been forecast that in 10 years time most new software/hardware developments will contain embedded agent-based systems [Jennings, 2000].

Looking back on the evolution of software modeling approaches, the *first generation of software* consisted of a single program, which was modeled according to the individual programmer's own logic. James Odell summarized it as Monolithic Modeling and Programming [Odell, 2002, Wooldridge, 2002]. As software became more and more complex, structured/modular modeling emerged in the 1970s for developing the *second generation of software*. Structured modeling decomposes a single complex program into smaller units, such as functions or modules [Demarco, 78]. With modular modeling, the complexity is reduced. However, each module has no control to itself. External entities change the states and invoke the modules freely.

Object-oriented (OO) modeling [Rentsch, 82] was proposed in the 1980s, and has replaced structured modeling as the *third generation of software* [Mellor, 97]. OO decomposes the software system into objects, which demonstrate encapsulation, inheritance and polymorphism. With encapsulation, an object has the control to its internal properties. Nevertheless, it presents simple reactive behavior with its methods invoked by messages sent from others.

Recently, there has been a set of research papers addressing the inevitable move from *object-oriented paradigm* into *agent-oriented paradigm* [Faltings, 2000; Odell, 99;

Jennings, 2000; Jennings, 2001]. Although there are some similarities between objects and agents, there are fundamental differences that differentiate an agent from an object. A number of papers have addressed the significant differences between agents and objects [Odell, 99; Bauer, 2001; Carie, 2001; Wooldridge, 2000]. One of the most important differences between agents and objects is that an agent is proactive. It has its own goals to achieve. An agent works towards its goals. It perceives the environment changes, and takes proper actions autonomously according to its goals. Jennings summarized that an agent does things for "money" while objects do things for "free" [Jennings, 98].

With goals, an agent gains control not only to its property/state but also to its behaviours. In terms of organization, objects live in a centralized control organizational environment while agents live in a de-centralized organizational environment. In terms of dependency to the system, an object is highly coupled to the system, missing of an object may cause various exceptions, while an agent is autonomic and less coupled to a system. Other differences in nature between agents and objects can be referred to in related research papers [Bauer, 2001; Carie, 2001; Wooldridge, 2000].

Consisting of interactive agents, a new type of autonomic software entities that have the control not only of its property/state but also of its behavior towards its goal, multi-agent systems (MAS) provide a new software engineering approach for modeling, designing and implementation of software systems with new characteristics suited for the dynamic changing environment.

1.3 Research Challenges

Given this new landscape, there is a strong need for proposing agent-oriented modeling methodologies to assist in all the phases of the life cycle for the development of future generations of software. Most of the current efforts on agent development are still to employ object-oriented methodologies. An agent is modelled as an extended object or so-called "smart object" [Amandi, 97; Guessoum, 98; Odell, 99, OMG, 99; OMG, 2000; Hongsoon, 2000]. Recently, a few agent-oriented software engineering approaches have been proposed [Woodridge, 2001; Caire, 2002; Bresciani, 2001; Kolp, 2002]. Compared with traditional software engineering approaches (i.e. OO), new research challenges have been introduced by agent-oriented paradigm:

An Agent is goal-oriented, which necessitates a shift in modelling paradigm [Kim, 2000; Kolp, 2001; Lamsweerde, 2001; Liu, 2002]. The modelling method for agent should start from the concept of goal rather than limit an agent as an extended object. Obviously, goal modeling is one of the most important aspects in successful agent development. Therefore new types of goal-oriented modeling methodologies are required for modeling goals of agents in complex dynamic environments and for incorporating flexible learning/reasoning mechanisms to reach the goals.

An autonomous agent should present not only behavior autonomy (the ability of autonomous action selection) but also goal autonomy (the ability of autonomous goal selection) [Bonifacio, 2002]. Castelfranchi remarks that the definition of autonomy currently used in agent literature is referable to the behavior autonomy (as opposed to goal autonomy) [Bonifacio, 2002; Castelfranchi, 94]. As a consequence, this form of rationality produces rational behavior only if the environment is stable, changes in a

predictable way, and agents have complete knowledge about it. Otherwise, it may happen that agents, with no control on their goals, can irrationally pursue unrealistic goals or evaluate goals on the basis of unrealistic preferences. To overcome these limitations, an agent must be *goal autonomous*, namely it must have the possibility to decide not only how to achieve a goal, but also which goal to select. As Castelfranchi and Bonifacio underlined, a strong notion of an agent is that: an agent, if not goal autonomous, is not autonomous at all. Now the question becomes how such strong notion of agent autonomy can emerge in real world agent systems. Little research has been reported for addressing this issue.

Despite the significant progress in the field of agent research and development, to date, there is still a lack of widespread development and deployment of agent systems and multi-agent systems [Jennings, 98; Jennings, 2000; Jennings, 2001]. Agent technology provides many potential advantages for future generations of software. Nevertheless, there have been many challenges in design and implementation of agent systems and multi-agent systems [Susan, 97; Ndumu, 97; Jennings, 98; Sycara, 98; Nwana, 99; Wooldridge, 2000b; Wood, 2001]. Some of the major issues that will be tackled in this research are listed as follows:

- There are *gaps between requirements to agent design*, as well as *gaps between agent design and implementations*.

- Most of the current agent-oriented software engineering (AOSE) methodologies focus on the entity relationships such as goal and role, and there is lack of the modeling of the control to autonomous agents' behavior by the goals.

- There is a *gap between agent mental models and agent implementation*. A lot of effort has been put into research on agent mental models. Few efforts have been put in linking the agent mental models to the agent implementations.

- There is a need to support the *flexible integrations* with other systems including traditional systems. Software is becoming more and more integrated. The emergence of web services, grid computing, and peer to peer network makes it possible to link worldwide applications and integrate them with each other [Rana, 2000; Foster, 2002]. Every system might become a part of other systems. To our best knowledge, none of the current AOSE methodologies addressed the integration issues with traditional systems.

- There is a need to support the *decomposition and combination* throughout the life cycle of agent-oriented system development, including requirements, design, and implementation. The need of worldwide service integration results in the fact that no one has control over the whole system. We are now moving to a world where software systems need to be easily decomposed, modified and re-composed/re-combined "on the fly" by different people who are not under the control of the designers of the systems.

In order to apply AOSE paradigm in real world agent systems, practical modeling method, and AOSE methodologies for bridging the above gaps in designing and implementation of agent-oriented system are highly needed. In such a background, this research project aims to address some of the above issues. The objectives of this research are as follows:

- To explore a goal-oriented modeling method that enables agents to present not only execution autonomy but also goal autonomy by incorporating flexible learning/reasoning mechanisms.

- To explore a goal-oriented modeling method that bridges the gap between the agent mental models and agent implementations.

- To explore a goal-oriented modeling method that enables the decomposition and combination/integration in model, design and implementation of agent-oriented system.

- To propose a practical methodology for modeling, designing and implementation of multi-agent systems in various domains and for building the connections between agent design and requirement, and between design and implementations.

- To demonstrate the practice of the proposed methodology through the case studies in selected domains including business forecasting, e-learning, and grid computing.

1.4 Contribution

The main contributions of this research include:

1) A novel goal-oriented (GO) methodology for engineering agent-oriented systems for various application domains, which covers the whole life cycle from goal-oriented requirement analysis, goal-oriented agent/multi-agent modeling, to design and implementation of agent/multi-agent systems.

2) An original goal-oriented modeling mechanism, Goal Net, for modeling and architecting MASs. Goal Net is used for goal-oriented requirement analysis, and for modeling, design, and implementation of agent-oriented systems. Goal Net also acts as a multi-agent modeling, identification, organization and coordination tool.

3) a goal-oriented Multi-Agent Development Environment (MADE) for goal autonomous agent development which has integrated with the popular JADE, making MADE more powerful.

The proposed Goal Net, GO methodology, and MADE bridge the gaps between agent mental models and agent implementations as well as the gaps between requirement and design, and between design and implementation. The proposed methodology highly improves the openness, reusability, composition ability and integration ability of agent development towards a widespread and rapid development of agent-oriented applications.

1.5 Thesis Organization

Following this introduction, Chapter 2 presents the theoretical background and related work. Chapter 3 describes the proposed goal-oriented modeling method, Goal Net. Chapter 4 presents an agent model based on Goal Net and a Multi-agent Development Environment (MADE) for agent development. Chapter 5 summarizes a practical agent-oriented methodology for engineering agent-oriented systems. Case studies are explored in selected application domains in Chapter 6, and Chapter 7. Finally Chapter 8 discusses future work and concludes the thesis.

CHAPTER 2

THOERETICAL BACKGROUND AND RELATED WORK

This chapter reviews agent definition, agent modeling, agent architecture and agent-oriented software engineering (AOSE) methodologies for modeling, design and implementation of agent-oriented systems. It consists of five sections. Section 2.1 introduces basic concepts of software agents and multi-agent systems. Section 2.2 reviews related work in goal-oriented agent modeling approaches. Section 2.3 discusses the agent architectures. Section 2.4 enumerates the existing AOSE methodologies. The chapter is summarized in Section 2.5.

2.1 Software Agents and Multi-agent Systems

An agent is defined in the dictionary [Webster, 2000] as "one who acts for another". But in the field of computer science, agent stands for software agent, which is an agent

implemented in software [Huhns, 98]. To date, there have been many definitions of agents. However, none of the definitions have won overwhelming acceptance. Some of the notable definitions are listed below:

> "Software agent is a more specific type of agent while there are many definitions. Generally, a software agent is an *autonomous* software entity that can interact with its environment." [AWG, 99]

> "An agent is a software thing that knows how to do things that you could probably do yourself if you had the time." [Janca, 95]

> "Agent is a piece of software which performs a given task using information gleaned from its environment to act in a suitable manner so as to complete the task successfully. The software should be able to adapt itself based on changes occurring in its environment, so that a change in circumstances will still yield the intended result." [Hermans, 96]

> "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." [Franklin, 96]

Instead of a universal definition, a number of agent characteristics classified by Wooldridge and Jennings have been widely accepted:

> "Agent is a hardware or (more usually) software-based computer system that enjoys the following properties:

- goal orientation and pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative;

- autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;

- social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;

- reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it." [Wooldridge, 95]

An agent is not necessarily intelligent. However, agents with intelligence, which are so called intelligent agents, are considered as one of the most important types of agents. Intelligence can be categorized into two contexts, which are behavioral and structural. In the behavioral context, intelligence can be considered a capacity of a system to execute a set of actions that are recognized in humans as the necessary symptoms of "intelligence". In the structural context, intelligence is a property of both the function architecture and the reasoning mechanism of an abstract system. A system's complex internal processes cause numerous, external observed intelligent behaviors [Gadomski, 93].

The intelligence of an agent mainly consists of three aspects: knowledge representation, the ability to learn and the ability to infer. A lot of effort has been devoted to research in the area of agent knowledge representation, agent learning and agent reasoning mechanisms [Wooldridge, 95; Kendall, 98b; Nwana, 99; Ndumu, 97; Carmel, 96]. The intelligence of an agent is not isolated from other characteristics of an agent. For example, agents are goal-oriented. It is the intelligence that empowers an agent to decide the proper actions it needs to take in a changing environment so as to achieve its goal autonomously.

In this research, we define an agent as "a software entity that has the capability to act towards its goal autonomously in the specific or even unknown environment on behalf of users, other software or other agents". It has the following basic properties:

- *autonomous* – acts without human intervention and has some kind of control over its actions and internal states;
- *goal-oriented/pro-active* – acts towards its goals;
- *social* – communicates with other agents or users via structured messages;
- *reactive* – responds to environment changes;
- *intelligent* – has the knowledge about its goals and behaviors and possibly has reasoning and/or learning capabilities.

Among these properties, the goal-oriented property is regarded as one of the most important attributes of an agent, which distinguishes itself from an object and other software components

The capacity of a single agent is limited by its knowledge, its computing resources and its perspective. In order to solve complex problems multi-agent systems (MASs), in which multiple agents work cooperatively in a heterogeneous environment towards a common goal, have emerged.

A multi-agent system consists of a number of decentralized, autonomous agents. The characteristics of a MAS are that [Sycara, 98; Stone, 2000; Hongsoon, 2000; Wood, 2001]:

1) Each agent has incomplete information and capabilities for solving the problem, and thus, has a limited viewpoint;

2) Each agent has its own goal and meanwhile shares a common goal with other agents in a multi-agent system,

3) There is no system global control, each agent acts towards it's own goal,

4) Agents in a multi-agent system are decentralized; and computation is asynchronous.

Although each agent has its own goals to solve a particular problem aspect in a MAS environment, agents in a MAS perceive, reason and act by collaborating with each other towards a common goal. The goal modeling is one of the most important issues in MAS.

As introduced in Chapter 1, systems that consist of interacting agents i.e. multi-agent systems represent a new software engineering paradigm for engineering modern software systems to meet the increasing requirements/expectations in various application domains.

In the following sections, we review some related research in the area of agent-oriented software engineering.

2.2 Goal-oriented Agent Modeling

Goal orientation has become a recognized paradigm for eliciting, modeling and analyzing agent-based systems [Kim, 2000; Kolp, 2001; Liu, 2002]. Agents are driven by a set of goals. In order to achieve those goals agents need to undertake actions.

Goal modeling is an important issue to agent autonomy. However, most of the existing research work focuses on agent mental models such as agent inference models and agent learning models with the assumption that the goal of an agent is implicitly defined in the agent's behaviors. Recently, there have been increasing efforts in addressing goal-modeling issues [Park, 2000; Yu, 2001].

2.2.1 What are Goals?

There is an enormous amount of disparate literature in psychology and AI that is potentially relevant to the topic of goal definition and processing. Many articles have been published on the topics of motivation, emotion, "self-regulation", and attention related to the term "goal" [Boden, 72; Simon, 67; Sloman, 78; Sloman, 81].

In the AI literature, goal has been examined, but usually does not use the terms: motivation, emotion, self-regulation, and attention. In this thesis, we briefly discuss the well known "goal theory" of [Lee, 89].

Goal theory is supposed to provide a "specification of goal processes". The theory emphasizes the positive effect on the performance of individual "setting" specific and difficult goals. The main assumptions are that:

- the content of a goal determines the behavior towards the goal, and this behavior in turn impacts on performance;
- there are various causal relations between goal and behaviors.

Goals have four components:

- The *goal level* is the difficulty of the state to be achieved. For instance, a student might aim to be in the top 5th or 10th percentile—the higher the percentile, the higher the goal level.
- There is the degree of quantitative *specificity* of the goal.
- There is the *complexity* of the goal; by this they mean the number of sub-goals that are required to satisfy it.
- There is the *conflict* between target goals and other goals. Goal level and goal specificity are assumed to combine additively to affect the behavior profile.

The behavior profile comprises direction of effort. This simply means that behavior is selectively directed towards the goal. There is a quantitative dimension of amount of *effort*, and one of *persistence* in the face of external difficulties. And "task strategy" represents the plans that are used to execute the task [Locke, 90]. Characterizing goal theoretic research is useful in order to put the proposed agent goal model in a right and

distinctive context. Goal theory underscores a number of variables that need to be considered in agent goal modeling.

In simple terms, a goal can be viewed as an end situation or a future state a system or a process should achieve. Vertically, goals may be categorized to different levels of abstraction: higher level goals and lower level goals. Higher level goals can be divided to lower level goals or called sub-goals. For example, "*do business forecasting*" is a higher level goal of a business forecasting system. It can be divided to lower level goals, such as "*collect data*", "*prepare data*", "*train the model*", and "*compute forecasting results*". Obviously, the lower level goals are used to guarantee the achievement of the higher level goals. Higher level goals and lower level goals can then form a goal hierarchical structure. Higher level goals and lower level goals can be involved in one goal hierarchy. Horizontally, goals can also be differentiated by types of concerns: functional concerns, associated with the functionalities agents provide, and nonfunctional concerns, associated with quantative measurement of goal achievement such as satisfaction, accuracy, performance, etc.

An agent has goals, on whose fulfillment the agent would prefer to act. Goals help the agent determine what actions to take under particular circumstances. As mentioned above, there are causal relationships between goals and behaviors. An example relationship between goals is that goals can have sub-goals. Agent behaviors (actions) are driven by their goals. An example relationship between a goal and actions is that a set of actions can be associated with a goal. So, an agent goal model needs to address the structure, the interaction of goals, and various types of associations between goals and behaviors.

2.2.2 Agent Goal Models

Compared with research in the intelligence of agents, research on the goal-oriented modeling is relatively weak. The best-known agent goal models are the task-oriented goal model and the state-oriented goal model [Rosenschein, 94].

Traditionally, there have been mainly two types of agent goal models, task-oriented model and state-oriented model [Rosenschein, 94].

Task-oriented model assumes that agents live in a task-oriented domain, the goal of an agent is a set of tasks to perform. For instance, a notification agent has a goal to notify the customers of new products. The agent's goal is specified as a set of tasks:

1) Check if any new products arrive;

2) For each new product, find the customers who might be interested in the new product from customer profile database;

3) Notify the customers by sending an email with the new product information to every corresponding customer.

Hence the task-oriented goal is a fixed list of tasks; the goal is reached only when the agent finishes all the tasks, regardless of the state changes caused by the tasks. Then it will iterate the process.

State-oriented model assumes agent lives in the state-oriented domain. The agent's environment is evolved with a sequential finite set of the states. A goal of the agent is a final state that the agent tries to reach from its current state by going through a sequence of states. For example, the goal of a manufacturing design agent is to reach the final state

by going through a sequence of states specified by a workflow for modifying a design file. The initial state of the design agent is *waiting for an order*, after receiving an order, the agent will *check out* the design file, *lock* the design file, *modify* the file, *check in* the file after modification, and *unlock* the design file, to reach the states showing in Figure 2.1 respectively. After the goal is reached, the agent returns to the initial state.

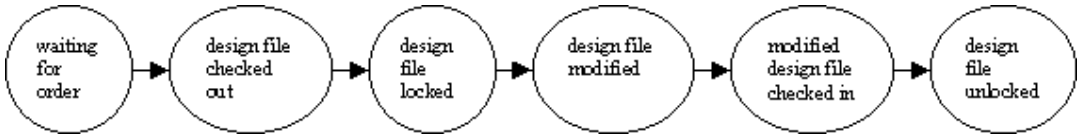

Figure 2.1 An example of state-oriented agent goal model

[Macfadzean, 96] defines a goal tree to model agent goal relationships. A goal tree is a decomposition of a top-level abstract goal into progressively lower level goals. The top goal is called a root node in goal tree. Leaf nodes represent primitive tasks or actions that can be performed by an agent. All top level goals are achieved by the performance of some ordered set of primitive tasks or actions.

An agent has goals. There are various kinds of relationships between agent's goals. Most of the agent goal models [Macfadzean, 96; Kim, 2000; Park, 2000; Kolp, 2002] model the agent's goal relationships from a static structure view (for example, goal tree, Tropos). To our best knowledge, ***little research has been reported to model the dynamic goal relationships by linking agent's goals, agent's environmental situations and goal directed behaviors during the goal pursuit process in a changing environment.*** In this research we propose Goal Net, and goal-oriented agent modeling method, for modeling both the structure relationships of agent goals and the dynamic relationships among agent goals in a real-world changing environment.

2.2.3 Action Selection for Goal Pursuit

[Maes, 94] describes action selection problem as: "Given an agent that has multiple time-varying goals, a repertoire of actions that can be performed and specific sensor data, what actions should this agent take next so as to optimize the achievement of its goals"? There are currently many algorithms existing in the world. But there is no one single method to cater for all the situations. This is because:

- Definition and modeling of agent goals vary.
- Problem environment varies. Available resource may be limited. Information may be incomplete or incorrect;
- Solutions to the problems vary.
- Criteria for action selection vary. It is hard to say that one algorithm is better than another one unless they are mentioned in the same particular environment;
- And so on.

Existing action selection algorithms can be categorized into the following two classes:

- Computation based solution: Factors or environment variables contain uncertain information. Sensor data may be incomplete or incorrect. Action selection algorithms are proposed based on mathematics such as fuzzy theory, heuristic search algorithms, etc. or using the techniques that are results from AI research, such as neural network, fuzzy cognitive map, etc [Miao, 2002; Ohtani, 2000].

- Learning/Reasoning based solution: Factors and environment variables are identified according to an agent's goals. Action selection algorithms are proposed using reasoning mechanism, such as rule-based reasoning, case-based reasoning, practical reasoning (BDI), etc. [Jennings, 98; Russell, 95; Wooldridge, 2000a].

In the following, we review some notable work of agent action selection mechanisms:

- Rule-Based Reasoning for Action Selection

  Rule-based action selection infers the appropriate actions based on condition-action rules [Jennings, 98; Russell, 95]. Rule-based action selection has good performances if the domain is limited, small enough and well defined. It is easy to cover all the possibilities and foresee all the possible situations. However, if the agent has not gained complete knowledge of its environment, or the environment keeps changing, there is always a chance that an unexpected case occurs that cannot be resolved by the rule-based action selection. Moreover, a set of action selection rules adapted to a particular problem, may not be adaptable to another problem. As a result, it has been well accepted that rule-based action selection are well suited to certain limited domains and problems only.

- Case-Based Reasoning for Action Selection

  A Case-Based Reasoning (CBR) agent evolves over time, and solves new problems based on previous experiences, which are represented as cases base [Aamodt, 94; Curet, 96; Corchardo, 98; Lenz, 98]. A case represents a situation that happened in the past. It is in general represented as a pair "problem, solution". The problem represents the description of the past situation and the solution is the

description of the action taken at that time. A case base consists of a set of cases. Case-based action selection allows shortcuts in reasoning. If an appropriate case is found, a suitable action can be found very fast (faster than generating a solution from scratch). Also, it enables agents to avoid past errors and exploit past successes. In case-based reasoning, the record of each situation that occurred is kept and used for new action selection. However, there are some pitfalls that appear. The main limitation of case-based action selection is the search time (time complexity). The size of the case base (space complexity) could increase very fast as the agent learns new cases which results in the slow response of the agents.

- BDI Reasoning for Action Selection

  The belief-desire-intention (BDI) agents are characterized by some "mental states" with three components: belief, desire, and intention. Beliefs represent information that the agent has about the environment. Desires represent goals of the agent. Intentions represent plans that the agent has chosen and has committed resources to. An agent's action selection involves repeatedly updating beliefs from information in the environment, deciding what actions are available, and acting on the basis of intentions [Rao, 92; Bratman, 87; Georgeff, 89; Wooldridge, 2000a].

  Most of the BDI work has gone into the formalization and architecture of BDI agents [Wooldridge, 2000a; Rao, 95; Busetta, 98]. The action selection relies on complex symbolic logic deduction. Therefore the implementation becomes very complicated. Rao [Rao, 95] proposed an approach to simplify the action-selection using a decision tree. However, the ability of the decision tree for solving complex problems is very limited.

In most of the current agent systems, ***one agent only employs one action selection mechanism /algorithm to act towards its goal***. As human beings, we do action selection for solving a problem in a flexible manner. Very often, we use different reasoning mechanisms to support our action selection for solving a specific problem. Some times, we use rule based reasoning for action selection, sometimes we may use case based reasoning or probabilistic reasoning. Like a human being, in a real world complex situation, one single agent may need to employ different action selection mechanisms for deciding its actions towards its goals. In this research, we are not targeting to propose a new action selection mechanism, rather, we propose a goal-oriented agent modeling method, Goal Net that facilitates an agent to incorporate different learning/reasoning mechanisms for its action selection in different environment situations in order to solve a real world problem.

2.3 Agent Architecture

2.3.1 Deliberative Agents

Since the early 1970s, the AI planning community has put tremendous effort on the design of artificial agents. Deliberative agents are also known as planning agents, or classic goal-based agents. Planning includes the design of a course of action that, when executed, will result in the achievement of some desired goal. Within the symbolic AI community, it has been assumed that some form of AI planning system will be a central component of any artificial agent. The well-known early planning system was STRIPS [Fikes, 71]. This system takes a symbolic description of both the world and a desired goal

state, and a set of action descriptions, which characterize the pre- and post-conditions associated with various actions. It then attempts to find a sequence of actions that will achieve the goal, which essentially involves matching the post-conditions of actions against the desired goal. Much effort was subsequently devoted to developing more effective techniques. Two major innovations were hierarchical and non-linear planning [Sacerdoti, 75; Sacerdoti, 74]. However, in the mid 1980s, Chapman established some theoretical results that indicate that even such refined techniques will ultimately turn out to be unusable in any time-constrained system [Chapman, 86]. Another major drawback is that planning agents assume that the environment does not change while pursuing its goal [Deugo, 99; Jennings, 98]. These results have had a profound influence on subsequent AI planning research. Some researchers started questioning the whole symbolic AI paradigm that has thus led to the work on alternative approach, Reactive Agents.

2.3.2 Reactive Agents

Reactive agents sense the environment and use rule-based knowledge to infer the appropriate actions. A reactive agent's rule-based knowledge consists of condition-action rules through which the agent takes the action that matches the current situation agent perceived [Jennings, 98; Russell, 95]. As reactive planning, reactive approaches to agent design aim to encode every possible result of a processing system in a series of rules. If a match is found then the action of the matched rule is executed. The principle behind reactive agent approaches is that the world is in its own best representation. This stance was popularized by Rodney Brooks [Brooks, 86; Brooks, 91] and has been challenged in a variety of places (e.g. [Kirsh, 91]). Examples of reactive agents and approaches to

building them include the Agent Network Architecture [Maes, 91] and work done by Leslie Pack Kaelbling on reactive systems [Kaelbling, 89]. Generating behavior in a reactive manner has a number of benefits. The principal benefit is the high speed at which an agent can determine its next action. This is due to the low processing overheads.

2.3.3 Hybrid Agents

Just as agents can be purely reactive and purely deliberative, it is also possible to construct an agent that is hybrid. A hybrid approach involves combining the main strengths of a variety of methods (e.g. reactive rules for fast reactions and a deliberative planner for goal-directed behavior) in an architecture that controls their execution. There are many ways in which hybrid agents can be constructed, and hence there are different examples in the literature [Ferguson, 92; Jennings, 98; Sycara, 98].

By evaluating the reactive and deliberative architectures, researchers drew a conclusion that for most problems, neither a purely deliberative architecture nor a purely reactive architecture is appropriate [Jennings, 98; Sycara, 98]. Therefore hybrid architectures seem a better choice. Typically, these architectures consist of a number of layers, each catering to a different level of abstraction. A famous hybrid agent architecture example is TouringMachines developed by Ferguson [Ferguson, 92]. The architecture consists of three layers: the *reactive layer* generates actions according to a set of situation-action rules in response to events that happen too quickly; the *planning layer* constructs plans and selects actions to execute in order to achieve the agent's goals; and the *modeling layer* contains symbolic representations of the cognitive state of the entities in the agent's environment.

Each layer in the TouringMachines agent architecture is an independent, activity-producing, concurrently executing process. The three layers are able to communicate with each other (via message passing), and are embedded in a control framework by using *control rule*s. Hybrid architectures such as TouringMachines have obvious advantages over both purely deliberative and purely reactive architectures. However, an outstanding problem with such architectures is its difficulties in combining multiple agent reasoning mechanisms (e.g. Rule based reasoning for reaction, and Symbolic reasoning/induction for planning) from interacting layers into a control framework. Moreover, there is a gap between the theoretical architectures and the actual implementation, which limits the applications of such architectures.

In this research, we develop a multi-agent framework for supporting the proposed goal-oriented AOSE methodology based on Goal Net. The framework presents a novel hybrid agent architecture. By combining reactive and goal directed processing in a hybrid agent, the overall problem solving process allows both the developer and agent to avoid having to tackle particular problem situations with unsuitable methods. More specifically, Goal Net incorporates both the re-active control and the anytime planner-based control in a seamless way. The novel feature of our agent framework is supported by anytime algorithms of Goal Net that enables the agents to have a practical plan at anytime, and they are able to respond immediately to environmental change.

2.4 Agent-oriented Software Methodologies

2.4.1 Goal-oriented Modeling for Software Engineering

Most of the existing agent system developments still employ object-oriented methodology. Agents are goal-oriented which necessitates a shift in modeling paradigm from object-oriented modeling to goal-oriented modeling. In fact, goal-oriented modelling has received increasing attention over the recent few years in engineering not only agent based systems but also traditional software systems [Jacobs, 95; Park, 2000; Lamsweerde, 2001; Kolp, 2001; Liu, 2002]. Research on goal-oriented modeling falls into three categories:

- Goal-oriented modeling is used as a requirement engineering method;
- Goal-oriented modeling is used for system organization;
- Goal-oriented modeling is used for system design.

Goal-oriented modeling has been used as a requirement engineering method since early 1990s. Goal-oriented requirements engineering is concerned with the use of goals for eliciting, elaborating, structuring and specifying requirements. NFR framework [Mylopoulos, 92] models goals in terms of a set of abstractions such as goal types, goal attributes and goal links. KAoS [Bradshaw, 96; Lamsweerde, 2001] presents a Knowledgeable Agent-oriented System architecture. KAoS as a goal-oriented requirement engineering framework introduces responsibility to link goals and agents [Darimont, 98]. KAoS also introduces AND/OR operational links to relate goals to the operations.

[Jacobs, 95] shows how goals can be used to manage information systems development by integrating goals and business models. This demands that goals have to be used through the whole process of business modeling. The role of goals has to change from a starting point of a top-down satisfaction to central criteria driving all decisions within the design process. Goals have to be used to estimate current models, to evaluate single alternatives, and thus help to guide the development process according to the visions on the to-be-built information system.

[Kolp, 2001] proposed architectural styles for MAS, which are intended to represent a macro-level architecture of a MAS. They were modelled using the i* framework [Yu, 2001] which offers actors (agents, roles, or positions), goals and actor dependencies as primitive concepts. The key promise here is that actors and goals can be used as fundamental concepts for analysis and design during all phases of software development, not just requirement analysis.

[Kim, 2000; Park, 2000] use goal modeling to identify agents in a multi-agent environment. Goal directed analysis, goal hierarchical system diagrams, goal structure diagrams and agent class diagrams are proposed for agent identification in a problem domain. Based on the goal hierarchical system diagram, the criteria to identify agents are given.

With such a background, a few complete methodologies for modeling and designing agent-oriented systems have been proposed which will be reviewed in the next section.

2.4.2 Agent-oriented Software Methodologies

The most famous agent-oriented methodology is Gaia methodology [Wooldridge, 2000]. Gaia targets to explore the organizational abstractions as guidelines for the analysis and design of complex and open software systems. Gaia consists of a set of models including an environment model, role model, agent model, interaction model, service model, and organizational structure and rules. The objective of the analysis phase of Gaia was to define organizational rules, an environment model and a role model, derived from the system requirement specification, together with descriptions of the protocols in which the roles will be involved. In the architectural design phase, an organizational structure will be formed followed by the agent model, interaction model, and service model in detailed design stage.

Gaia targets to provide a general methodology. It does not deal with any particular modeling method, early requirement analysis, and implementation issues. It focuses on studying organizational abstractions.

The MaSE Methodology [Wood, 2001] provides guidelines for developing MASs based on a multi-step process. In analysis, the requirements are used to define use-cases and identify application goals and sub-goals for eventually identifying the roles to be played by the agents and their interactions. In design, agent classes and agent interaction protocols are designed based on the output of the analysis phase, generating a complete architecture of the system.

The MESSAGE methodology [Caire, 2002] exploits organizational abstractions that can be mapped into the abstractions identified by Gaia. In particular, MESSAGE defines an organization in terms of a *structure*, determining the roles to be played by the agents and

their topological relations corresponding to the concept of organizational structure promoted by Gaia. In MESSAGE, an organization is also characterized by a *control entity* and by a *workflow structure*, mapping into Gaia's concept of organizational rules. In addition, MESSAGE is tightly bound to UML (and to AUML).

The Tropos methodology, first proposed in [Bresciani, 2001] and re-fined in [Kolp, 2002], attempts to use goals and plans in all phases of software development, from early analysis down to the actual implementation to meet both functional and nonfunctional requirements through goals and soft goals. It targets to cover the whole life cycle of agent development. However, Tropos's analysis still focuses on the early requirement analysis, which may undermine the effectiveness of the analysis and increase the complexity of the subsequent design phase.

All these AOSE methodologies have some common drawbacks. They only model the static structural relationships between goals without modeling the dynamic interactive relationships between goals in a goal pursuing process. Goal orientation is a most important agent property. In the existing methodologies, some efforts have appeared to attempt to model agent's goals. However, there is no systematic methodology to address this problem. The emergence of AOSE methodologies is an important step for widespread development of agent-oriented systems. However, there are still many research issues that have not been fully addressed by the current AOSE methodologies:

- There are gaps between requirements to agent design, as well as between agent design and implementations. For instance, the well-known AOSE methodology Gaia is a high level agent-oriented methodology. Gaia neither deals with requirement capturing and modeling, nor deals with implementation issues. One

important purpose of the agent design is to guide the implementation. There is a need to establish a precise connection between the agent design and implementation. Another important purpose of the design is to facilitate the realization of the requirements in an implemented system. There is also a need to build the connection between the requirement and the agent design. Hence, there is a need for an agent-oriented modeling tool that aids designers throughout the whole life cycle of agent-oriented system development.

- Most of the current AOSE methodologies focus on the entity relationships such as goal and role, and lack of the modeling of the control to autonomous agent behavior by the goals.

- There is a gap between agent mental models and agent implementation. A lot of efforts have been put into research on agent mental models. Few efforts have been put in linking the agent mental models to the agent implementations.

- There is a need to support flexible integration with other systems including traditional systems. Software is becoming more and more integrated. The emergence of web services, grid computing, and peer to peer networks makes it possible to link world wide applications and integrate them with each other. Every system might become a part of other systems. To our best knowledge, none of the current AOSE methodologies addressed the integration issues with traditional systems.

- There is a need to support the decomposition and combination throughout the lifecycle of agent-oriented system development, including requirements, design, and implementation. The need of worldwide service integration results in the fact

that no one has control over the whole system. We are now moving to a world that software systems need to be easily decomposed, modified and re-composed/re-combined "on the fly" by different people who are not under the control of the designers of the systems.

- In the above situation, how can we enable people to specify their own preference/expectations of their software as well as to gain control of their software?

From an AOSE point of view, although the autonomous and de-centralized nature of interactive agents makes multi-agent system a promising solution for the new generation of software, to date, there is still a lack of agent-oriented methodologies that can bridge the above gaps for assisting the whole development life cycle towards the widespread development and deployment of multi-agent systems. In this thesis, we present a goal-oriented methodology for bridging the above gaps and for developing multi-agent systems.

2.5 Summary

Although agents are promising in many application domains, there have been many challenges in agent research and development for real world agent-based applications.

To date, little work has been reported for modeling and characterizing the nature of an agent's goals, goal autonomy and modeling goal interactions in a multi-agent system. The existing agent systems assume the goals of an agent are implied in its actions. In the following chapters, we first characterize the goal of agents. Then, we propose a new goal-

oriented modeling method, Goal Net. The new agent goal model not only represents the characteristics of agent's goals but also models the goal decomposition, goal interactions and actions for reaching the goal. Goal measurement is also addressed to measure the goal achievement quantitatively. In addition, the proposed goal model also provides a method for identifying agents and for modeling agent coordination in multi-agent systems.

In a complex real world application, the agent execution environment keeps changing. Agents should make decisions to pursue new goals, and select suitable actions for reaching their goals. The proposed agent goal model supports both goal selection and action selection mechanisms. In particular, the proposed goal model facilitates agents using different reasoning mechanisms (for instance, probabilistic Bayesian networks is considered as one of the action-selection mechanisms) to tackle the agent action selection issues in a dynamic environment. The detailed information of the proposed agent goal model is given in the next chapter.

Analyzing, designing and implementing software as a collection of interacting autonomous agents represents a promising approach for next generation software. Agents in a multi-agent system act towards a common goal. In addition to an agent goal model, Goal Net also serves as a goal-oriented requirement and modeling tool, and a multi-agent identification, organization and coordination model. As a new software-modeling tool, Goal Net assists in all the phases of the life cycle for development of agent-based applications. The current development of agent-based systems relies on different tools in different phases of the whole life cycle. The modeling and design of multi-agent systems using the proposed goal-oriented approach have demonstrated a novel agent-oriented software engineering paradigm for designing and developing complex software systems in open distributed environments.

CHAPTER 3

GOAL NET: A GOAL-ORIENTED MODELING APPROACH

This chapter starts with characterizing the goal of an agent. To model the complex goal of an agent, Section 3.2 defines the proposed Goal Net, followed by its reasoning and learning algorithms given in Section 3.3. Section 3.4 describes the modelling of multi-agent systems using Goal Net. Finally the chapter is summarized in Section 3.5. Unlike traditional task-based or state-based agent goal models, Goal Net models the dynamic relationships among goals, environment changes during goal pursuit, and tasks for achieving the goals. Goal Net facilitates the flexible learning/reasoning mechanisms for autonomous goal selection and action selection. Goal Net encapsulates a dynamic goal pursuit process in real world environments into a composite state, which can be reused, decomposed and re-combined.

3.1 Characterizing Agent's Goal

It is well known that to date, there is no universal definition of agent. The question "*what is an agent?"* is embarrassing for the agent computing community in just the same way that the question "*what is intelligence?*" is embarrassing for the AI community. Similarly within the field of agents, "what is a goal of an agent?" is something hard to define.

There have been increasing demands for modeling an agent's goal. An agent goal model offers a method to link an agent's goals and actions. Task-oriented goal models assume that an agent lives in a task-oriented domain; the goal of an agent is a set of tasks to perform. State-oriented goal models assume an agent lives in the state-oriented domain. The agent's environment is evolved with a finite set of states. A goal of an agent is a desired state that the agent tries to reach from its current state by going through a sequence of states. Specifically, agents are designed to do certain tasks, or perform certain functions. Most of these do not know what their goals are until a specific task with a specified outcome is assigned to them. Thus they have a goal. For example, an engineer performs engineering. If we tell him to design a bridge with specifications, he has an objective or goal.

Although there is no universal agent definition, a set of agent properties or characteristics such as autonomy, intelligence, goal-oriented etc. have been well accepted in the agent community. To date, little work has been reported for characterizing the goal of an agent. Motivated by this situation and the success of characterizing agent characters, we use a similar way of conceptualizing agents for conceptualizing a goal of an agent through a list of basic characteristics or properties.

The characteristics of an agent goal are classified based on the goal theory, which includes:

- Specified: A goal of an agent can be described by a "specification".

There are many actions can be done at any point in time, but goals help an agent to choose among all of these possibilities. Goals tell an agent what is really important at a time and direct the agent to take appropriate actions for reaching them. Goals need to be specified and measurable so agents can judge when they have completed or reached the goal. If there were no clear way to tell if they reached a goal, they would not know when to stop.

- Realizable: A goal has to be realistic so that an agent can accomplish it with some effort.

A realizable goal is one for which an agent clearly has the necessary resources (including time, expertise, etc.) to accomplish the goal within a specified time period.

- Measurable: goals can be measured quantitively.

A measurable goal is one that can easily be evaluated as completed. This way an agent will know when it has accomplished its goal.

- Temporal: A goal should be given a specified time frame.

A goal must have a clear start and stop time. This tells an agent when it must begin its work and how long it has to complete it. Without a clear starting date, it is easy to put off tasks or wait too long to begin them. Without a specific stop time, agents cannot judge the progress and may not be able to finish on time.

Goals should be specific, measurable, and realistic. This ensures that an agent will spin its wheels working towards the defined goals.

An agent goal can also have one or more the following characteristics:

- Decomposable: A complex goal can be further decomposed into sub-goals.

  A general goal may be complex and require to be decomposed into more than one sub-goal to achieve it.

- Interactive: A goal has relationships with other goals.

  There are various interactive relationships between goals and behaviors. For example, a goal may be conflict with another goal or a goal may be achieved after another goal is achieved.

- Fuzzy: A goal can be specified using fuzzy concepts.

- Partial: A goal can be measured during an agent is pursuing it. For a certain goal, an agent might only need to reach it partially.

Goals can be classified by different characteristics such as fuzzy goal, partial goal, sub-goal, composite goal, atomic goal etc. The goal characterization is an important part for conceptual modeling, analysis and evaluation of the goals of agents.

3.2 Modeling with Goal Net

3.2.1 Basic Concept of the Goal Net

Agents are goal-oriented which necessitates a shift in modeling paradigm. With this new landscape, given a complex problem in a specific application domain, this chapter proposes Goal Net, a novel goal-oriented modeling approach for solving the problem through an agent-oriented system.

A goal of an agent is a desired state that the agent intends to achieve. To solve a real world problem, an agent need to achieve very complex goals that associated with many properties or characteristics listed in the above section. Give a problem specification, using object-oriented modeling method, we start from identifying objects as well as the properties and behaviors of the objects. Instead of modeling objects, using Goal Net, we start from identifying goals and possible behaviors for achieving the goals. A Goal Net is composed of four basic entities: *states*, *transitions arcs*, and *branches*.

The *states*, are used to represent different goals that agents intend to achieve. The *transitions*, are used to represent tasks that are possible to change agent from one state to another. A transition specifies the relationships between the states it joins. Each transition has at least an *input state* and an *output state*. It is associated with a task list $\{t_1, t_2, ..., t_n\}$ that defines tasks an agent may perform in order to fire a transition. When certain conditions in an agent environment are satisfied, the transition fires, and the agent will evolve from the input states to the output states. An agent environment is represented by an environment variable set $\{ev_1, ev_2, ..., ev_m\}$. An agent may choose to perform different tasks/actions for reaching a same goal under different environment situations. Moreover,

an agent may employ different reasoning mechanisms for task selection towards a desired goal.

The *arcs*, are used to connect states and transitions. An arc indicates the relationships between the state and the transition it connects.

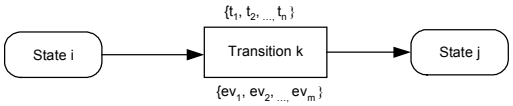

Figure 3.1 The basic elements of the Goal Net

As shown in Figure 3.1, state *i* will be changed to state *j* through the transition *k*. The arcs connect the state *i* to the transition *k* and connect the transition *k* to the state *j*.

For example, assume that a personal transport recommendation agent helps the student to decide how to go to *lecture theater* (state *j*) before 9 am every morning for taking a course subject from his *resident hall in campus* (state *i*) by transition *k* which is associated with a list of the tasks {*take a taxi*, *ride a bicycle*, *walk*, take campus shuttle bus}. The environment can be represented by an environment variable set (*time*, *whether*).

The agent recommends the student to take a task from the task list associated with the transition *k* based on the perception of the environment. The agent may use different reasoning mechanisms for task selection based on different environment situation. For instance, the agent may use a rule-based reasoning for recommending a task to the student: If it's raining and lacks of the time, take a taxi.

As illustrated, the Goal Net models the dynamic relationships among goals, environment changes during goal pursuit process and tasks for achieving the goals. An agent

autonomously chooses different tasks/actions for reaching a same goal under different environment situations.

3.2.2 The Definitions

[**Definition 3.1**] A **state** is a tuple (P, V, F, r) where P is a set of variables that define the profile of the state; V is a set of application variables; F is a set of internal functions that define behaviors on the state; and r is a time stamp.

The profile of a state *P* contains information about the state, including *ID*, *name*, *description*, *type*, *status*, *worth value*, *cost*, and *transaction number*, etc. The state *ID* and *name* are the unique identity of the state; and the *description* defines the content of the state. The *type* defines the state a composite state or an atomic state. The *worth value* indicates achievement of the goal. The *transaction number* is the number of a current transaction. Each transaction represents one round of goal pursuing of an agent. The *status* of a state could be *active*, *inactive*, *ready* or *passed* which indicates the agent is in this state, the agent has not reached this state, the agent is ready to leave or the agent has passed this state respectively. The variable *cost* indicates the effort spent on the state.

The variables in *V* are application specific attributes that represent the state of the goal pursuit process. When an agent arrives in a state, the variables will be updated, after which the status of the state will become *ready*.

The functions define behaviors of a state. They include an *initialization* function that initializes or resets the state, *variable manipulation* functions that operate the variables defined in the state, *measurement* functions (such as duration function, distance function, worth function, and goal selection function, etc.), and other application specific functions.

The *time stamp* includes the time when the state is initialized, the time when the state is most recently reset, and the time when the status of the state becomes *active*, *ready* or *passed* respectively.

There are two kinds of states in the model: an *atomic state*, accommodates a single state which could not be decomposed any more; a *composite state*, can be decomposed into other states (either composite or atomic) connected via transitions.

[**Definition 3.2**] A **transition** is a tuple (P, V, F, T, K, r) where P is a set of variables that define the profile of the transition; V is a set of application specific variables; F is a set of internal functions that define behaviors of the transition; T is a finite set of task lists; K is an action selection mechanism; and r is a time stamp.

Similar to the state, the profile of a transition contains information about the transition, including *ID*, *name*, *description*, *type*, *status*, *transaction number*, etc. The transition *ID* and *name* are the unique identity of the transition; and the *description* indicates the purpose of the transition. The *type* indicates the transition is a *direct* transition, *conditional* transition or *probabilistic* transition. The *transaction number* is the number of the current transaction. The *status* could be *enabled*, *firing*, or *disabled* which indicates the transition is enabled, firing and disabled respectively. The variable *cost* indicates the efforts spent on the transition.

The variables in *V* are application specific attributes that include the running environment variables, internal variables, and factor variables used by action selection mechanism. The functions in *F* define behaviors of a transition. A transition has input states and output states connected by input arcs and output arcs respectively. If all of the input states are ready then the transition is enabled.

T contains all the tasks that are needed for a transition. Each task can be fulfilled to make the transition fire in a certain situation. *K* defines an action selection mechanism to choose a suitable task in the current situation. There are three kinds of action selection strategies: sequential execution, rule-based reasoning and probabilistic inference. The detailed discussion about action selection mechanism proposed in this research can be found in the next section. The selected task will be executed when the transition is enabled. Each transition has a *fire condition* function. The fire condition function specifies the fire condition of progress from the input states of a transition to its output states. The most important behavior of a transition is when the fire condition is satisfied, it can *fire* to move the agent from its input states to its output states. Therefore, the evaluation of the fire function enables the agent to decide its behavior autonomously.

The time stamp records the time when the transition is enabled and the time when the transition is fired respectively. The time delay variable indicates the time delay between the time the transition is enabled and the time the transition fires.

There are three types of transitions corresponding to the three strategies of action selection mechanisms, which will be discussed in details in the next section: *direct*, *conditional* and *probabilistic*. Nevertheless, Goal Net is not limited to these three types of transitions. It also supports other types of actions selection mechanisms or user defined actions selection mechanisms.

The *direct* transitions indicate the input states can be transited to the output states via a fixed task execution. There is not any action selection mechanism involved. The *conditional* transitions indicate the task, which makes a transition fire after completion, must be selected dynamically according to the runtime conditions. Rule-based reasoning

will be involved for the action selection. In a *probabilistic* transition, the probabilistic inference will be used to select tasks in an uncertain environment. Figure 3.2 shows the three types of transitions represented by different shapes in Goal Nets.

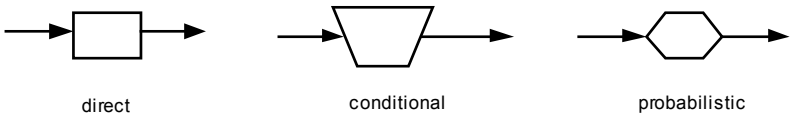

Figure 3.2 The types of transitions

In the example given by Section 3.2.1, the personal agent of the student uses a conditional transition for action selection, i.e., to decide which transport to take to go to the lecture theater. In fact based on experience, the chance is uncertain for getting a taxi from campus resident hall when it is raining. So the agent may use a probabilistic mechanism for action selection, when it is raining and time is short, to handle the uncertainty for maximizing the goal achievement.

Apart from defining the properties of a transition, the model also defines a set of firing rules for each transition, which provides a mechanism for capturing and denoting dynamic characteristics of the goal model. In particular, firing rules of the model can be summarized as follows:

1) A transition is enabled when all the input states are *ready*;

2) A transition fires as soon as the agent has successfully carried out the task selected in the task list and the fire conditions are satisfied;

3) When a transition fires, its input states become *passed*, and its output states become *active*.

[**Definition 3.3**] An **arc** is a tuple (P, i, o) where P is a set of variables that define the profile of the arc object; *i* is a link to the input state or transition; *o* is a link to the output state or transition.

An arc connects a state with a transition. There are two types of arcs: *triangle* arrows and *diamond* arrows. The two types of arcs function like normal arcs defined as above. The difference between the two types of arcs is that in the choice situation, that is, a state has more than one output arc, triangle arrows means "or" relationship between any two triangle arrows while diamond arrows represent "and" relationship between any two diamond arrows. Figure 3.3 illustrates the two types of arcs and represented relationships.

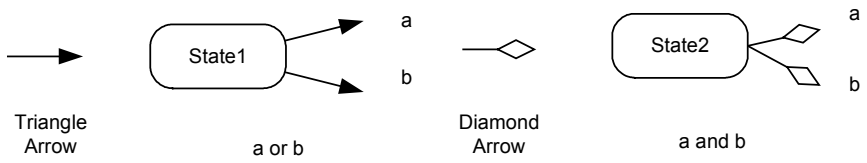

Figure 3.3 The types of arcs

[**Definition 3.4**] A **process** is defined as a tuple P = (S, T, A) where:

S is a finite set of *states*;

T is a finite set of *transitions*;

$A \subseteq S \times T \cup T \times S$ is a finite set of *arcs* that joint states and transitions;

$S \cap T = \varnothing$ and $S \cup T \neq \varnothing$;

$\exists\ s_s \in S$ where s_s is an atomic state and is the only one that has no input but at least one output. The state s_s is called *start state*;

$\exists$ $s_e \in S$ where s_e is an atomic state and is the only one that has no output but at least one input. The state s_e is called *end state*.

A process is a solution to a goal pursuit. The *start* state initializes the process and the *end* state indicates the completion of the process. If all states of a process are atomic states, the process is called a *basic* process. The Goal Net containing a basic process is called *basic Goal Net*.

[**Definition 3.5**] A **branch** is a tuple (P, V, F, s, l) where P is a set of variables that define the profile of the branch; V is a set of variables; F is a set of internal functions that define behaviors of the branch; *s* is a link to a composite state; and *l* is a link to the start state or end state of a process.

Two branches connect a composite state to a process. The left branch connects the composite state to the start state of the process while the right branch connects the composite state to the end state of the process. The *copy* function in *F* of the left branch will copy the state of the composite state to the start state of the process. After the end state is reached, the *copy* function in *F* of the right branch will copy the state to the composite state.

[**Definition 3.6**] A **composite** state is defined as a tuple $S_c = (P, s_i, s_t, B)$ where:

P is a process;

s_i is the initial state of the composite state;

s_t is the target state of the composite state;

B = $\{b_l, b_r\}$ is a two element set containing two branches. The *left branch* b_l joins s_i and the start state of P whereas the *right branch* b_r joins s_t and the end state of P.

The process *P* defines the detailed structure of the decomposition of the composite state S_c. The target state s_t is the goal of the process *P*. And the goal is reached through the process *P*. Suppose s_s is the start state of the process *P* and s_e is the end state of *P*, the following rules apply:

- The start state s_s is activated if s_i is activated.
- The state s_t is *ready* only if s_e is *ready*.
- The process *P* will be reset after the state s_t becomes *passed*.

Following the same example given in Section 3.2.1, assume that the personal transport selection scenario is frequently repeated in a complex problem. A composite state can be defined to encapsulate the goals, transitions, tasks for achieving the goals, and the dynamic goal pursuit process (Figure 3.4). The composite state represents a solution to a specific problem and can be re-used and combined with other states in a Goal Net for solving more complex problems.

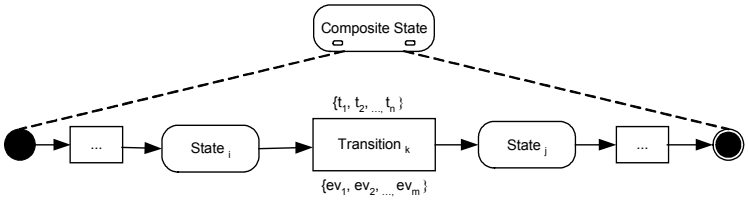

Figure 3.4 A composite state

[**Definition 3.7**] A **goal** is a desired state an agent intends to achieve. The **goal** of an agent is the root composite state of a Goal Net.

[**Definition 3.8**] A **Goal Net** is a hierarchical net. It is defined as a tuple: GN = (C, R, H, R_0) where:

C is a set of composite states, that is, goals;

$R \in C$ is the only root of the net;

H is level number of the structure;

R_0 is the initial settings of the net;

$\forall c \in C\text{-}\{R\}$, if $c \neq \emptyset$, $\exists d \in C\text{-}\{c\}$, $c \in S_d$ where S_d is a set of states of the process of d.

As defined above, in a Goal Net, a goal as a composite state can be decomposed to other states or other composite states that can be further decomposed. A Goal Net can be composed of many other Goal Nets due to the state decompositions. The states in a same decomposition are connected by transitions that indicate the relationships and scheduling between the states.

[**Definition 3.9**] For Goal Net G and Goal Net G', if the root state of G' is a state of the decomposition of the root state of G, the goal of G', g', is called a **sub-goal** of the goal of G, g, whereas g is called the **super goal** of g'. And, G' is called a sub-net or sub-Goal Net of G whereas G is called the super net or super Goal Net of G'.

For example, Figure 3.5 shows the structure of an agent goal model modeled by Goal Net. The root composite state in the highest level of the model represents the overall goal of the agent. The composite states in lower levels represent sub-goals of the agent. A higher level of composite state (goal or sub-goal) can be decomposed into lower-level states connected via transitions. In Figure 3.5, the 3-level hierarchical goal model contains 13 states, 9 transitions, and 18 arcs. Three of the states are composite states. The one in the top level is the goal of the agent whereas the other two composite states in the middle level are sub goals.

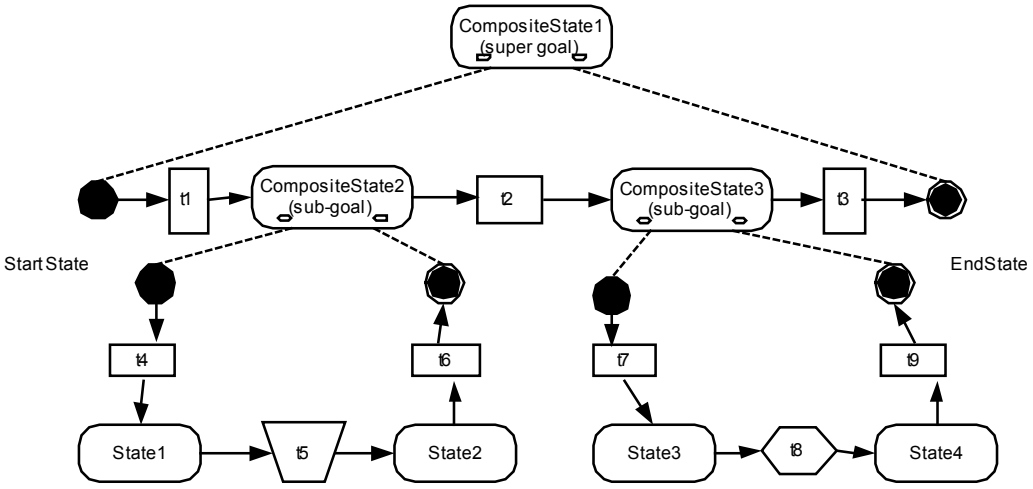

Figure 3.5 Sub goals and super goals in the Goal Net

The Goal Net proposed in this chapter has rich ability to model agent goals. To reach a goal, an agent requires many processes. That is, a goal can be divided to many sub-goals. Similarly, a sub-goal can be further divided to many sub-goals according to the complexity of the processes for reaching the goal. So, the goal, sub-goals and their relationships form a Goal Net. The root of the Goal Net is the final goal. An agent has many sub-goals during its pursuit of the final goal.

The goal pursuit of an agent is the execution of the tasks associated with the transitions that transform the agent from the initial state to the target state. It starts from the initial state of the root state, which represents the goal of the agent; then goes through the Goal Net via processes in different sub nets; and finally goes back to the target state of the root state. The goal of the agent is said to be reached at this time.

3.2.3 Goal Relationship

Transitions can represent four basic temporal relationships between states: *sequence, choice, concurrency, and synchronization*.

- ***Sequence***: this type of relationship designates a direct connection in sequence from input states to output states. It defines successive relationship between input states and output states. For example, in Figure 3.6 (a), State *i* is connected to State *j* via a transition. This implies that State *j* should be reached after State *i* is reached, in other words, State *j* is a continuous state of State *i*. In Figure 3.6 (b), it indicates that after State *i* is reached, the next state will be either State *j* or State *k* depending on the execution of the transition. For instance, if the execution of the task of the transition succeeds, it will reach State *j*; otherwise, it will reach State *k*.

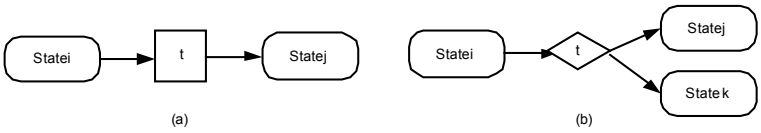

Figure 3.6 Sequence relationship

- ***Concurrency***: this type of relationship specifies a concurrent occurrence between states. It defines concurrent relationship between states. For example, in Figure

3.7, State *j* and state *k* are two concurrent states. It indicates that after State *i* is reached, it will pursue the State *j* and State *k* concurrently. The difference between (a) and (b) is that in (a) the State *j* and State *k* will be reached by the same transition whereas in (b) the State *j* and State *k* will be reached by different transitions.

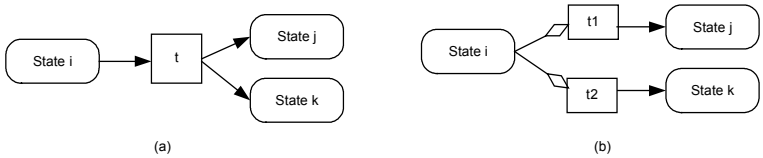

Figure 3.7 Concurrency relationship

- ***Choice***: this type of *relationship* specifies a choice connection from one state to other states. It defines a choice relationship between states. In Figure 3.8, State *i* is connected to State *j*, while State *i* is also connected to state *k*. This indicates agent may choose to proceed from State *i* to State *j* in a certain condition or from State *i* to State *k* in another condition. The goal selection mechanisms are used to solve the conflict introduced by this type of relationship.

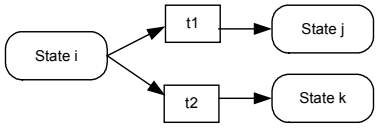

Figure 3.8 A choice relationship

- ***Synchronization***: this type of relationship specifies a synchronization point between states. It defines synchronization relationship between states. For example, in Figure 3.9 (a) and (b), State *i* and State *j* are synchronized before the State *k*. The difference between (a) and (b) is that in (a), the transition is enabled

only when State *i* and State *j* are both reached whereas in (b) State *i* and State *j* have separate transitions to reach the State *k*.

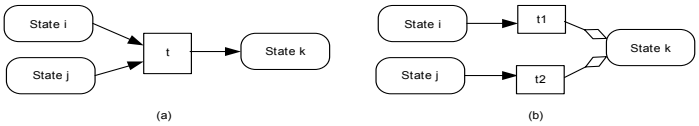

Figure 3.9 Synchronization relationship

With different combination of the four relationships between states, a wide range of complicated relationships between states can be represented, which could accommodate various complex goals of the intelligent agents.

3.2.3.1 Goal Composition

A **composite goal** is a goal that is composed of sub-goals. The composite goal is also called super goal of the sub-goals. An **atomic goal** is a goal that is not composed of any sub-goal.

For example, in Figure 3.5, composite state 1, composite state 2 and composite state 3 represent three goals, say g_1, g_2 and g_3 respectively in the Goal Net. The goal g_1 is a composite goal or super goal which is composed of the goal g_2 and g_3 whereas g_2 and g_3 are atomic goals. They are the sub-goals of the goal g_1.

3.2.3.2 Goal Interaction

The sub-goals of a super goal have interaction relationships for achieving the higher level goal, the super goal. The fundamental relationships include "*one of*", "*all of*" and "*sequential*" relationships. Figure 3.10 shows a goal relationship diagram with 6 sub-goals.

In Figure 3.10, the sub-goals g_1, g_2, g_3, g_4 are required to be achieved before the sub-goal g_n can be achieved. That is, "*all of*" goals are to be achieved. However, the order of the goal achievement is not of concern, which means the sub-goals g_1, g_2, g_3, g_4 can be achieved concurrently.

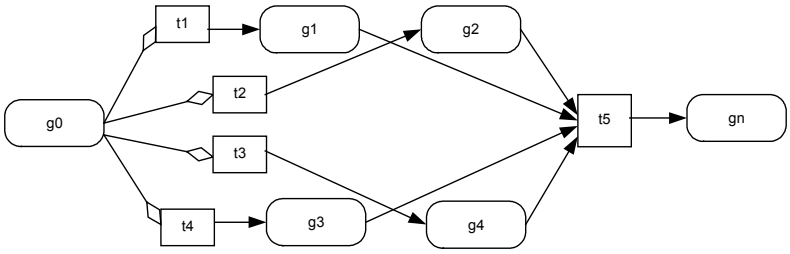

Figure 3.10 The "all of" relationship

As we know, upon reaching a goal, achievements are collected. The transition from one goal to another goal may have requirement of certain achievement. For example, a student needs to accumulate enough academic credits to satisfy the graduation requirement.

The second fundamental relationship is "one of", which is illustrated in Figure 3.11. After achieving goal g_0, we need to achieve one of the sub-goals g_1, g_2, g_3 to reach g_n.

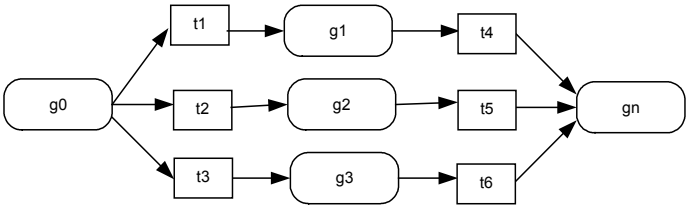

Figure 3.11 The "one of" relationship

For example, a student needs 4 more credits to graduate. g_1= "Algorithm", g_2= "Software Agent", g_3 = "Distributed Computing" are 4 credit subjects that he has not chosen. He could choose any "one of" the subjects to be eligible for graduation (g_n).

Besides the two goal relationships, a "sequential" relationship has been implied in the descriptions above. The "sequential" relationship means that a number of goals have to be achieved in sequence. An example is shown in Figure 3.12, where g_0= "Data Communication", g_1= "Computer Networks", g_2= "Multimedia Network Design", g_3 = "Distributed Computing", and g_n = "Graduation".

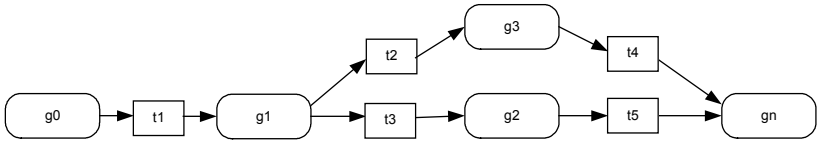

Figure 3.12 The "sequential" relationship

There are two valid selections after reaching the goal g_1 and the order has to be maintained. A student could not take g_2 before he/she takes g_0 or g_1, as the subjects have pre-requisites for the background knowledge.

3.2.3.3 Goal Generalization

Goal Net also supports inheritance. A Goal Net is an independent self-contained module that encapsulates the internal details of its states, transitions and processes. A goal is corresponding to a Goal Net or a sub-net based on the definition of a goal.

Two goals may have parent-child relationship. The child goal inherits the Goal Net structure from the parent goal. For example, *to get a course learned* is a parent goal while *to get a Java course learned* is a child goal. The parent goal is modeled with a Goal Net, in which the way to learn a course is modeled and defined. The Java course is a course so the way to learn a general course should be able to be used to learn the Java course. Therefore by inheritance from the parent goal, the child goal can share the same structure

of the Goal Net. However, the child goal can change the structure and definitions of the inherited Goal Net to meet the specific requirement for the Java course. In details, a child goal can have the following forms of inheritance:

- Add new states and transitions – In the child Goal Net, new states and transition can be added.
- Override existing definitions – In the child Goal Net, the definitions inherited from the parent goal can be re-defined.

3.2.4 Goal Measurement

Goal measurement facilitates goal pursuit monitoring and agent reasoning. In this research, goal measurement includes:

- Achievement: represents a recognizable benefit of reaching a goal;
- Distance: indicates how close the current state is to a composite state or a sub-goal;
- Completeness: represents a percentage of the entire goal fulfillment; and
- Cost: means the time, memory, money, etc. spent or required to be spent from one state to another.

In a Goal Net, a path from a state s_1 to another state s_n is a sequence of connected states that starts from the state s_1 and ends at the state s_n. The shortest path from a state to another is the path between the two states that contains the least number of states while the longest path is the path that contains the most number of states. The distance from a

state to another is the number of states on the longest path between them. The distance from a state to itself is 0 and the distance from a state to an unreachable state is ∞.

In each state, a *local distance value* indicates the number of states on the longest path to the end state; a *cost value* indicates the cost spent on the state; a *total cost value* indicates the cost to reach the state; an *achievement value* indicates the achievement can be obtained after reaching the state; a *total achievement value* indicates the achievement accumulated after reaching the state; and a *completeness value* indicates the percentage of the entire goal fulfillment. The initialization functions of each state will set the initial values of above variables respectively. The *local distance value* can be known by the model structure. For example, if a *local distance value* of current state equals to 3, there will be 3 states between current state and the end state in the longest path.

Suppose a Goal Net is an L-layered Goal Net. In a process of a state decomposition, the distance from the start state to the end state is D_{lj} and from any state to the end state is d^k_{lj} where l is the layer number; j is the jth process at the layer l; and k is the kth state in the jth process. Then the goal completeness at a state s can be measured as following:

$$C(s) = \prod_{i=2}^{l}(1-\frac{d_{ij}^{k}}{D_{ij}}) \quad (\text{i} = 2, \ldots, l \text{ is the layer number}) \qquad (3.1)$$

By goal measurement, an agent can measure the goal completeness at any state during the agent's goal pursuit period. Similarly, achievement on each state can be measured using worth value function. Satisfaction or a threshold value can be defined in the case a goal is partially completed or the achievement is partially achieved. The goal that is partially achieved is called partial goal. Sometimes a goal is difficult to achieve but partial goal is acceptable. By defining a threshold value, the agent can proceed to pursue the next goal if the partial goal has been achieved within the given time range or cost range.

However, sometimes it is difficult to decide a precise value for the threshold value. To allow linguistic specifications such as "essentially satisfied" or "approximately satisfied", there is a need for expressing such *imprecise* specifications with fuzzy values. The goal whose achievement is described using fuzzy values is called fuzzy goal. Therefore, the worth value function will be defined as a fuzzification function for the fuzzy goal measurement.

The achievement and the cost measurement will be further discussed for goal selection and action select in the section 3.3.

3.2.5 Properties of the Goal Net

A Goal Net has property: safety, liveness, and modularity.

The decomposition of a composite state is a refinement of a state.

[**Definition 3.10**] In a Goal Net Z, “a composite state s is refined” means that the process P of the decomposition of the state s is inserted in the place of the state s such that for every arc x <- t from X to P or y -> t from P to X, it is true that:

- X is the Goal Net Z without state s;
- x is the start state of P, y is the end state of P, and t is a transition of X;
- in Z there is an arc s <- t or s -> t.

[**Definition 3.11**] A Goal Net X is a refinement of a Goal Net Z, if X is a result of the refinement of a number of composite states of Z.

3.2.5.1 Safety

In a refinement of Goal Net, a state s is reachable from the initial state s_0 if there exists a sequence of transition firings that transforms s_0 to s.

A refinement of Goal Net is safe if for any state s, it is reachable from the initial state s_0 and the target state s_n is reachable from the state s.

A Goal Net is safe if its refinement is safe.

The property safety indicates whether an agent exhibits all desirable behaviors and no undesirable ones.

3.2.5.2 Liveness

In a refinement of Goal Net, a transition t is live if there is a sequence of transitions whose firing reach a state that enables t. In other words, t is live if there is a transition firing sequence that includes t. A refinement of Goal Net is live if every transition in it is live. A Goal Net is live if its refinement is live.

In a refinement of Goal Net, a transition t is dead if there exists a situation such that there is no sequence of transition firings to enable t starting from a state s. A Goal Net contains a deadlock if there exists such situation at which no transition is enabled.

Liveness of a Goal Net means that for any transition and the task associated with the transition in the Goal Net, there exists a situation that the task would be fulfilled and the transition would be fired so as to move the agent to the next state.

3.2.5.3 Modularity

The hierarchical structure of a Goal Net is built through state/goal decomposition. The scope of a state is in the process of the decomposition. So there is no transition that joins two states from different processes.

A Goal Net is correct if it satisfies the above three properties.

3.3 Reasoning and Learning With the Goal Net

3.3.1 Goal Reasoning and Learning

An agent tries to achieve a set of goals in a complex, dynamic environment. It is autonomous, that is, it reasons and decides itself how to select the next goal and what actions it should take so that its goal is attended to successfully. An agent should be able to improve its behaviors over time, that is, it becomes better with experience at selecting the next goal and achieving the goal by taking correct actions. Specifically, this leads to two problems:

- Agent reasoning: How can an agent select its goals and how can an agent select actions for achieving the selected goal?

- Agent learning from experience: How can an agent improve its performance over time based on its experience? In other words, how can it correct "wrong" or ineffective action selecting structures?

In current research, there are few papers on the nature of goals and goal interactions for agents [Maes, 98]. The goals of agents are either implicitly defined in their actions or

explicitly defined with situation-goal-action rules. There are few architectures supporting active or goal-driven action selection.

This section will address these goal reasoning and learning issues from the Goal Net perspective. Goal Net defines goals of an agent and their interactions as well as actions for the transitions between the goals. Section 3.3.2 presents three goal selection algorithms, each of which caters for a particular situation an agent falls in. The complexity of each algorithm is also discussed. Section 3.3.3 introduces the action selection strategies used in the goal model and propose two action selection algorithms catering for the environment with uncertainty. Two learning mechanisms are given at the end of the section as well.

3.3.2 Goal Selection

One of the important state behaviors is that it can compute the *worth value* of the state through the *worth* function. The worth function of the state gives a quantitative value for specifying the progress that the agent will reach. With the goal measurement, an agent is able to choose its next sub-goal autonomously based on its available resources and constraints, such as time constraint, cost constraint, etc.

When a running agent has reached a state that has choices, it needs to decide the next goal to pursue. There are two methods in Goal Net for this based on the definition of the current goal. One method is based on condition rules. For example, a business forecasting agent takes a neural network as its own knowledge to do forecasting. The neural network based forecasting model needs to be updated regularly to maintain the accuracy of the inference results. But it is not necessary to train the model very often. So each time when

the forecasting agent infers the forecasting result, it will make a decision whether it needs to train the forecasting model this time before doing forecasting. The goal selection function will be defined to evaluate the forecasting model based on the errors of the past forecasting results. The fuzzified evaluation result will be used to make the decision for its next goal, i.e. to train the model or to proceed to do forecasting using the current model. This method only considers the next possible goals into computation.

In the other method, the goal selection function will compute the index values by going through possible paths. Based on the time or cost constraints and the returned value of the goal selection function, the decision is made accordingly. The following sessions will focus on this method.

As defined in the last section, a goal is a state to be achieved. A goal is either an atomic goal or a composite goal. A composite goal is denoted by G. An atomic goal is denoted by g. Given a problem, to solve the problem is regarded as the *overall goal*. An atomic goal is clearly defined as a non-decomposable goal. When the problem, or *overall goal* is complex, it could be decomposed into sub-goals. Sub-goals could be further decomposed until the hierarchical structure and the relationships of the goals are clearly defined. Obviously, the process is completable as all sub-goals can finally be decomposed to atomic goals.

Once a Goal Net is constructed, each goal knows the goals it can reach and the corresponding achievements required. An achievable goal set is the collection of all the goals that a goal g_0 could reach directly and is denoted as $AGS(g_0)$.

3.3.2.1 Measurement for the Goal Selection

The fundamental goal selection is to choose a series of goals that lead to the final goal. There are a number of measurement factors that affect the goal selection.

- **Cost**

In a Goal Net, cost could be generated at two scenarios. One is the transition from one goal to another. For example, a passenger flying from Singapore to Tokyo could be represented as the transition from one goal g_0 = "be in Singapore" to another goal g_1="be in Tokyo". Such transitions cost time, money and energy. Another type of cost happens when a goal is achieved. For example, a mobile bio-agent migrates to a genetic alignment process server to check whether there are known patterns in the protein fragment (sequence) it carries. Upon finishing, a statement of either yes (or no) is obtained and the goal is achieved (or failed). Such a service costs time and money if it is not a free service. After finishing the current goal, it is ready to transit to another goal towards the final goal. In different applications, costs take different shapes. The general consideration is to minimize the cost or keep it at an acceptable level. Cost of the transition from goal g_i to goal g_j is denoted as $COST(g_i, g_j)$. Cost of a goal g_i is denoted as $COST(g_i)$. Additionally, the total cost consumed upon finishing a goal g_i is denoted as $\sum COST(g_i)$.

- **Achievement**

An achievement indicates a recognizable benefit of reaching a goal. Similar to costs, achievement takes different shapes in different applications. It could be:

1) Accumulative: whenever it completes a goal, it has a certain amount more of achievement accumulated. Examples can be found in many applications such as business, when a merchant makes purchase, he would have more in the store. Or when a traveler makes a trip, he/she accumulates mileage that could win him/her free tickets.

2) Valid: certain achievements have valid period. It would expire after certain time. For example, academic credits may only be valid for five years for applying for graduation. Another example is a traveler who does not have enough energy to continue the travel. He/she may choose a sub-goal of "rest in hotel" to gain the achievement of energy. This achievement would expire after 10 hours if he/she does not use it for travel.

3) Costly: cost and achievement could affect each other thus we need to trade off while performing goal selection. For example, a student who has achieved high marks in a pre-requisite subject, would need less cost (time, energy) to pass the current subject.

4) Non-accumulative: repeating a subject won't win more credits if the credit has already been gained.

The achievement of a goal g_i is represented as $\mathcal{ACHIEVEMENT}(g_i)$. The accumulated achievement is denoted as $\sum\mathcal{ACHIEVEMENT}(g_i)$. The achievement form goal g_i to goal g_j is denoted as $\mathcal{ACHIEVEMENT}(g_i, g_j)$.

- **Constraints**

Constraints are enforced requirements for achieving the final goal. It could be based on cost, such as reaching the final goal in 10 hours; or based on achievement, such as arriving at the meeting room with annual reports and minimum 70% energy. Constraints also help in avoiding the endless loop in goal selection. For example, time is a non-decreasing cost. An algorithm could always terminate upon the allowed time frame.

- **Index**

To compare two or more choices of goal selection, an index is needed for comparison. As there could be many costs and achievements, trading off is often needed to make a choice. Index is a function of costs and achievements that map to a real number. For example, if a student would like to spend least time to have most academic credits, he/she has to indicate his weight for the two factors: time (cost) and academic credits (achievement). Index at a goal is denoted as $INDEX(g_i)$. The index form goal g_i to goal g_j is denoted as $INDEX(g_i, g_j)$. The index function $INDEX(g_i) = F_{index}(\sum COST(g_i), \sum ACHIEVEMENT(g_i))$.

Goal selection could be very complex. Together with the additional requirements of minimizing the cost, maximizing the achievement and satisfying constraints, there could not be a practically efficient algorithm that could provide the best choice. In fact the general problem could be a NP-Complete problem. Therefore, in a Goal Net, we try to avoid such a situation by

1) using hierarchical composite goals to reduce the complexity,

2) using multiple agents for load balancing to improve the performance.

By doing so, we may not have the global optimal solution. However, we can more likely have a practical approximate optimal solution.

3.3.2.2 The Goal Selection Algorithms

With the definition of cost, achievement, constraints and index, we could look for the optimal goal selection algorithms. The ideal goal autonomy of an agent is that it can always find the optimal goal pursuit path by which it can obtain the maximum achievement at the lowest cost or the shortest time. However, an agent sometimes can finish the computation within an acceptable time and sometimes it can't, in which case, the agent just failed the goal pursuit. Based on the anytime algorithm [Hansen, 2001], an agent should always have a solution in allowable time for the goal pursuit. The goal selection algorithms listed below provide a valid goal sequence first before an optimal sequence is found.

Goal Selection Algorithm 1 – GSA1

This anytime algorithm is suitable for agents in an environment, such as mobile environment, where time and computation capacity may not always allow the optimal algorithm to finish. The algorithm could quickly return a goal pursuit path for the agent before a more optimal path is computed, which will replace the previous one. The more time an agent has, the more optimal path the agent can compute.

First, let us look at the goal selection algorithm for optimal cost and achievement. (i.e. no constraints). The algorithm is based on Dijkstra's algorithm [Dijkstra, 59].

Goal Selection Algorithm 1 – GSA1

Input: a set of goals G=$\{ g_0, g_1, \ldots g_n\}$ of a Goal Net
$\mathcal{COST}=\{\mathcal{COST}(g_i, g_j)| g_i, g_j \in G\}$
$\mathcal{ACHIEVEMENT}=\{\mathcal{ACHIEVEMENT}(g_i, g_j)| g_i, g_j \in G\}$
Output: a sequence of goals that leads from the initial goal g_0 to the final goal g_n.
S←Φ //initialize a temporary goal set as empty set
I←Φ //initialize a temporary index set as empty set

P←Φ //initialize a temporary goal set as goal path
For each goal $g \in G - \{g_0\}$ do
    If $g \in AGS(g_0)$ then
        $COST(g_0, g) = COST(g_0, g) + COST(g)$
        $ACHIEVEMENT(g_0, g) = ACHIEVEMENT(g)$
        $INDEX(g_0, g) = \mathcal{F}_{index}(COST(g_0, g), ACHIEVEMENT(g_0, g))$
        $I = I \cup \{INDEX(g_0, g)\}$
    Else
        $COST(g_0, g) \leftarrow \infty$
        $ACHIEVEMENT(g_0, g) \leftarrow 0$
        $INDEX(g_0, g) \leftarrow \infty$
    End If
    $\pi(g) \leftarrow g_0$ // the predecessor for the goal, temporary storage
    $l(g) \leftarrow 0$ //path length , temporary storage
End For
$COST(g_0, g_0) \leftarrow 0$
$ACHIEVEMENT(g_0, g_0) \leftarrow 0$
$S \leftarrow S \cup \{g_0\}$
$P \leftarrow P \cup \{g_u\}$ g_u is the goal in G-S with the smallest $INDEX(g_0, g_u)$ in I
While (G-S) and I are not empty do // G-S is the minus of two sets
    $g_u \leftarrow$ the goal in G-S with the smallest $INDEX(g_0, g_u)$ in I
    $S \leftarrow S \cup \{g_u\}$
    $I \leftarrow I - \{INDEX(g_0, g_u)\}$
    For each goal $g_v \in AGS(g_u)$ do
        If $INDEX(g_0, g_v)$
            $> \mathcal{F}_{index}(COST(g_0, g_u) + COST(g_u, g_v) + COST(g_v),$
                $ACHIEVEMENT(g_0, g_u) + ACHIEVEMENT(g_v))$
        Then
            $COST(g_0, g_v) = COST(g_0, g_u) + COST(g_u, g_v) + COST(g_v)$
            $ACHIEVEMENT(g_0, g_v) = ACHIEVEMENT(g_0, g_u) + ACHIEVEMENT(g_v)$
            $INDEX(g_0, g_v) = \mathcal{F}_{index}(COST(g_0, g_u) + COST(g_u, g_v) + COST(g_v),$
                $ACHIEVEMENT(g_0, g_u) + ACHIEVEMENT(g_v))$
            If $g_v \neq g_n$ then
                $I \leftarrow I \cup \{INDEX(g_0, g_v)\}$
            End If
            $\pi(g_v) \leftarrow g_u$
            $l(g_v) \leftarrow l(g_u) + 1$
            If $g_v = g_n$ then // We have found a goal sequence
                $P(l(g_n)) \leftarrow g_n$
                For $i \leftarrow (l(g_n) - 1)$ to 0 do
                    $P(i) = \pi(P(i+1))$
                End For
                $P = \{P(0), \ldots, P(l(g_n))\}$
            End If
        End If
    End For
End while

Computational Complexity of GSA1

There are three major operations in the algorithm:

1) Find minimum which has $O(n^2)$ complexity

2) Update the values of factors, which loops *e* times, where *e* is the number of arcs. Each time it will calculate the values of *s* factors. The factors include cost, achievement and index in this case. So the complexity for updating values is O(*nes*).

3) Generate the goal paths, which loops *l* times, where *l* is the number of arcs in a path. So the complexity for generating the goal paths is O(*nel*).

So the complexity of the algorithm is $O(n^2 + nes + nel) = O(n^2 + ne^2)$.

[Theorem 3.1] GSA1 gives the best goal selection according to the index.

Proof:

After all steps of the algorithm (all round of the *While*), the selected goal sequence is in *P*. Suppose $g_u \in P$, $g_n \in AGS(g_u)$, the goal sequence would be $g_0, \ldots, g_u, g_n$.

If the goal sequence in *P* is not the optimal one based on the index, there exists a goal g_x where $g_n \in AGS(g_x)$ and $g_x \neq g_u$, we have,

$$\begin{aligned} & \mathcal{F}_{index}(COST(g_0, g_x) + COST(g_x, g_n) + COST(g_n), \\ & \qquad ACHIEVEMENT(g_0, g_x) + ACHIEVEMENT(g_n)) \\ < \quad & \mathcal{F}_{index}(COST(g_0, g_u) + COST(g_u, g_n) + COST(g_n), \\ & \qquad ACHIEVEMENT(g_0, g_u) + ACHIEVEMENT(g_n)) \end{aligned}$$

According the algorithm, if g_u is put in *S* before g_x, we would have

$$\begin{aligned} INDEX(g_0, g_n) = \mathcal{F}_{index}(& COST(g_0, g_x) + COST(g_x, g_n) + COST(g_n), \\ & ACHIEVEMENT(g_0, g_x) + ACHIEVEMENT(g_n)) \end{aligned}$$

The goal sequence would be $g_0, \ldots, g_x, g_n$. If g_u is put in *S* after g_x, the goal sequence would not change, because

$$\begin{aligned} & INDEX(g_0, g_n) \leq \mathcal{F}_{index}(COST(g_0, g_x) + COST(g_x, g_n) + COST(g_n), \\ & \qquad ACHIEVEMENT(g_0, g_x) + ACHIEVEMENT(g_n)) \\ < \quad & \mathcal{F}_{index}(COST(g_0, g_u) + COST(g_u, g_n) + COST(g_n), \\ & \qquad ACHIEVEMENT(g_0, g_u) + ACHIEVEMENT(g_n)) \end{aligned}$$

So, in both case, the goal $g_u \notin P$. This is in contradiction with the assumption $g_u \in P$.

The contradiction proves that the Theorem 3. 1 holds.

Anytime Solution of GSA1

GSA1 has a desirable feature. That is, it could provide an approximate optimal solution at "any time" before the global optimal solution is found. Here the "any time" means that at a different (lower) level of computational complexity, an approximate optimal solution is obtained via aiming the global optimal solution.

The algorithm GSA1 gives a way of goal selection to maximize the achievement and to minimize the cost, according to the index function defined for the overall goal. Besides the optimal concern, there are also some constraints for the goal selection. For example, if a student could not obtain sufficient academic credits in 9 years (with 5 years extension to the 4 years program), he would not be able to fulfill the goal of graduation.

Normally, constraints are also based on costs and achievements. As mentioned above, the constraints would be that the cost time is shorter than 9 years and the achievement of academic credits is more than a certain number. Unlike the index function that has to be a scalar function, constraints could be multi-dimensional functions over costs and achievements. Here $\mathcal{F}_{constraints}$ is used to represent the constraint function. The following algorithm is for goal selection with optimal index subject to constraints.

Goal Selection Algorithm 2 – GSA2

Input: a set of goals G=$\{ g_0, g_1, \ldots g_n \}$ of a Goal Net
$\mathcal{COST}=\{ \mathcal{COST}(g_i, g_j) | g_i, g_j \in G\}$
$\mathcal{ACHIEVEMENT}=\{ \mathcal{ACHIEVEMENT}(g_i, g_j) | g_i, g_j \in G\}$
Output: a sequence of goals that leads from the initial goal g_0 to the final goal g_n.
S←Φ //initialize a temporary goal set as empty set

I←Φ //initialize a temporary index set as empty set
P←Φ //initialize a temporary goal set as goal path
For each goal $g \in G - \{g_0\}$ do
    If $g \in AGS(g_0)$
        and $\mathcal{F}_{CONSTRAINT}(COST(g_0, g) + COST(g), ACHIEVEMENT(g))$ then
        $COST(g_0, g) = COST(g_0, g) + COST(g)$
        $ACHIEVEMENT(g_0, g) = ACHIEVEMENT(g)$
        $INDEX(g_0, g) = \mathcal{F}_{index}(COST(g_0, g), ACHIEVEMENT(g_0, g))$
        $I = I \cup \{INDEX(g_0, g)\}$
    Else
        $COST(g_0, g) \leftarrow \infty$
        $ACHIEVEMENT(g_0, g) \leftarrow 0$
        $INDEX(g_0, g) \leftarrow \infty$
    End If
    $\pi(g) \leftarrow g_0$ // the predecessor for the goal, temporary storage
    $l(g) \leftarrow 0$ //path length , temporary storage
End For
$COST(g_0, g_0) \leftarrow 0$
$ACHIEVEMENT(g_0, g_0) \leftarrow 0$
$S \leftarrow S \cup \{g_0\}$
$P \leftarrow P \cup \{g_u\}$ g_u is the goal in G-S with the smallest $INDEX(g_0, g_u)$ in I
While (G-S) and I are not empty do // G-S is the minus of two sets
    $g_u \leftarrow$ the goal in G-S with smallest $INDEX(COST(g_0, g_u), ACHIEVEMENT(g_0, g_u))$ in I
    $S \leftarrow S \cup \{g_u\}$
    $I \leftarrow I - \{INDEX(g_0, g_u)\}$
    For each goal $g_v \in AGS(g_u)$ do
        If $INDEX(g_0, g_v)$
            $> \mathcal{F}_{index}(COST(g_0, g_u) + COST(g_u, g_v) + COST(g_v),$
            $ACHIEVEMENT(g_0, g_u) + ACHIEVEMENT(g_v))$
            and
            $\mathcal{F}_{CONSTRAINT}(COST(g_0, g_u) + COST(g_u, g_v) + COST(g_v),$
            $ACHIEVEMENT(g_0, g_u) + ACHIEVEMENT(g_v))$
        Then
            $COST(g_0, g_v) = COST(g_0, g_u) + COST(g_u, g_v) + COST(g_v)$
            $ACHIEVEMENT(g_0, g_v) = ACHIEVEMENT(g_0, g_u) + ACHIEVEMENT(g_v)$
            $INDEX(g_0, g_v) = \mathcal{F}_{index}(COST(g_0, g_u) + COST(g_u, g_v) + COST(g_v),$
                $ACHIEVEMENT(g_0, g_u) + ACHIEVEMENT(g_v))$
            If $g_u \neq g_n$ then
                $I \leftarrow I \cup \{INDEX(g_0, g_v)\}$
            End If
            $\pi(g_v) \leftarrow g_u$
            $l(g_v) \leftarrow l(g_u) + 1$
            If $g_u = g_n$ then // We have found a goal sequence
                $P(l(g_n)) \leftarrow g_n$
                For $i \leftarrow (l(g_n) - 1)$ to 0 do
                    $P(i) = \pi(P(i+1))$
                End For
                $P = \{P(0), \ldots, P(l(g_n))\}$
            End if
        End If
    End For
End while

Computational Complexity of GSA2

There are three major operations in the algorithm similar to the algorithm GSA1.

So the complexity of the algorithm is $O(n^2 + nes + nel) = O(n^2 + ne^2)$.

The proof of correctness of the algorithm is similar to the proof of Theorem 3.1. The difference is that the optimal solution will be obtained according to both the index function and the constraint function. It is omitted here.

Another scenario in the goal selection is that there could be trustiness relationship among goals. That is to say, a goal could only accept the transitions that are originated from a few goals. For example, the US embassy, for a period of time, does not accept personal visa applications. Tourists should however, go through certain travel agencies that are authorized by the embassy. Another example could be found that a number of international corporations put their headquarters in Singapore although the main factories are in the neighboring countries such as Malaysia, Indonesia and so on. Orders come to Singapore and then go to the factories. The products are shipped back to Singapore for distribution. One of the main reasons would be the trust issue.

Goal Selection Algorithm 3 – GSA3

Input: a set of goals $G=\{ g_0, g_1, \ldots g_n\}$ of a Goal Net
$\mathcal{COST}=\{ \mathcal{COST}(g_i, g_j) | g_i, g_j \in G\}$
$\mathcal{ACHIEVEMENT}=\{ \mathcal{ACHIEVEMENT}(g_i, g_j) | g_i, g_j \in G\}$
$C=\{c_i \subseteq G | i=0,1, \ldots n\}$ //non-trust set
Output: a sequence of goals that leads from the initial goal g_0 to the final goal g_n.
S←Φ //initialize a temporary goal set as empty set
I←Φ //initialize a temporary index set as empty set
P←Φ //initialize a temporary goal set as goal path
For each goal $g_x \in G - \{g_0\}$ do
If $g_x \in AGS(g_0)$ and IsTrusted (g_x, c_x) then
$\mathcal{COST}(g_0, g_x) = \mathcal{COST}(g_0, g_x) + \mathcal{COST}(g_x)$
$\mathcal{ACHIEVEMENT}(g_0, g_x) = \mathcal{ACHIEVEMENT}(g_x)$
$\mathcal{INDEX}(g_0, g_x) = \mathcal{F}_{index}(\mathcal{COST}(g_0, g_x), \mathcal{ACHIEVEMENT}(g_0, g_x))$

```
                I = I ∪ {INDEX(g0, gx)}
        Else
                COST(g0, gx)← ∞
                ACHIEVEMENT(g0, gx)←0
                INDEX(g0, gx)← ∞
        End If
        π(gx)←g0                    // the predecessor for the goal, temporary storage
        l(gx)←0                     //path length , temporary storage
End For
COST(g0 , g0)← 0
ACHIEVEMENT(g0 , g0)←0
S←S ∪ { g0 }
P←P ∪ { gu } gu is the goal in G-S with the smallest INDEX(g0, gu ) in I
While (G-S) and I are not empty do  // G-S is the minus of two sets
        gu ← the goal in G-S with smallest INDEX(COST( g0 , gu ), ACHIEVEMENT( g0 , gu)) in I
        S←S ∪ { gu }
        I ← I - { INDEX(g0, gu ) }
        For each goal gv ∈AGS(gu) do
                If INDEX(g0, gv)
                        > Findex(COST(g0, gu) + COST(gu, gv) + COST( gv),
                        ACHIEVEMENT(g0, gu) + ACHIEVEMENT(gv))
                        and IsTrusted (gv, cv)
                Then
                        COST(g0, gv) = COST(g0, gu) + COST(gu, gv) + COST(gv)
                        ACHIEVEMENT(g0, gv) = ACHIEVEMENT(g0, gu) + ACHIEVEMENT(gv)
                        INDEX(g0, gv) = Findex(COST(g0, gu) + COST(gu, gv) + COST( gv),
                                ACHIEVEMENT(g0, gu) + ACHIEVEMENT(gv))
                        If gu ≠ gn then
                                I ← I ∪ { INDEX(g0, gv ) }
                        End If
                        π(gv)← gu
                        l(gv)← l(gu)+1
                        If gu = gn then  // We have found a goal sequence
                                P(l(gn))← gn
                                For i←( l(gn)-1) to 0 do
                                        P(i)= π(P(i+1))
                                End For
                                P = {P(0),…,P(l(gn))}
                        End if
                End If
        End For
End while
```

Algorithm IsTrusted(g_v, c_v)

```
g ← gv
while g <> g0  do
        if g∈ cv then
                return false
        else
                g ← π(g)
        end if
end while
return true
end of algorithm
```

Computational Complexity of GSA3

Similar to the previous two algorithms, there are three major operations in the algorithm. However for updating the values of factors, it loops e times, where e is the number of arcs. Each time the complexity comes from checking IsTrusted, which will check whether the goals in the path from g_0 to the current goal contains the goals mentioned in the non-trusted set of the current goal. So the complexity to update the values has $O(n^2e)$ complexity.

So the complexity of the algorithm is $O(n^2 + n^2e + nel) = O(n^2e + ne^2)$.

The proof of correctness of the algorithm is similar to the proof of Theorem 3.1. The difference is that the optimal solution will be obtained according to both the index function and the IsTrusted checking. It is omitted here.

It is well known that the planning problem for the traditional discrete-state case is NP-hard [Baral, 2002]. The significant difference of goal selection algorithms of Goal Net from the traditional planning algorithms is its low complexity and anytime planning feature. The goal selection algorithms are any time algorithms that can be interrupted at any time during its execution and return a practical solution. Given a dynamic changing environments that an agent lives in, an anytime algorithm empowers the agent to respond immediately to environmental changes.

3.3.3 Action Selection between Goals

In a Goal Net, transitions contain many actions to drive the agent from one state to the next. Ideally, an agent can take a fixed sequence of actions to go from one state to the

next. But in the real world, the external environment is always changing. There are many internal or external factors affecting action selection. In order to move to the next state, an agent has to consider both internal factors and external factors to select actions. For example, one normally goes to work from home to office by bus. If it is raining or he is late, he may choose other more rapid transportation tools such as a taxi. If there is no taxi available, he may ask for somebody's help. So, external factors are important for action selection. As human beings, we can make decision to act according to the real situation by considering all the factors. As an agent, many factors are missing from the model. Only a few important factors are considered which makes it hard for an agent to select actions. Agents must be able to handle the uncertainty. In the above example, the way he finally chose to arrive at his office is uncertain. It depends on the situation he faces. Whether it is raining, he is leaving home late, taxi is not available, or whether he can find help from other sources are all uncertain factors. With incomplete information, human beings usually make decisions based on probability. For example, in the above situation, if the probability that a taxi is not available is low, he may choose to wait for a taxi. Otherwise, he may call his friends for help.

In this research, we propose an action selection method based on Bayesian networks [Heckermann, 95; Stephenson, 2000]. However, multiple action selection strategies are allowed in the proposed agent goal model.

3.3.3.1 Action Selection Strategies

In the proposed agent goal model, transitions are associated with tasks, each of which involves many actions. There are currently three types of transitions defined in the model: direct, conditional and probabilistic corresponding to three types of action selection

strategies: sequential execution, rule-based inference and probabilistic inference respectively.

- Sequential execution: This is the simplest situation. There is no action selection needed. Agents can move from one state to the next state by the execution of the fixed sequence of actions.

- Rule-based inference: In this situation, complete information for action selection is present. Agents can make decision according to the rules and current values of all the factors or states.

- Probabilistic inference: In this situation, information for action selection is not complete. A Bayesian network that represents the relationships between factors and actions can be constructed. The agent then reasons its actions through the Bayesian network inference.

In a transition object (P, V, F, T, K, r), *K* represents an action selection mechanism. It is defined as a tuple (D, F, t) where *D* is a knowledge base; *F* is a set of inference functions; and *t* is the type of action selection mechanism. For the fixed sequence of actions, the knowledge base is not necessary. For rule-based reasoning, the knowledge base contains the rules and variables about the action selection knowledge. Similarly, for Bayesian network reasoning, the knowledge base contains the Bayesian network model and the causal variables about the selection knowledge.

For each transition, a suitable action selection method is chosen according to the expected achievement. The fixed sequence actions method is the simplest way to fulfill tasks. There is actually no action selection involved. Rule-based reasoning simply applies

reasoning rules to choose a suitable task from the task list. These rules are represented as 'if <condition> then <do task>' statements. The 'if' portion represents the condition and the 'then' represents the actions that will be taken if the condition is satisfied. As we can see, rule-based representation of knowledge resembles some of the ways humans think. A rule-based reasoning mechanism searches for appropriate rules to fire.

Although rule-based action selection mechanisms can cater for some situations where an agent can obtain all the required factors to make a decision, the agent cannot obtain the entire knowledge about the dynamic running environment in most cases. For example, when an agent is running, some environment variables are not measurable or the changes of the variables cannot be predicted. In another situation, an agent may take actions based on the results of other agents' actions. The information about such factors is uncertain. Agents need to handle such situations and be able to take actions to get the best results for their goal pursuing. Therefore, three action selection strategies are proposed in this research to help agents select the right actions at the right time.

3.3.3.2 Action Selection in Probabilistic Transitions

Suppose $V = \{v_1, v_2, ..., v_n\}$ is a causal factor set including internal factors and external factors; $T = \{T_1, T_2, ..., T_k\}$ is a set of independent tasks, each of which contains a group of actions, that is $T_i = \{a_1, a_2, ..., a_m\}$ ($i = 1, 2, \ldots, k$) where a_j is the j^{th} action ($j = 1, 2, \ldots, m$) in the task T_i; and $R = \{(c, d) \mid c, d \in (V \cup T)\}$ is a set of causal relationships between

causal factors and tasks. So the probability of a task *Ti* is selected can be computed as following:

$$P(T_i \mid V \cup T) = \frac{P(V \cup T \mid T_i)P(T_i)}{\sum_{i=1}^{k} P(V \cup T \mid T_i)P(T_i)} \tag{3.2}$$

Since $\forall T_i \in T$ is independent with each other, the equation (3.2) can become:

$$P(T_i \mid V) = \frac{P(V \mid T_i)P(T_i)}{\sum_{i=1}^{k} P(V \mid T_i)P(T_i)} \tag{3.3}$$

$$= \frac{(\prod_{j=1}^{n} P(v_j \mid T_i))P(T_i)}{\sum_{i=1}^{k}(\prod_{j=1}^{n} P(v_j \mid T_i))P(T_i)} \tag{3.4}$$

The action selection through probabilistic reasoning is to find out an action group in the current situation that will give a satisfactory result.

Assuming the prior probabilities of internal factors, external factors, and tasks of a transition are known. When the transition is enabled, with the evidence of all the factors, the posterior probabilities can be computed. Then the action group with the biggest probability would be selected.

If the action group making the transition fire is not the suitable one, then learning is needed to adjust the prior probabilities of the factors.

3.3.3.3 Action Selection Algorithms

Action selection between two goals varies largely. It could be very complex, or very simple. In case we do not consider uncertainty, it is a matter of comparison of optional actions to select the one that could generate the smallest *INDEX*.

Suppose at goal g_i, the known factors are $COST(g_0, g_i)$ and $ACHIEVEMENT(g_0, g_i)$. $ACTION_{ij}$ are the actions that could transit to goal g_j from g_i. Different actions would result in different costs and achievements. $COST(g_i, g_j, Action_{ij})$ denotes the cost by taking the action $ACTION_{ij}$ while $ACHIEVEMENT(g_i, g_j, Action_{ij})$ denotes the achievement by taking the action $ACTION_{ij}$. The cost and achievement are not only affected by the existing cost and achievement of the action chosen, but also affected by certain conditions, which are represented by $CONDITION_{ij}$.

Action Selection Algorithm 1 - ASA1

Input: g_i, g_j, $COST(g_0, g_i)$, $ACHIEVEMENT(g_0, g_i)$, $Action_{ij}$
Output: Selected action
For each action $ACTION^k_{ij}$ // (k = 1, 2, …, n)
$Index'_k = Index(g_0, g_i) + F_{INDEX}(COST(g_0, g_i) + COST(g_i, g_j, Action^k_{ij}),$
$ACHIEVEMENT(g_0, g_i) + ACHIEVEMENT(g_i, g_j, Action^k_{ij}))$
End For
Select action $ACTION^k_{ij}$, where $Index'_k$ = Min $\{Index'_1, Index'_2, \ldots, Index'_n\}$

However, we cannot always assume that the environment is known or deterministic. For example, although we know taking a taxi is normally faster than taking public transport, if the traffic is heavy, taking public transport, such as the Subway could be faster.

Action Selection Algorithm 2 - ASA2

Input: g_i, g_j, $COST(g_0, g_i)$, $ACHIEVEMENT(g_0, g_i)$, $ACTION_{ij}$, $CONDITION_{ij}$
Output: Selected action
$Index' \leftarrow \infty$
For each possible $COST^{k1}(g_i, g_j)$
For each possible $ACHIEVEMENT^{k2}(g_i, g_j)$
$tempIndex' = Index(g_0, g_i) + F_{INDEX}(COST(g_0, g_i) + COST^{k1}(g_i, g_j),$

$\mathcal{ACHIEVEMENT}(g_{0,}g_i) + \mathcal{ACHIEVEMENT}^{k2}(g_i, g_j))$

If $temIndex' < Index'$ Then

$k1^* \leftarrow k1$

$k2^* \leftarrow k2$

$Index' \leftarrow temIndex'$

End If

End For

End For

For each Action $\mathcal{ACTION}^k_{ij}$

$\mathcal{P}^k_{ij} \leftarrow 0$

For each condition combination $condition_{ij}$

$\mathcal{P}^k_{ij} = \mathcal{P}^k_{ij} + \mathcal{P}(\mathcal{COST}^{k1^*}(g_i, g_j), \mathcal{ACHIEVEMENT}^{k2^*}(g_i, g_j) |$
$\mathcal{COST}(g_{0,}g_i), \mathcal{ACHIEVEMENT}(g_{0,}g_i), \mathcal{ACTION}^k_{ij}, condition_{ij})$

End For // Next condition combination

End For // Next Action

Select action $\mathcal{ACTION}^k_{ij}$, where $\mathcal{P}^k_{ij}$ = Max $\{ \mathcal{P}^1_{ij}, \mathcal{P}^2_{ij}, \ldots, \mathcal{P}^m_{ij} \}$

// m is the number of condition combinations

Action selection algorithm ASA2 attempts to find the action that could have the highest chance to have the most desirable cost and achievement. The following example illustrates the scenario.

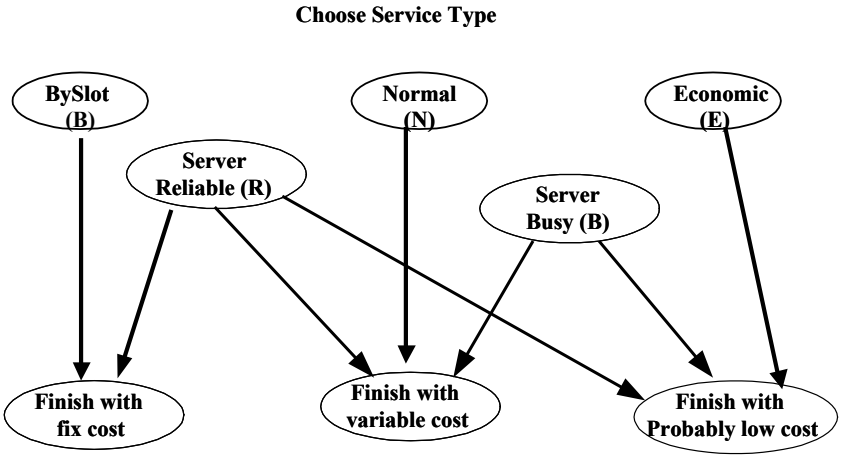

Figure 3.13 The type selection in grid computing services

In Figure 3.13, we show a grid node (server) which provides service in three ways: BySlot, Normal and Economic. We are not sure about the reliability of the server thus we put 0.5 for the probability, i.e. $\mathcal{P}$ (R) = 0.5. BySlot type of services is not affected by whether the server is busy or not. The services are assigned according to certain time slots. However, the other two types are affected, $\mathcal{P}$(B) = 0.05. Suppose we expect the job

to be finished without consideration of the cost. Algorithm ASA2 would select the service type that could have the highest chance to get the job finished.

Table 3.1 to Table 3.3 list the prior probabilities of the factors we have known.

E	B(H)	R(D)	P(finish | BRE)
T	F	F	0.02
T	F	T	0.8
T	T	F	0.01
T	T	T	0.9

Table 3.1 The prior probabilities of the Economic type of services

N	B(H)	R(D)	P(finish| BRN)
T	F	F	0.09
T	F	T	0.95
T	T	F	0.1
T	T	T	0.99

Table 3.2 The prior probabilities of the Normal type of services

S	R(D)	P(finish| RS)
T	F	0.01
T	T	0.9

Table 3.3 The prior probabilities of the BySlot type of services

We suppose the reliability and the busy status of the server are independent. According to ASA2, we have:

$$\begin{aligned}\mathcal{P}^{E}_{ij} = {} & \mathcal{P}(\text{finish}|\ \neg B\neg RE) * \mathcal{P}(\neg B) * \mathcal{P}(\neg R) + \\ & \mathcal{P}(\text{finish}|\ \neg BRE) * \mathcal{P}(\neg B) * \mathcal{P}(R) + \\ & \mathcal{P}(\text{finish}|\ B\neg RE) * \mathcal{P}(B) * \mathcal{P}(\neg R) + \\ & \mathcal{P}(\text{finish}|\ BRE) * \mathcal{P}(B) * \mathcal{P}(R) \\ = {} & 0.95*0.5*0.02+0.95*0.5*0.8+0.05*0.5*0.01+0.05*0.5*0.9 \\ = {} & 0.5*(0.95*0.82+0.05*0.91) \\ = {} & 0.41225\end{aligned}$$

$$\begin{aligned}\mathcal{P}^{N}_{ij} = {} & \mathcal{P}(\text{finish}|\ \neg B\neg RN) * \mathcal{P}(\neg B) * \mathcal{P}(\neg R) + \\ & \mathcal{P}(\text{finish}|\ \neg BRN) * \mathcal{P}(\neg B) * \mathcal{P}(R) + \\ & \mathcal{P}(\text{finish}|\ B\neg RN) * \mathcal{P}(B) * \mathcal{P}(\neg R) + \\ & \mathcal{P}(\text{finish}|\ BRN) * \mathcal{P}(B) * \mathcal{P}(R) \\ = {} & 0.5*(0.95+1.04+1.09*0.05) \\ = {} & 0.52125\end{aligned}$$

$$\mathcal{P}^{S}_{ij} = \mathcal{P}(\text{finish}|\ \neg RS) * \mathcal{P}(\neg R) + \mathcal{P}(\text{finish}|\ RS) * \mathcal{P}(R)$$

= 0.5*0.9+0.5*0.01
= 0.455

Obviously, choosing the normal service would have highest chance to achieve the goal.

Computational Complexity of ASA2

Basically, the algorithm consists of two nested loops. We assume that from goal g_i to goal g_j, there are n^1_{ij} types of conditions, n^2_{ij} types of achievements, n^3_{ij} types of costs, n^4_{ij} types of actions. Denote $n_1 = \underset{i,j}{Max} \{ n^1_{ij} \}$, $n_2 = \underset{i,j}{Max} \{ n^2_{ij} \}$, $n_3 = \underset{i,j}{Max} \{ n^3_{ij} \}$, $n_4 = \underset{i,j}{Max} \{ n^4_{ij} \}$, the computational complexity bound of ASA2 is $\mathbf{O(n^1_{ij} \times n^4_{ij} + n^2_{ij} \times n^3_{ij})}$ for goal g_i to goal g_j, or $\mathbf{O(n_1 \times n_4 + n_2 \times n_3)}$ for all.

3.3.3.4 Observation of Conditions

Alternatively, and more often, we would use observations to detect those factors that we do not have enough knowledge of. If there could not be such observations, we would apply learning mechanisms that build the knowledge along the experience accumulated.

Figure 3.14 shows a Bayesian network of the factors related to the action selection between goal g_i and goal g_j. Here we have some observations, $OBSERVATION_{ij}$ to estimate the conditions, where we do not know the values of the variables of $CONDITION_{ij}$.

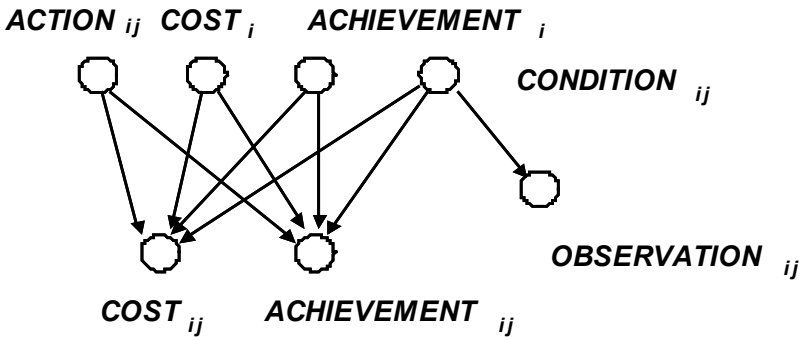

Figure 3.14 The action selection with observations of conditions.

For each combination of the possible $COST_{ij}^{*}$, $ACHIEVEMENT_{ij}^{*}$, the probability of having it with a certain action $ACTION_{ij}^{*}$ would be:

$$\sum_{condition \in CONDITION_{ij}} P(COST_{ij}^{*}, ACHIEVEMENT_{ij}^{*} \mid COST_i, ACHIEVEMENT_i, ACTION_{ij}^{*}, condition)$$
$$\times P(condition \mid OBSERVATION_{ij})$$
$$= \sum_{condition \in CONDITION_{ij}} P(COST_{ij}^{*}, ACHIEVEMENT_{ij}^{*} \mid COST_i, ACHIEVEMENT_i, ACTION_{ij}^{*}, condition)$$
$$\times (P(OBSERVATION_{ij} \mid condition) / \sum_{condition' \in CONDITION_{ij}} P(OBSERVATION_{ij} \mid condition'))$$

Similar to ASA2, by the above equations, we could work out the probabilities of all the possible costs and achievements incremental for each optional action. Therefore, according to the index function, a most desirable action could be selected.

3.3.3.5 Learning of the Probabilistic Action Selection

Assuming an agent will record all the historical data of its action selections in a knowledge base, when the achievement is not satisfied by the fulfillment of the task selected with the highest probability by the Bayesian network inference, the Bayesian network needs to be trained by the user in future.

In this research, we adopt the likelihood function to train the Bayesian networks. The data obtained during the agent running is the new evidence for the training. Together with previous historical data, the Bayesian network will be trained again. The knowledge base of the agent will be updated thereafter so that the agent will use the new knowledge in the next run.

Learning from experience is a way to have better action selection. While the experience accumulates, we expect the agent to have better estimation of the probabilities, and thus

make wiser choices. However, the learning in an uncertain environment has proven to be extremely difficult. To make the learning model affordable to software agents, lots of researchers fall back to the reactive model for action selection and the learning would be much easier, for example, the mapping between actions and sensed conditions. However, reactive models are too simple to model complex applications.

The Goal Net uses hierarchical structures and restricts action selection between two goals. We use the number of cases to represent our belief, which is denoted by *BN*. Obviously, the more cases we have, the better understanding we can derive. When the initial prior probabilities are given, we give the belief number as well. The more belief number is given, the stronger belief we have, and it is less likely to be affected by individual cases. On the contrary, the smaller belief number is given, the less certain we know about the factor and the more we rely on the new cases.

For example, in the example shown in Figure 3.12, we are not sure about the reliability of the server. While for the rest of the prior probabilities, we are quite happy about the estimation. Therefore, we expect that every time we choose a service type and based on the result, we would try to learn the reliability of the server.

Action Selection Condition Learning Algorithm - ASCLA

Input: g_i, g_j ,$COST(g_{0\,,}g_i)$, $ACHIEVEMENT(g_{0\,,}g_i)$, $ACTION_{ij}$, $CONDITION_{ij}$, BN_{ij} ,
$CONDITION^*_{ij}$, BN^*_{ij}
// $CONDITION^*_{ij}$ is the corresponding condition that to be learned and
// BN^*_{ij} is the belief number.
Output: Updated parameters
Index' ← ∞
For each possible $COST^{k1}(g_{i\,,}g_j)$
For each possible $ACHIEVEMENT^{k2}(g_{i\,,}g_j)$
tempIndex' = Index(g_0, g_i)+F_{INDEX}($COST(g_{0\,,}g_i)$ + $COST^{k1}(g_{i\,,}g_j)$,
$ACHIEVEMENT(g_{0\,,}g_i)$+ $ACHIEVEMENT^{k2}(g_{i\,,}g_j)$)
If temIndex' < Index' Then
$k1^*$ ← $k1$
$k2^*$ ← $k2$
Index' ← temIndex'

End IF
End For
End For
For each Action $ACTION^{k}_{ij}$
$P^{k}_{ij} \leftarrow 0$
For each condition combination $condition_{ij}$
$P^{k}_{ij} = P^{k}_{ij} + P(COST^{k1*}(g_i, g_j), ACHIEVEMENT^{k2*}(g_i, g_j) |$
$COST(g_0, g_i)$, $ACHIEVEMENT(g_0, g_i)$, $ACTION^{k}_{ij}$, $condition_{ij})$
End For // Next condition combination
End For // Next Action
Select action $ACTION^{k}_{ij}$, where P^{k}_{ij} = Max { P^{1}_{ij}, P^{2}_{ij}, …, P^{m}_{ij} }
// m is the number of condition combinations. Now update the belief
$P_{ij}(CONDITION^*_{ij}, \text{numerator}) \leftarrow 0$
For each condition combination $condition_{ij}$ except $CONDITION^*_{ij}$=*false*
$P_{ij}(CONDITION^*_{ij}, \text{numerator}) = P_{ij}(CONDITION^*_{ij}, \text{numerator}) +$
$P(COST^{k1*}(g_i, g_j), ACHIEVEMENT^{k2*}(g_i, g_j) | COST(g_0, g_i)$,
$ACHIEVEMENT(g_0, g_i)$, $ACTION^{k}_{ij}$, $condition_{ij})$
End For // Next condition combination
$P_{ij}(CONDITION^*_{ij}, \text{denominator}) \leftarrow 0$
For each condition combination $condition_{ij}$
$P_{ij}(CONDITION^*_{ij}, \text{denominator}) = P_{ij}(CONDITION^*_{ij}, \text{denominator}) +$
$P(COST^{k1*}(g_i, g_j), ACHIEVEMENT^{k2*}(g_i, g_j) | COST(g_0, g_i)$,
$ACHIEVEMENT(g_0, g_i)$, $ACTION^{k}_{ij}$, $condition_{ij})$
End For // Next condition combination
$P_{ij}(CONDITION^*_{ij}, \text{updates}) =$
$P_{ij}(CONDITION^*_{ij}, \text{numerator}) / P_{ij}(CONDITION^*_{ij}, \text{denominator})$
If job succeeded, update $P(CONDITION^*_{ij})$ with
$(BN^*_{ij} \times P(CONDITION^*_{ij}) + P_{ij}(CONDITION^*_{ij}, \text{updates})) / (BN^*_{ij} + 1)$
Else, i.e., not succeeded, update $P(CONDITION^*_{ij})$ with
$(BN^*_{ij} \times P(CONDITION^*_{ij}) + 1 - P_{ij}(CONDITION^*_{ij}, \text{updates})) / (BN^*_{ij} + 1)$
End If

In the example of Figure 3.13, after the agent choose Normal type of service, and if the job is successfully finished, we could follow the Action Selection Condition Learning Algorithm, ASCLA to update its belief to the reliability of the server.

Suppose $BN^*_{ij} = BN(R) = 20$

$P_{ij}(R, \text{numerator}) = P(\text{B}) \times P(\text{R}) \times P(\text{finish} \mid \text{BR}) + P(\neg\text{B}) \times P(\text{R}) \times P(\text{finish} \mid \neg\text{BR})$
$P_{ij}(R, \text{denominator}) = P(\text{BR}) \times P(\text{finish} \mid \text{BR}) + P(\neg\text{BR}) \times P(\text{finish} \mid \neg\text{BR})$
$+ P(\text{B}\neg\text{R}) \times P(\text{finish} \mid \text{B}\neg\text{R}) + P(\neg\text{B}\neg\text{R}) \times P(\text{finish} \mid \neg\text{B}\neg\text{R})$
$P_{ij}(\text{R}, \text{updates}) = P_{ij}(R, \text{numerator}) / P_{ij}(R, \text{denominator}) = 0.976$
$P(\text{R}) = (P(\text{R}) * BN(R) + P_{ij}(\text{R}, \text{updates})) / (BN(R) + 1) = (20*0.5+0.976)/21=0.5267$

We can see that a successful case would increase our confidence in the reliability of the server. Similarly, we could also update the probability if the job failed.

Computational Complexity of ASCLA

The algorithm consists of two loops for updating belief in various conditions. We still use the notations in ASA2, i.e. from goal g_i to goal g_j, there are n^1_{ij} combinations of conditions, n^2_{ij} types of achievements, n^3_{ij} types of costs, n^4_{ij} types of actions. Denote $n_1 = \underset{i,j}{Max}\ \{ n^1_{ij} \}$, $n_2 = \underset{i,j}{Max}\ \{ n^2_{ij} \}$, $n_3 = \underset{i,j}{Max}\ \{ n^3_{ij} \}$, $n_4 = \underset{i,j}{Max}\ \{ n^4_{ij} \}$, the computational complexity bound of the belief updating in ASCLA is $\mathbf{O(n^1_{ij} \times n^4_{ij} + n^2_{ij} \times n^3_{ij})}$ for goal g_i to goal g_j, or $\mathbf{O(n_1 \times n_4 + n_2 \times n_3)}$ for all.

Another situation that we might need to learn is the distribution of the conditional probability. For example, in Figure 3.13, we know the probabilities of server being busy and server reliability. However, we are not sure about the knowledge of how much the busy and reliable conditions could affect the chance of having the job finished. Again we use *BN* to represent the belief to existing knowledge. As conditional probability is linked to the target and its parents, the corresponding *BN* is to the target and its parents.

Action Selection Relationship Learning Algorithm - ASRLA

Input: g_i, g_j ,$COST(g_0, g_i)$, $ACHIEVEMENT(g_0, g_i)$, $ACTION_{ij}$, $CONDITION_{ij}$, BN_{ij}
Output: Updated parameters
Index' ← ∞
For each possible $COST^{k1}(g_i, g_j)$
    For each possible $ACHIEVEMENT^{k2}(g_i, g_j)$
        tempIndex' = Index(g_0, g_i)+F_{INDEX}($COST(g_0, g_i)$ + $COST^{k1}(g_i, g_j)$,
                $ACHIEVEMENT(g_0, g_i)$+ $ACHIEVEMENT^{k2}(g_i, g_j)$)
        If temIndex' < Index' Then
            $k1^*$ ← $k1$
            $k2^*$ ← $k2$
            Index' ← temIndex'
        End If
    End For
End For
For each Action $ACTION^k_{ij}$
    P^k_{ij} ← 0
    For each condition combination $condition_{ij}$
        $P^k_{ij} = P^k_{ij} + P(COST^{k1*}(g_i, g_j), ACHIEVEMENT^{k2*}(g_i, g_j) |$
            $COST(g_0, g_i)$, $ACHIEVEMENT(g_0, g_i)$, $ACTION^k_{ij}$,$condition_{ij}$)
    End For // Next condition combination
End For // Next Action
Select action $ACTION^k_{ij}$, where P^k_{ij} = Max { P^1_{ij} , P^2_{ij} , …, P^m_{ij} }
// m is the number of condition combinations. Now let's update our belief

For each condition $CONDITION^{k}_{ij}$
  $P_{ij}(CONDITION^{k}_{ij}$, numerator) ← 0
  For each condition combination $condition_{ij}$ except $CONDITION^{k}_{ij}$ =*false*
    $P_{ij}(CONDITION^{k}_{ij}$, numerator) = $P_{ij}(CONDITION^{k}_{ij}$, numerator) +
      $P(COST^{k1*}(g_i, g_j), ACHIEVEMENT^{k2*}(g_i, g_j) | COST(g_0, g_i)$,
      $ACHIEVEMENT(g_0, g_i)$, $ACTION^{k}_{ij}$, $condition_{ij}$)
  End For // Next condition combination
  $P_{ij}(CONDITION^{k}_{ij}$, denominator) ← 0
  For each condition combination $condition_{ij}$
    $P_{ij}(CONDITION^{k}_{ij}$, denominator) = $P_{ij}(CONDITION^{k}_{ij}$, denominator)
      + $P(COST^{k1*}(g_i, g_j), ACHIEVEMENT^{k2*}(g_i, g_j) | COST(g_0, g_i)$,
      $ACHIEVEMENT(g_0, g_i)$, $ACTION^{k}_{ij}$, $condition_{ij}$)
  End For // Next condition combination
  $P_{ij}(CONDITION^{k}_{ij}$, post) =
    $P_{ij}(CONDITION^{k}_{ij}$, numerator) / $P_{ij}(CONDITION^{k}_{ij}$, denominator)
End For // Next condition
If job succeeded,
  For each $ChildFamily^{k}_{ij}$
    For each combination of conditions $condition^{k'}_{ij}$ in the $ChildFamily^{k}_{ij}$
      Macro_Update_Success
    End For // combination of conditions
  End For // $ChildFamily^{k}_{ij}$
Else, i.e., job did not succeed
  For each $ChildFamily^{k}_{ij}$
    For each combination of conditions $condition^{k'}_{ij}$ in the $ChildFamily^{k}_{ij}$
      Macro_Update_Not_Success
    End For // combination of conditions
  End For // ChildFamily $^{k}_{ij}$
End If

Macro_Update_Success
  Let $P^{*k'}_{ij}$ denote P ($COST^{k1*}(g_i, g_j)$, $ACHIEVEMENT^{k2*}(g_i, g_j)$ |
    $COST(g_0, g_i)$, $ACHIEVEMENT(g_0, g_i)$, $ACTION^{k}_{ij}$, $condition^{k'}_{ij}$)
  $P^{*k'}_{ij}$(numerator) = $(P^{*k'}_{ij} \times BN^{k}_{ij} + \prod_{condition^{kr}_{ij} \in condition^{k}_{ij}} P(condition^{k'r}_{ij}))$
  $P^{*k'}_{ij}$(denominator) = $(BN^{k}_{ij} + \prod_{condition^{kr}_{ij} \in condition^{k}_{ij}} P(condition^{k'r}_{ij}))$
  Update $P^{*k'}_{ij}$ with $P^{*k'}_{ij}$(numerator) / $P^{*k'}_{ij}$(denominator)
End Macro_Update_Success

Macro_Update_Not_Success
  Let $P^{*k'}_{ij}$ denote P ($COST^{k1*}(g_i, g_j)$, $ACHIEVEMENT^{k2*}(g_i, g_j)$ |
    $COST(g_0, g_i)$, $ACHIEVEMENT(g_0, g_i)$, $ACTION^{k}_{ij}$, $condition^{k'}_{ij}$)
  $P^{*k'}_{ij}$(numerator) = $P^{*k'}_{ij} \times BN^{k}_{ij}$
  $P^{*k'}_{ij}$(denominator) = $(BN^{k}_{ij} + \prod_{condition^{kr}_{ij} \in condition^{k}_{ij}} P(condition^{k'r}_{ij}))$
  Update $P^{*k'}_{ij}$ with $P^{*k'}_{ij}$(numerator) / $P^{*k'}_{ij}$(denominator)
End Macro_Update_Not_Success

For the example in Figure 3.13, after the Normal type of service is chosen, if the job is successfully finished, we could follow the Action Selection Relationship Learning

Algorithm, ASRLA to update its belief to the relationship of the reliability of the server, the busy status of the server and the probability of the job being finished.

Again, suppose BN =20

$$\mathcal{P}_{ij}(\mathcal{R}, \text{numerator}) = \mathcal{P}(\text{B}) \times \mathcal{P}(\text{R}) \times \mathcal{P}(\text{finish} \mid \text{BR}) + \mathcal{P}(\neg\text{B}) \times \mathcal{P}(\text{R}) \times \mathcal{P}(\text{finish} \mid \neg\text{BR})$$
$$\mathcal{P}_{ij}(\mathcal{R}, \text{denominator}) = \mathcal{P}(\text{BR}) \times \mathcal{P}(\text{finish} \mid \text{BR}) + \mathcal{P}(\neg\text{BR}) \times \mathcal{P}(\text{finish} \mid \neg\text{BR}) + \mathcal{P}(\text{B}\neg\text{R}) \times \mathcal{P}(\text{finish} \mid \text{B}\neg\text{R}) + \mathcal{P}(\neg\text{B}\neg\text{R}) \times \mathcal{P}(\text{finish} \mid \neg\text{B}\neg\text{R})$$
$$\mathcal{P}_{ij}(\mathcal{R}, \text{post}) = \mathcal{P}_{ij}(\mathcal{R}, \text{numerator}) / \mathcal{P}_{ij}(\mathcal{R}, \text{denominator}) = 0.976$$
$$\mathcal{P}_{ij}(\mathcal{B}, \text{numerator}) = \mathcal{P}(\text{B}) \times \mathcal{P}(\text{R}) \times \mathcal{P}(\text{finish} \mid \text{BR}) + \mathcal{P}(\text{B}) \times \mathcal{P}(\neg\text{R}) \times \mathcal{P}(\text{finish} \mid \text{B}\neg\text{R})$$
$$= 0.05\times0.5\times 0.9+0.05\times0.5\times0.01 = 0.05\times0.5 \times 0.91 = 0.02275$$
$$\mathcal{P}_{ij}(\mathcal{B}, \text{denominator}) = \mathcal{P}(\text{BR}) \times \mathcal{P}(\text{finish} \mid \text{BR}) + \mathcal{P}(\neg\text{BR}) \times \mathcal{P}(\text{finish} \mid \neg\text{BR}) + \mathcal{P}(\text{B}\neg\text{R}) \times \mathcal{P}(\text{finish} \mid \text{B}\neg\text{R}) + \mathcal{P}(\neg\text{B}\neg\text{R}) \times \mathcal{P}(\text{finish} \mid \neg\text{B}\neg\text{R})$$
$$= 0.05 \times 0.5 \times 0.9 + 0.05 \times 0.5 \times 0.01 + 0.95 \times 0.5 \times 0.8 + 0.95 \times 0.5 \times 0.02$$
$$= 0.5 * (0.91 * 0.05 + 0.95 * 0.82) = 0.5 * 0.8245 = 0.41225$$
$$\mathcal{P}_{ij}(\mathcal{B}, \text{post}) = \mathcal{P}_{ij}(\mathcal{B}, \text{numerator}) / \mathcal{P}_{ij}(\mathcal{B}, \text{denominator}) = 0.05518$$

Now let us update our belief:

$$P(\text{finish} \mid \text{BR}) = (P(\text{finish} \mid \text{BR}) * BN + P_{ij}(\text{R}, \text{post}) * P_{ij}(\text{B}, \text{post})) / (BN + P_{ij}(\text{R}, \text{post}) * P_{ij}(\text{B}, \text{post})) = 0.990026855523875$$
$$P(\text{finish} \mid \neg\text{BR}) = (P(\text{finish} \mid \neg\text{BR}) * BN + P_{ij}(\text{R}, \text{post}) * (1-P_{ij}(\text{B}, \text{post}) / (BN + P_{ij}(\text{R}, \text{post}) * (1-P_{ij}(\text{B}, \text{post}))) = 0.952203751933588$$
$$P(\text{finish} \mid \text{B}\neg\text{R}) = (P(\text{finish} \mid \text{B}\neg\text{R}) * BN + (1-P_{ij}(\text{R}, \text{post})) * P_{ij}(\text{B}, \text{post})) / (BN +(1- P_{ij}(\text{R}, \text{post})) * P_{ij}(\text{B}, \text{post})) = 0.990026855523875$$
$$P(\text{finish} \mid \neg\text{B}\neg\text{R}) = (P(\text{finish} \mid \neg\text{B}\neg\text{R}) * BN + (1-P_{ij}(\text{R}, \text{post})) * (1-P_{ij}(\text{B}, \text{post}))) / (BN +(1- P_{ij}(\text{R}, \text{post})) * (1- P_{ij}(\text{B}, \text{post}))) = 0.0910305749905649$$

So, the results are listed in Table 3.4.

N	B(H)	R(D)	P(finish| BRN)	Updated P(finish| BRN)
T	F	F	0.09	0.0910305749905649
T	F	T	0.95	0.952203751933588
T	T	F	0.1	0.100059590454158
T	T	T	0.99	0.990026855523875

Table 3.4 Updated prior probabilities of the Normal type of services

Similarly, if the job is unsuccessful, we could update the belief as well. We would expect the probability of the job to be finished to decrease in this case.

Computational Complexity of ASRLA

The computation complexity of the algorithm ASRLA is higher than that of ASCLA because the belief under various conditions should be updated accordingly. Again, we use

the notations in ASA2 that from goal g_i to goal g_j, there are n^1_{ij} types of conditions (note that the action is selected and thus is known, and so do the costs and achievements). We use the same denotation of the algorithm ASCLA, the computational complexity bound of ASRLA is $\mathbf{O}(n^2_{ij} \times n^3_{ij} + n^1_{ij} \times n^4_{ij} + n^1_{ij} \times n^1_{ij} + n^1_{ij} \times n^1_{ij} \times n^1_{ij}) = \mathbf{O}(n^1_{ij} \times n^1_{ij} \times n^1_{ij})$ for goal g_i to goal g_j, or $\mathbf{O}(n_1^3)$ for all. However, this estimation could be improved, as in a Bayesian network, the probability distribution of a node is only affected by its family, that is, the nodes that have direct connections with the node. Suppose the family that has the maximum condition members has m condition members. The computational complexity is bounded by $\mathbf{O}(n_1^2 \times m)$. Normally, m is much smaller than n_1.

Besides the above three actions selection strategies, Goal Net is open for new types of action selection strategies and user defined action selection strategies. For instance, a new action selection algorithms based on Fuzzy Cognitive Maps [Kosko, 86] has been being developed, whose details are omitted here.

3.4 Modeling Multi-Agent System with Goal Net

When a problem is complex and a single agent cannot handle it with an acceptable performance, a multi-agent system will be considered. In a multi-agent system, a number of agents collaborate with each other to solve the problem towards a common goal in a complex and dynamic changing environment. Therefore issues such as how to model, identify, organize, and manage these agents need to be addressed. In this section, we illustrate how Goal Net can be used to model a multi-agent system. In particular, we present a systematic method to model the process of refining goals, identifying agents, and agent coordination.

Agents in a MAS environment are not isolated, they need to share common knowledge, goals and collaborate with each other in order to reach a common goal. Agents in a goal-oriented multi-agent system are modeled by the proposed Goal Net.

Agent identification and agent coordination are the most important issues of multi-agent modeling. In the next two sub-sections, we address these two issues by structurally extending the Goal Net that was proposed in the previous sections.

3.4.1 Agent Identification

After an application problem is modeled with the Goal Net, multiple agents can be identified from the goal model. As described, a Goal Net is depicted as a hierarchical structure, Goal Net. Each composite state in a Goal Net can be further decomposed into a number of subnets. Therefore, a subnet consists of a composite state together with its decomposition. When a Goal Net is too complex to be realized by a single agent, a subnet can be used to form the goal model of a new agent. As a result, a multi-agent system can be derived from a Goal Net. Agents in a multi-agent system modeled by the Goal Net, are organized in a hierarchical structure, namely *agent hierarchy*.

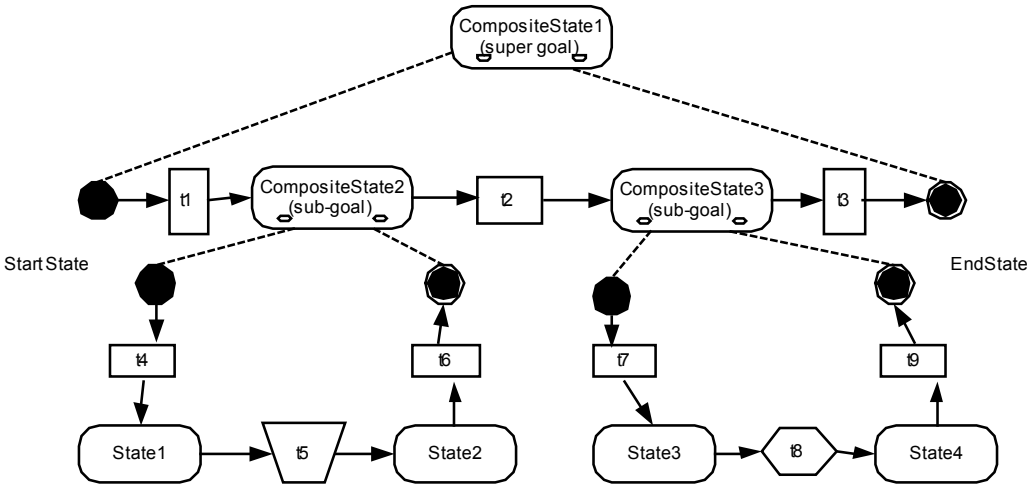

Figure 3.15 A Goal Net

For example, a Goal Net shown in Figure 3.15 consists of three composite states, *A*, *B*, and *C*, leading three subnets *A*, *B*, and *C*. The three subnets can be identified as goal models for the three new derived agents as shown in Figure 3.16. Based on the original Goal Net, the generated agents form an agent hierarchy.

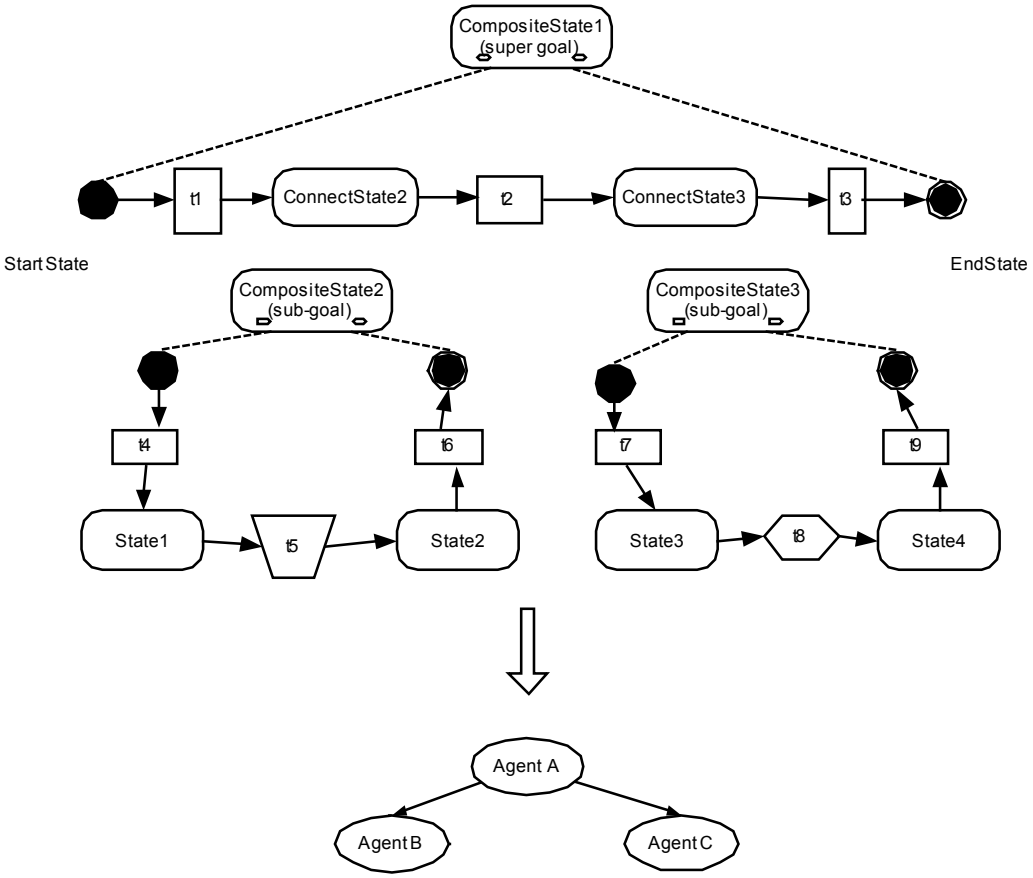

Figure 3.16 The derived agent hierarchy from the goal model shown in Figure 3.15

If the composite state is in a higher layer of the original Goal Net, the derived agent will also be in the higher layer of the generated agent hierarchy. If the composite state is at the same layer with another composite state, the two derived agents will be at the same layer in the generated agent hierarchy. The dashed arrows indicate the hierarchical relationships between agents. As a result, in Figure 3.16, Agent *A* becomes the parent agent of agent *B* and *C*.

When a composite state together with its decomposition in a Goal Net is selected to form the goal model of a new agent, it becomes the root node of the new goal model and this composite state becomes the goal of the new agent. In the original goal model, the place

for the split composite state will be replaced by a *connection* state. *A connection* state is an atomic state, which logically connects the goal model to another goal model. So, in Figure 3.15, when the sub nets *B* and *C* are split out, the connection state B' and *C*' will take the places of the composite states *B* and *C* respectively. The Goal Net *A* will then connect Goal Net *B* or Goal Net *C* through the corresponding state *B*' or *C*' respectively.

The number of agents to be generated from a Goal Net will be decided by strategies. It is not necessary to select each subnet in the entire Goal Net to generate a new agent. Following lists some strategies for identifying agents:

- Based on similarity of the goals (for example, assign the goals related to data collection as a data collection agent).
- Based on roles that the identified agents will play (for example, the data collection agent, the forecasting agent in business forecasting system).
- Based on locations that the identified agents will run in (for example, the agents will run in a distributed environment in different organizations).
- Based on specific types of agents (for example, some goals will be achieved by mobile agents).
- Based on functional systems that the identified agents will belong to (for example, agents for different sub-systems).
- Based on load balancing (for example, we can balance the number of sub-goals in the identified agents to reduce the complexity of one agent).
- Based on other concepts of real agent applications.

In following, the algorithm to split a Goal Net for agent identification is given.

The Split Goal Net Algorithm - SGNA

```
Input: Goal Net G
Output: Goal Net set Gs
    Gs ← Φ
    l ← number of layers – 1;
    For i = l; i>0; i--; Loop
            For each composite state s in layer l
                If the sub net N rooted by s is split, Then
                        Gs = Gs ∪ N
                        Split N from G
                        Create a connection state s' to replace s in G
                        Set profile of s in s'
                End If
            End For
    End For
End
```

For a complex problem, the above strategies can be used in different levels. For instance, at the high level, we identify the agents based on the functional systems; at the middle level, we identify the agents based on the locations; and at the low level, we do it based on the roles or based on the load balancing strategy.

3.4.2 Coordination

Agents identified by the proposed method are organized in a hierarchical structure, and form an agent hierarchy. The higher level of agent becomes a coordinator of lower level of agents, and at the same time, it is pursuing its own goals. The goal represented in the original model becomes the common goal of the derived multi-agent system. The transitions between states define the coordination tasks and schedules. The original Goal Net becomes the coordination model.

A coordinator agent assigns a sub-goal to a child agent through a connection state. The position of the connection state in the goal model indicates the synchronization point

between the coordinator and the child agent. The original goal model assures the common goal will be achieved if the derived agents achieve their goals. In detail, when a coordinator runs to a connection state. A sub-goal will be assigned to the child agent pointed by the connection state. The child agent will then initialize its goal model with this assigned goal to start pursuing this goal. After the goal is reached, the child agent will notify the original agent. The original agent will exchange information with the child agent and then proceed its goal pursuit. For example, suppose composite state *B* is a state of decomposition of state *A*. The state *B* together with its decomposition forms a model of a new agent *B*. The state *A* together with the other members of its decomposition forms agent *A*. The agent *A* is then called the parent agent of agent *B* while the agent *B* is called the child agent of the agent *A*. The agent *A* has its own goal. It needs agent *B*'s work to reach its own goal. It coordinates the child agents according to its goal model. The child agent starts to work when it is assigned a goal from its parent agent. The goal of agent *A* can be regarded as the common goal of the multi-agent system.

One agent can be assigned two or more goals at the same time to improve the efficiency. When a parent agent assigns a sub-goal to its child agent, it does not wait. Instead, it can save the state of the current work and continue to do another work, that is, to pursue another goal. After the child agent reaches its goal, it will notify its parent agent. The parent agent will restore the saved work and proceed to pursue this goal.

In summary, by extending the Goal Net for multi-agent system modeling, agents can be identified easily and the coordination relations and schedules can be derived.

3.5 Summary

This chapter proposed a Goal Net for modeling complex goals of agents. The dynamic goal relationships and their representation with the proposed goal model were elaborated. Furthermore, the goal measurement and the properties of the goal model were discussed. Goal Net not only models the dynamic relationships between, goals, tasks, and environment, but also incorporates flexible learning/reasoning mechanisms for autonomous goal selection and action selection in a open, distributed and dynamic changing environment. In summary, this chapter introduced a novel goal-oriented modeling method for modeling agent as well as multi-agent system.

Compared to the existing task-oriented goal model, state-oriented goal model, and other goal-oriented modeling methods (KAOS, Tropos, Goal Tree etc.). Goal Nets not only model the static and structured goal relationships but also model the dynamic goal relationships during goal pursuit process in a changing environment. Following is a comparison based on the characteristics of agents' goals.

- Interactive: Neither the task-oriented goal model nor the state-oriented goal model supports the dynamic interactions between goals. Although the goal hierarchy defines the relationships between goals in terms of logical relationships, and parent-child relationships, it does not model the dynamic interactions between goals in a changing environment. The proposed Goal Net provides ways to define not only the logical and hierarchical relationships between goals but also the dynamic interactions between goals.

- Measurable: Goal measurement is important to the goal reasoning of an agent. Both the task-oriented goal model and the state-oriented goal model measure the goal either reached or not reached. It is not clear how goals are measured with the goal hierarchy model. The Goal Net introduces a mechanism for goal measurement so that an agent can reason its next goal based on the measurement and the agent execution can be monitored.

- Temporal: By introducing the time dimension into the goal model, an agent can measure its performance and make decision during its goal pursuit. For example, if an agent did not reach its goal within the required time period, its achievement would be very low or 0 even if it finally reached the goal. The Goal Net considers the time concept as a kind of cost in the goal measurement and uses it in the goal reasoning and action selection.

- Flexible/Adaptable: The agent running environment keeps changing. The goal an agent is to pursue should be decided dynamically and the actions the agent will take to pursue the goal should be selected according to the current states of the running environment. Unlike the existing goal models, the Goal Net supports flexible reasoning and learning mechanisms for the goal selection and the action selection in adapting a changing environment. It also considers the future states in the computation to generate the "optimal" solutions.

- Decomposable/re-combinable: A complex goal can be decomposed to many sub-goals to reduce the complexity. A Goal Net can be re-used and re-combined into other forms of Goal Net.

- Fuzzy: In a Goal Net, the next goal is selected based on the goal measurement including achievement, cost, time and other factors. In the case that goal measurement cannot be represented precisely, say the achievement, fuzzy concept is used in the Goal Net to handle the "fuzzy" factors.

- Partial: Unlike other existing goal models, partial goal can be computed based on the goal measurement mechanism. The partial goal computation is important in the agent monitoring, goal selection and achievement calculation.

This chapter also presents a series of anytime algorithms for agent reasoning and learning. Based on these algorithms, an agent can make its execution plan for achieving its overall goal at anytime including when it is interrupted. An agent can immediately revise its execution plan according to changes in the dynamic environment. An additional benefit is that anytime algorithms are ideally suited for integration into hybrid agent architectures that will be presented in the next Chapter.

In the last part of this chapter the Goal Net is used for modeling and designing multi-agent systems. It illustrates that the Goal Net is not only used to model the goals of single agents but also used to model the multi-agent systems including agent identification and coordination. The coordination schedule can also be derived from the goal model.

In the next chapter, an agent model for goal-based agents will be proposed based on the Goal Net. The agent development framework will also be discussed.

CHAPTER 4

FROM AGENT MODELING TO AGENT DESIGN AND IMPLEMENTATION

Currently, there is still a gap between agent mental models and agent implementations. One of the major reasons is that research on narrowing the gap between agent mental models and agent implementation is rare. To bridge the gap, a goal-oriented agent model is proposed based on Goal Net. A multi-agent development environment (MADE) has also been developed for facilitating the design and implementation of agent-oriented systems in this research.

The main objective of proposing the agent model and multi-agent framework based on Goal Net is to present goal-oriented agent modeling from theory to practice. Section 4.1 proposes the goal-oriented agent model based on the Goal Net theory. It is followed by the agent design model given in the Section 4.2. Section 4.3 describes the agent workflow and summarizes the characteristics of the proposed agent. A multi-agent model is given in Section 4.4 and a multi-agent development environment is introduced in Section 4.5.

Section 4.6 and Section 4.7 present a reusable architecture for the construction of an agent system and a multi-agent system respectively. Finally, this chapter is summarized in Section 4.8.

4.1 A Goal-oriented Agent Model

In Chapter 2, various agent definitions and a list of well-accepted agent characteristics have been reviewed. In short, an agent acts autonomously towards its goal. Traditionally, an agent mainly consists of the following tasks, which are executed repeatedly and form a Perceive-Reason-Act (PRA) cycle [Huhns, 97]:

1) **Perceive**: The agent perceives its environment continuously to sense any new situations.

2) **Reason**: The agent infers actions based on its goal, knowledge, and the perception of its environment.

3) **Act**: The selected actions are executed.

By evaluating the reactive and deliberative agents in Chapter 2, we have drawn a conclusion that for most real world problems, neither a purely deliberative agent nor a purely reactive agent is appropriate [Jennings, 98; Sycara, 98].

A reactive agent responds immediately to its percepts based on so-called *condition-action rules*. Humans have many such reflexes, for example, closing the eyes when something is approaching them. The whole knowledge of the agent is then encoded into these rules.

Although such agents can be implemented very efficiently, their range of applicability is very narrow. Even for very simple environments, the need for an *internal state* arises to keep track of specific information, when the complete access to the environment is not guaranteed.

If a reactive agent is extended by some sorts of goal planning, the agent can then combine its perception with the possible outcome of its actions in order to choose actions, which lead towards the goal state. A goal-based agent has a planning unit to find the appropriate sequence of actions to reach its goal. There are two levels of agent autonomy, behavior autonomy and goal autonomy. However, most of the existing effort on goal-based agents is focused on behavior autonomy which assumes that the goals and the environment remain unchanged. However, just behavior autonomy is not enough for agents to operate in a complex, dynamic environment. Most of the hybrid agent architectures proposed separates the planning and reaction in different layers. If there are no environmental changes, the agent will act as a pure classic planning agent, otherwise, it will shift to a pure reactive agent to respond to the environmental changes. A major drawback is that: for many real world problems, an agent has only incomplete information about the environment. It is often impossible to build the reactive rules that fully cover the uncertain environments.

An agent needs to reason and decide by itself how to select the next goal and what actions it should take in a real world environment so that its goal is attended to successfully. An agent should be able to improve its behaviors over time, that is, it becomes better with experience at selecting the next goal and achieving the goal by taking correct actions. Hence, an autonomous agent should present not only behavior autonomy but also goal autonomy.

In Chapter 3, we have shown that Goal Net models the dynamic relationships among goals, tasks/actions and environment changes. Goal Nets incorporate flexible learning/reasoning mechanisms within Goal Net that provides a theoretic basis for a seamless integration of dynamic planning and reaction. In this section, we propose a novel goal-oriented agent model based on Goal Net that supports both behavior autonomy and goal autonomy. Based on Goal Net, agents are not only able to choose the right action towards its current goal, but also able to reason the next goal to pursue towards its overall goal. Moreover, at anytime, an agent always has a practical plan to reach its goals.

[**Definition 4.1**] From the logic view, an goal autonomous agent can be formally specified as a tuple, GAA = (S, A, M, FG, FA, K, R, E), where

S is a set of states defining agent goals,

A is a set of actions defining agent behaviors,

M is an agent goal model represented by the composite state goal model, i.e. Goal Net,

K is the knowledge of the agent,

R is a set of situation-action rules defining reactive behavior of the agent,

FG is the functions for goal selection defining goal autonomy of the agent,

FA is the functions for action selection defining behavior autonomy based on the selected goals,

E is the agent environment that the agent lives in and perceives.

A goal autonomous agent mainly consists of the following tasks, which are executed repeatedly and form a PR^2A cycle:

1. **Perceive**: The agent perceives its environment continuously to sense any new situations.

2. **Reason for goal selection**: The agent infers the next goal, based on its goal model, knowledge, and the perception of its environment.

3. **Reason for action selection**: The agent infers actions based on the selected goal, knowledge, and the perception of its environment.

4. **Act**: The selected actions are executed.

The goal selection and action selection are done by the functions *FG* and the functions *FA*. The functions *FG* decide the next goal of the agent, based on the goal selection algorithms of the Goal Net. The functions *FA* decide the next task the agent needs to perform, based on the action selection algorithms of the Goal Net. Most of the current agent models predict the next goal (state) of agent based on the past states:

$$S_{n+1} = f(S_1, S_2, \ldots S_n) \tag{4.1}$$

In our proposed agent model, the next state of an agent is a function of the past states and the predicted future states:

$$S_{n+1} = f(S_1, S_2, \ldots S_n, pS_{n+1}, pS_{n+2}, \ldots pS_{n+k}) \tag{4.2}$$

where pS_{n+1} is the predicted state of S_{n+1}, $k>0$.

There are two types of actions, reactive actions and goal-based actions. The reactive actions are selected based on the condition-action rules. The goal-based actions are selected by the functions *FA*, whose internal algorithms are presented in Chapter 3.

The goal-oriented agent model is a hybrid model. It integrates both high-level goal-based reasoning based on the Goal Net and low-level reactive rule-based reasoning based on the condition-action rules in dynamic goal pursuing processes. By supporting from anytime planning algorithms and different reasoning mechanisms during the goal pursuing processes the goal autonomous agent is enabled to act as an "anytime" agent. No matter how complex the environment is, there is always a goal directed action available for the agent to execute.

From the structural view, the goal-oriented agent model consists of eight units, which include *perception* unit, *goal process unit*, *control* unit, *action* unit, *communication* unit, *knowledge* unit, *compute* unit and *data* unit as shown in Figure 4.1.

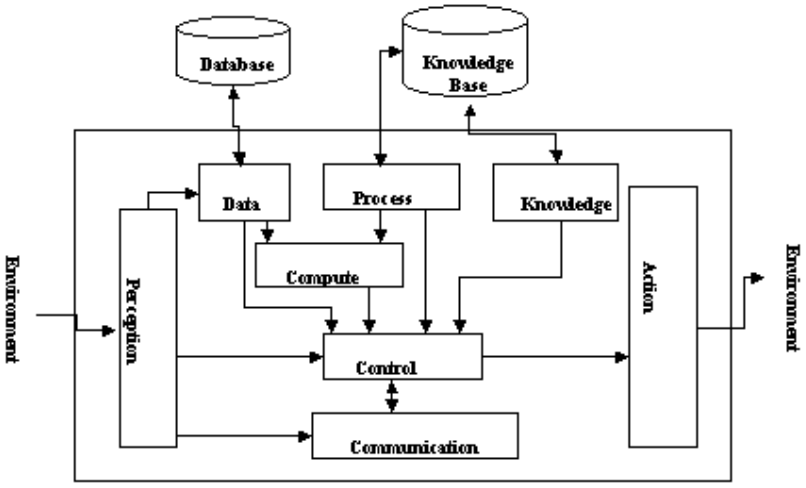

Figure 4.1 The goal-oriented agent model

- The perception unit senses the agent environment. It contains a list of states to indicate the status of the environment. If the environment changes, the perception unit will update the database through the data unit and notify the control unit to take actions against the change.

- The goal process unit defines the goal(s) of an agent and their relationships (Goal Nets). The Goal Nets will be loaded into the goal process unit from the knowledge base after the agent starts to work.

- The action unit contains all the actions the agent might perform.

- The compute unit makes decisions on both agent goal selection and action selection based on selected goal.

- The control unit coordinates all the other units to cooperate for the agent behavior. Moreover, if the perception unit of an agent detects an environment change, the control unit will enable the reactive action against the changes.

- The knowledge unit maintains the domain knowledge of the agent, which is used to handle the real world problems.

- The data unit defines data access mechanism to data resources.

- The communication unit defines the communication mechanism between agents.

A goal autonomous agent works as a continuous process that combines both reaction activity and planning activity. On one hand, the agent keeps sensing the environment. On the other hand, the agent keeps inferring its next goal and actions based on the current situation using the goal selection and action selection algorithms. No matter how much

the agent has already computed, there is always an action available for the agent to execute. The action is either a goal-based action or a reactive action.

This is achieved by improving the goal selection and action selection algorithms iteratively. An agent is always trying to pursue its current goal while inferring its next goal. In the case there is an unexpected change in the agent environment, a reactive plan will be available. The occurrence of such unexpected events will be used to improve and optimize the agent's goal model and the reasoning algorithms. The more time available for the agent's computation, the more intelligent the behavior will be. Furthermore, the iterative improvement enables the agent to easily adapt the action plan to unexpected situations. In summary, the goal-oriented agent model presents both behavior autonomy and goal autonomy with "anytime" agent feature.

4.2 From Agent Model to Agent Design

Despite the significant progress in the field of agent research and development, there is still a lack of widespread development and deployment environment of intelligent agent systems and multi-agent systems. In this research, a re-usable agent framework has been developed for designing and construction of goal autonomous agents.

We adopt Unified Modeling Language (UML) [Rumbaugh, 91] to illustrate the agent development framework. As described in the last section, a goal autonomous agent consists of eight basic units, which include *goal process unit*, *action* unit, *perception* unit, *communication* unit, *control* unit, *compute* unit, *data* unit and a *knowledge* unit. The agent structure is shown in Figure 4.2. The detailed diagram of each unit in UML is illustrated in Figure 4.3.

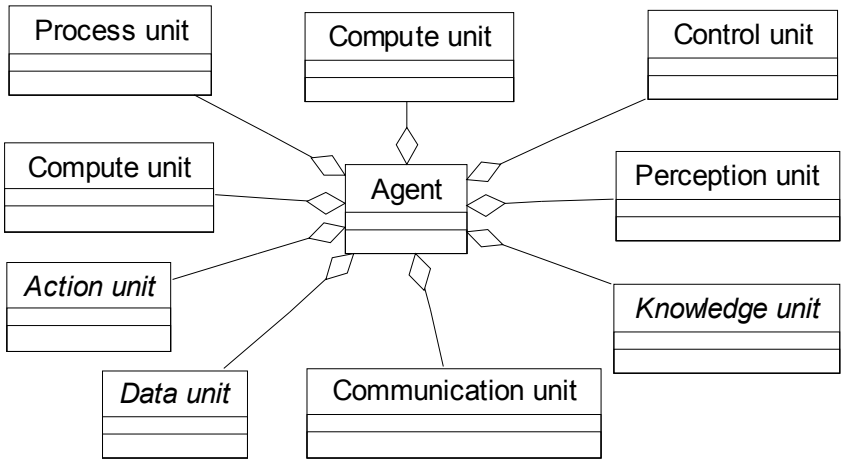

Figure 4.2 The agent structure

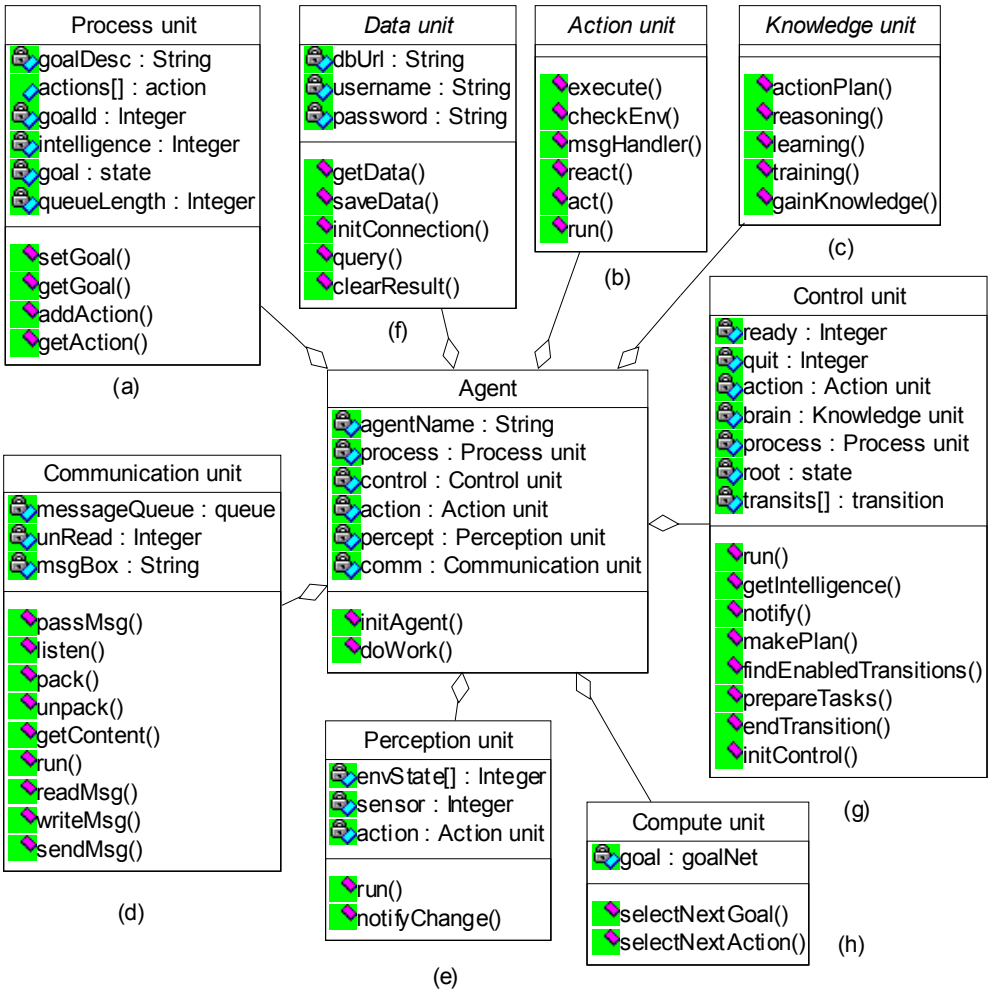

Figure 4.3 The components of a goal autonomous agent

- **Goal process unit**

The goal process unit is an abstraction of a real process in an agent goal pursuit life cycle. It defines the objective and tasks of the agent. In other words, the goal process unit defines the goal of the agent. Figure 4.3(a) describes the class definition of a goal process unit. After the agent starts, a goal *Id* is assigned to the agent. The Goal Nets will then be loaded to the unit from the knowledge base. The array *queue* simulates an action queue

for task fulfillment. The selected actions will be put into this queue for execution. The action unit will pick the actions from this queue and execute them one by one.

Figure 4.4 shows the structure of the class *goalNet*, which defines the agent goal model. As showed in the figure, each Goal Net is composed of state, transition, arc, and branch objects. The detailed definition of the classes is listed in Figure 4.5.

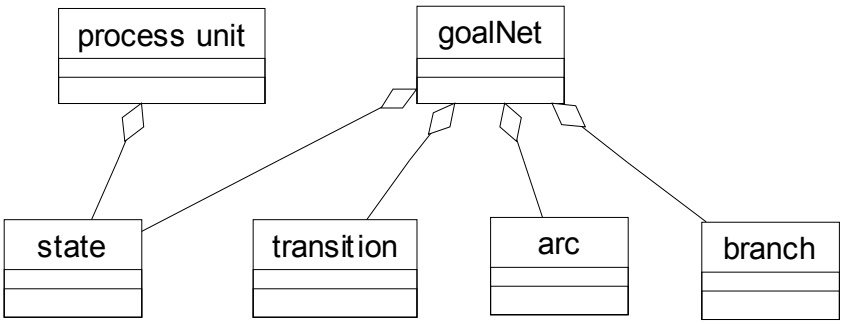

Figure 4.4 The diagram of Goal Net

Class *goalNet* contains a Goal Net that represents a goal or a sub goal of an agent. Each goalNet object has a unique id. A composite state uses this id to refer a particular goalNet object. The goalNet object has the references to the root state object , which represents the goal, and the branch objects of the Goal Net. The attributes *rootId*, *startId*, and *endId* are the identifiers of the root state object of the Goal Net, the start state object and the end state object of the decomposition of the root state. The function *getGoalNet()* retrieves the Goal Net definition from the knowledge base, based on the *Id* of the Goal Net. The function *loadGoalNet()* retrieves all the Goal Nets in a hierarchical Goal Net structure. The function *printGoalNet()* prints the structure of the Goal Net in text format. The function *init()* initializes the Goal Net while the function *reset()* resets the Goal Net. A

goal that is represented by a Goal Net can be assigned to an agent by passing the goalNet *Id*.

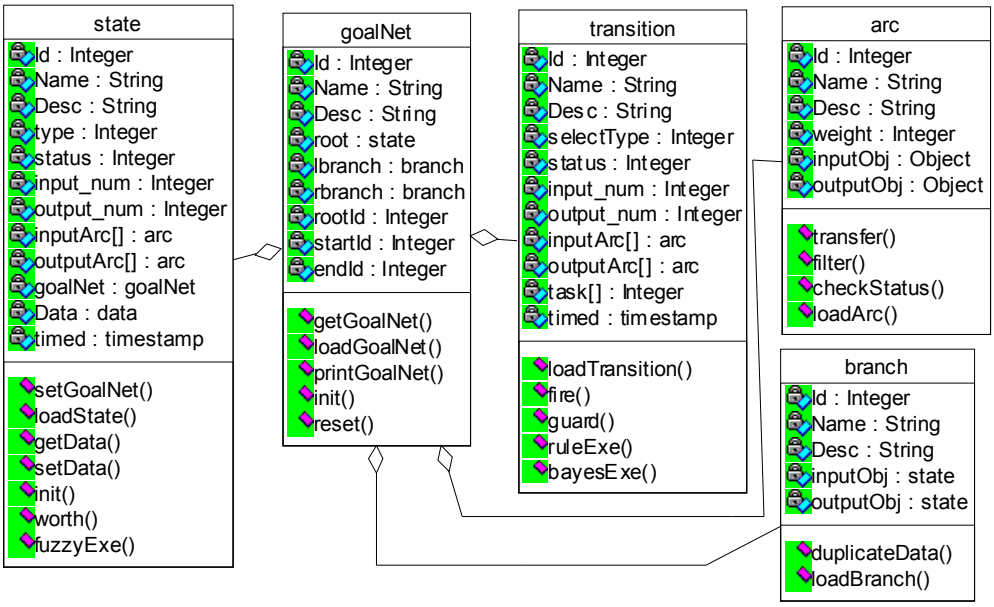

Figure 4.5 The class definitions of Goal Net

Class *state* defines all the profile variables and functions. Application specific variables and functions can be defined by extending this class in real agent construction. The attribute *type* indicates the state object is *atomic*, *composite* or *connect*. The attribute *status* indicates the status of the state. The number of input arcs and output arcs and the references to those arcs are stored in the corresponding variables. The attribute *goalNet* is used to store the reference of a Goal Net if the state object is a composite state. The function *setGoalNet()* initializes and loads Goal Nets if the state is a composite state. The function *loadState()* loads the state information from the knowledge base.

Class *transition* defines the profile variables and required function. Action selection implementation is also defined here. The attribute *selectType* indicates that the action

selection mechanism is a sequential execution, rule-based reasoning or Bayesian network inference. Similar to the state class, the number of input arcs and output arcs, as well as the references to those arcs are recorded in the corresponding variables. The attribute *status* indicates the status of the transition, that is, *enabled*, *idle*, or *fired*. The attribute *time stamp* records the timing information of the transition. The array *tasks* contain the task *Ids* for the transition. The task to be executed will be selected by the action selection method.

The function *loadTransition()* retrieves the transition information from the knowledge base. The function *guard()* checks the input states connected by the input arcs for the readiness. When the input states are all ready, the transition is enabled; and then the action selection engine will select the most suitable task to execute. Finally, the function *fire()* will be called to fire the transition. The transition may be fired under certain constraints. For example, if the parameter *time delay* is set, the transition can be fired only when the time delay is reached. The functions *ruleExe()* and *bayesExe()* are the rule-based reasoning and the Bayesian network inference respectively.

The functionalities of *arc* objects and *branch* objects are relatively simpler. An arc has one input object, *inputObj*, and one output object, *outputObj*. The attributes *inputObj* and *outputObj* contain the references of the objects and the types of the objects, state or transition. The function *transfer()* transfers the data from the *inputObj* to the *outputObj*. The function *filter()* is called in the function *transfer()* before the transferring actions. The *filter()* function can be overridden in a real agent application in order for necessary processing on the data. The function *checkStatus()* returns the status of the *inputObj* if the *inputObj* is a state. It is usually called by the guard function of a transition object.

A branch connects two states. The attributes *inputObj* and *outputObj* contain the references of the input state object and the output state object. The attribute *type* indicates the branch is a *left* branch or a *right* branch. If the attribute *type* of a branch object is "left", the function *copy()* will duplicate the data of *inputObj* into the *outputObj*. Otherwise, the function *copy()* will copy the data from the *outputObj* to the *inputObj*.

In summary, the goal process unit serves as an internal description of the agent's goal for solving a real world problem. In other worlds, it is also a model of the agent's external environment in the real world. The control unit makes decisions based on the goal model given by the goal process unit.

- **Perception Unit**

Perception unit defines the activities the agent senses the environment. It contains a list of states to indicate the status of the environment. If the environment changes, the perception unit will notify the control unit to take actions against the change. Figure 4.3(e) shows the definition of the perception unit. Once the agent starts work, the perception unit will keep running in a separate thread. The activities of perception unit are specified in the function *run()*, which is the execution body of the thread.

- **Control Unit**

Control unit makes decisions on agent goal selection and action selection. If the perception unit of an agent detects an environment change, the control unit calls function *react()* to react on the change. In the normal case, the control unit executes the functions based on the goal model in the goal process unit to pursue the goal of the agent. It monitors the progress of goal accomplishment and calls corresponding functions based on

the defined Goal Net. It drives to execute actions associated with the selected task in each transition. Figure 4.3(g) shows the definition of the control unit. The function *run()* of control unit defines the execution mechanism of the agent.

- **Action Unit**

Action unit contains all the action functions used by other units. It defines the mandatory functions that must be implemented in the real agent construction. For example, the function *react()* and *messageHandler()* that are used by the control unit are essential to the agent. Figure 4.3(b) shows the definition of the action unit. The other actions, such as the application specific functions, are also needed to be implemented in real agent construction. The action unit runs in a separate thread to execute selected actions.

- **Data Unit**

Data unit is an interface for data access. They are used to access data sources, such as databases or data files. The functions defined in the action unit use the functions defined in the data unit to manipulate data. They need to be implemented to control the real data operation during the agent construction. Figure 4.3(f) describes the interface definitions of the data unit.

- **Knowledge Unit**

Other than the goal model, an agent must have knowledge to solve real problems. For example, a business forecasting agent must have forecasting model to produce forecast results. Knowledge unit maintains such knowledge of an agent. It defines the functions of intelligent activities such as reasoning, learning, training and action planning for solving the real world problems. During the intelligent agent construction, the knowledge model,

the reasoning and learning algorithms of the knowledge model need to be implemented. Figure 4.3(c) describes the definition of the knowledge unit.

- **Communication Unit**

Communication unit defines the communication mechanism between agents. The agents communicate asynchronously through message queues. Figure 4.3(d) shows the definition of the communication unit. The function *pack()* and *unpack()* are used to pack messages and unpack messages respectively, based on the message transformation protocol. The function *listen()* is to check the message queue for any new message. Once it finds a new message, the control unit will be notified. The control unit will then take actions to handle the messages. The function *readMsg()* and *writeMsg()* are used to read messages from the message queue and to write messages to the message queue, respectively. Similarly, the communication unit keeps running in a separate thread when the agent is working. The function *run()* of the communication unit calls the function *listen()* to monitor the message queues.

- **Compute Unit**

The compute unit defines the goal selection function and the action selection functions. The goal selection algorithms, action selection algorithms and action selection mechanisms are implemented in the compute unit. The control unit will call the functions in this unit to select the next goal and actions according to the goal model given in the goal process unit. Figure 4.3(h) gives the definition of the compute unit.

4.3 Agent Work Flow and Characteristics

When an agent is running, there are four concurrent running threads, which are *perception* thread, *communication* thread, *control* thread and *execution* thread. The agent actions are executed through the execution thread.

After an agent starts to work, it is first assigned a goal to pursue. The agent will then restore the information for reaching the goal (Goal Net) from the knowledge base by the function *setGoal()* of the goal process unit. For the domain knowledge for solving the real problems, for example, a forecasting model for agent-based business forecasting, the agent will restore the knowledge from the knowledge base as well by binding the knowledge to the knowledge unit through the function *gainKnowledge()* of the knowledge unit. Then the agent starts the four concurrent threads: the execution thread, the perception thread, the control thread and the communication thread. The control unit (control thread) computes the next goal and the next actions based on the Goal Net through the compute unit to make the action plan and then puts the actions into the action list of the goal process unit. If the selected actions involve the knowledge in knowledge unit, the control unit will ask the knowledge unit to make the action plan by the function *actionPlan()* of the knowledge unit. At the same time, the action unit (execution thread) executes the actions in the action list of the goal process unit. If the perception unit (perception thread) detects any environment change, it notifies the control unit. The control unit then calls the function *react()* in the action unit to react to the change. Similarly, if the communication unit (communication thread) finds any new message received, it notifies control unit to handle the message using the function *msgHandler()* in the action unit. The agent is also able to send messages to the user by the function

sendMsg() defined in the communication unit. Figure 4.6 shows a sequence diagram of an agent.

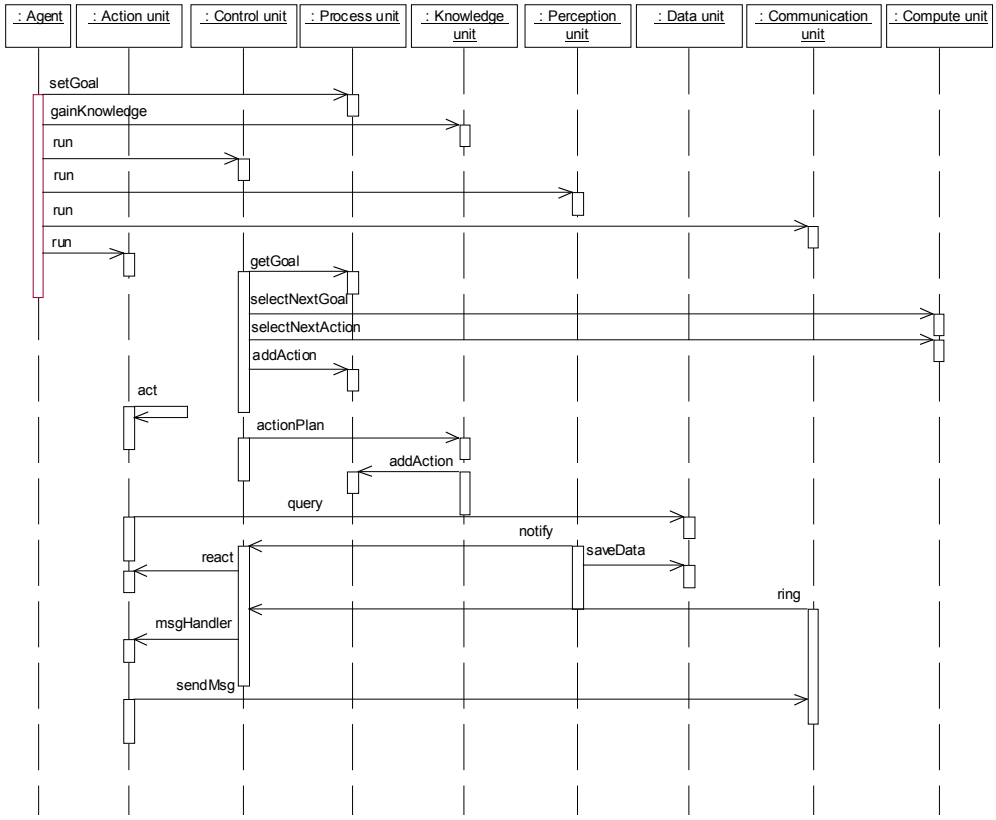

Figure 4.6 The sequence diagram of goal autonomous agent

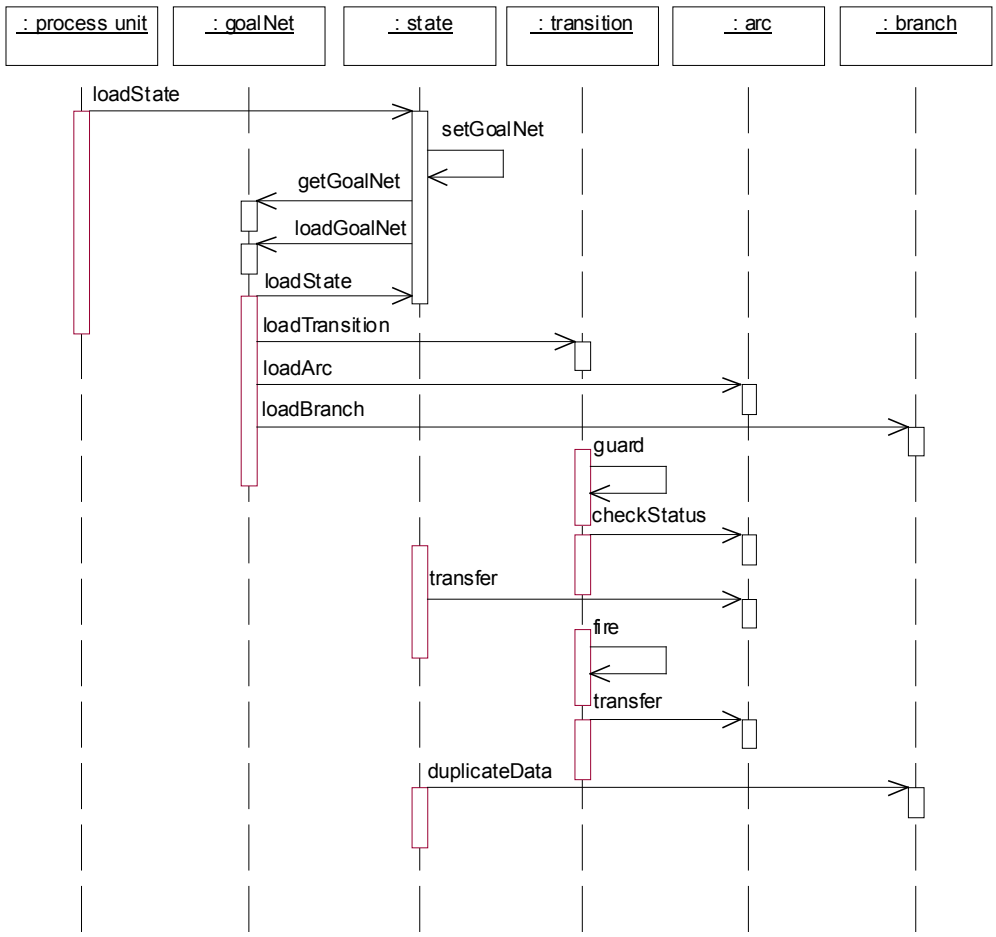

Figure 4.7 The sequence diagram of goal setting

Figure 4.7 shows the sequence diagram of a Goal Net. When an agent sets its goal by the function *setGoal()* of the goal process unit, the goal process unit will restore the Goal Net linked by the goal from the knowledge base through the function *loadState()* of the state object. This is a recursive process to load the goal model from the knowledge base. The first Goal Net is loaded first; then the states, transitions, arcs and branches of the Goal Net are loaded; the sub-nets of the newly loaded states are loaded respectively; the objects of

each of the sub-nets are followed. The loading will stop when all the states of sub-nets are atomic states.

The transitions will check for the status of the input states and fire accordingly based on the satisfaction of the certain conditions. The function *transfer()* of an arc is used to transfer data between a state and a transition whereas the function *duplicateData()* of a branch is used to duplicate data between the joint states.

A goal autonomous agent presents the following characteristics:

- Pro-active/goal-oriented

  The agents are goal-oriented, and pro-active to pursue their goals. The control unit acts as a co-ordinator and planner of the agent. It selects goals and actions, and makes the execution plans to pursue the goal. The action unit performs the actions in the action list proactively.

- Goal autonomy and behavior autonomy

  Once an agent starts running, the control unit, action unit, perception unit and communication unit work as separate threads concurrently. They make inference on both goal selection and action selection, to decide the next goal and suitable actions dynamically and autonomously based on the current environment and to react to the environment changes.

- Reactive

The perception unit of an agent is able to sense the environment. Once the environment changes, it notifies the control unit. The control unit then reacts to the changes through functions defined in the action unit.

- Intelligent

The agents have the ability of knowledge representation, reasoning and learning. On one hand, an agent can reason the goals and actions based on the goal model given in the goal process unit. It also can learn to improve the reasoning accuracy. On the other hand, an agent can solve the real world problems using the knowledge defined in the knowledge unit.

- Flexible

Goal autonomous agents are flexible and easily adapted to changes or un-expected situations.

4.4 A Goal-oriented Multi-agent System Model

Agents in a MAS environment are not isolated, they need to share common knowledge, goals and collaborate with each other in order to reach a common goal. Agents in a goal-oriented multi-agent system are modeled by the proposed Goal Net. In this section, we illustrate a multi-agent system can be further derived from a Goal Net.

[Definition 4.2] A goal-oriented multi-agent system is a tuple GAMAS = (A, C, O, K, E), where

- $A = \{agent_i \mid i = 1, 2, \ldots n\}$ is a set of goal autonomous agents;
- C is a set of communication channels;
- O is an ontology server;
- K is a knowledge base;
- E is the agent environment.

The main component of a goal-oriented multi-agent system is the goal autonomous agents proposed in this Chapter. Each of them works to achieve its own goal. In the mean time they collaborate with each other and present a system level behavior towards a common goal.

The knowledge base is shared by agents. Each agent has its own knowledge, which is stored in the knowledge base respectively. The ontology server provides descriptions of the concepts and relationships for agent communication. Ontologies maintained by the ontology server act as open-ended "dictionaries of words" describing common application areas and allowing consistency among the agents that have to communicate about those areas. The communication channels define the communication mechanisms between the agents.

The multi-agent system environment is where those agents live in. It defines the agents running environment including system environment (system architecture, operating system, network, database, etc), communication infrastructure (communication method, communication protocol, etc) and agent management environment (agent management server, environment variables, etc).

4.5 Multi-agent Development Environment (MADE)

We have developed a Multi-agent Development Environment (MADE) based on the proposed agent model and design for building multi-agent systems. The main components of MADE includes a Goal Net Designer, an Agent Creator, a Knowledge Loader and an Agent Space Manager. A three-simple-steps for developing an agent using MADE includes, defining the Goal-Net using Goal Net Designer, creating a dummy agent using the Agent Creator, and loading the specified Goal Net via the Knowledge Loader to the created agent.

- Goal Net Designer

  The main functions of the Goal Net Designer include a Goal Net Editor tool, and a Goal Net validation tool. Currently, the Goal Net Editor only has text interface. The graphic user interface (GUI) of Goal Net Editor has been being implemented to allow the user/developer to draw the Goal Net visually. After editing, the Goal Net will be saved in the knowledge base. A novel feature for the GUI-based Goal Net Editor is that it supports collaborative design. In another words, it allows multi-users to design a Goal Net collaboratively.

- Agent Creator

  An agent creator generates the codes needed for implementing an agent and creates the agents based on the proposed agent model and agent development framework described above. Besides the default code generation, it allows the user to modify the codes after the code is generated.

- Knowledge Loader

The knowledge loader loads the Goal Net stored in the knowledge base into the agent created by the agent creator. Moreover, it also provides the interface for loading the learning/reasoning mechanisms for a specific goal transition in the Goal Net. The significant advantage of Knowledge Loader is that it supports the design decomposition/re-combination, as well as the integration with external systems.

- Agent Space Manager

Agent space manager provide the services for managing the multi-agent system, such as agent registration, naming, directory etc. An interface for specifying agent communication protocols is also provided by Agent Space Manager.

MADE provides a complete environment for goal autonomous agent development. On the other hand, recently agent development tools for agent programmers have appeared. For example, Java Agent DEvelopment framework (JADE) [Bellifemine, 99] was developed to help implementation of multi-agent systems through a middle-ware that claims to comply with the Foundation for Intelligent Physical Agents (FIPA) specifications [FIPA, 2001]. It supports agent management, communication support and ontology server implementation. JADE starts to become a popular leading Agent Development Framework. However, JADE is only an agent development tool. It is not a modeling tool. Furthermore, it dose not support agent goal modeling.

To enable those who are familiar with JADE can also adopt goal-oriented modeling and design using MADE, we integrated MADE with JADE in this research. The Agent Creator and the agent development framework were integrated on top of the JADE.

In summary, MADE facilitates the multi-agent system design via a set of friendly and easy-to-use interfaces. In the future, the GUI-based Goal Net editor will allow developers to draw a multi-agent system visually. After the integration with JADE, we have a more powerful standard based JADE and a goal-oriented MADE. **A Reusable Architecture for Agent-oriented Systems**

The architecture mainly consists of three components: a goal autonomous agent, a knowledge base, and an agent management server, which is depicted in Figure 4.8. The agent management server is used to manage the agent, monitor the agent running environment, and interact with users.

- **Goal Autonomous Agent**

As given in Section 4.5, a goal autonomous agent can be developed by the Multi-agent Development Environment (MADE).

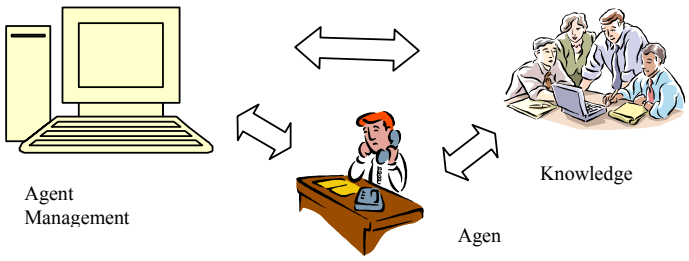

Figure 4.8 The agent system architecture

- **Knowledge Base**

The knowledge base is a knowledge repository of an agent. There are mainly three types of knowledge in the knowledge base, the agent goal model, domain knowledge, and the agent runtime data.

The goal models, as one type of agent knowledge are stored in the knowledge base. The goal model will be assigned to an agent when an agent starts to run. Different goal models lead the agent to different behaviors.

The diagram of the agent model in the knowledge base is presented in Figure 4.9.

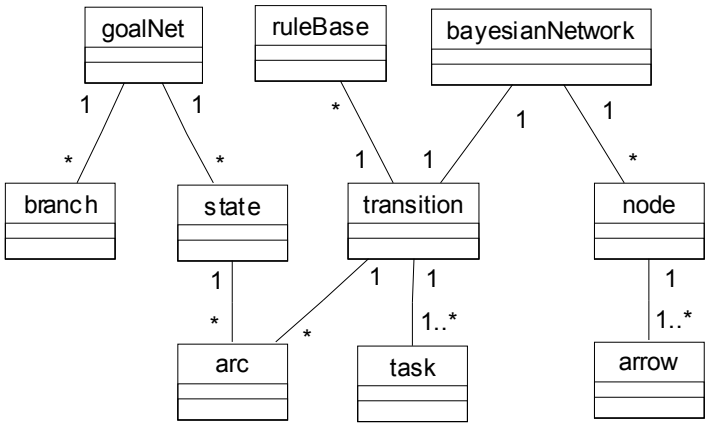

Figure 4.9 The goal model representation in knowledge base

As indicated in the diagram, Goal Nets, Bayesian networks and condition-action rules are associated through transitions. There are many goals in a knowledge base. The goals can be assigned to the same agent in different periods or assigned to different agents. A goal is mapped to a state, which participates in a Goal Net. A Goal Net is associated with states, transitions, arcs and branches. A state will be a state of a Goal Net but it may be the root state of another Goal Net at the same time.

Similarly, a Bayesian network is associated with many nodes and arrows. The causal variable names are associated with each Bayesian network. The prior probabilities of the causal relations are associated with corresponding arrows. Both goal models and Bayesian networks need to be loaded into the knowledge base before an agent can be started.

The domain knowledge is the knowledge for solving the real world problems. For example, a neural network based business forecasting model is the knowledge to do business forecasting.

The agent runtime data, such as goal accomplishment data and goal tracking data, is also stored in the knowledge base so that the agents' behaviors can be monitored.

- **Agent Management Server**

The agent management server provides many services for agent management. The services include:

1) Knowledge management service

The knowledge management service provides an interface for users to query, add, delete, and update the knowledge of agents in the knowledge base. The agent runtime data associated with an agent will be removed after the agent stops running. Users can set to save the runtime data permanently in a knowledge base for future reference as historical data.

2) User interface

The user interface is provided for users to consume services provided by agents and to communicate with the agents. For instance, with the user interface, users can create an agent, select service and start running an agent. Users can send special commands or messages to the agent through the interface while the agent may also send messages to the users for assisting decision-makings. The user interface can be web-based and can be accessed on the Internet.

3) Agent management service

The agent management service provides tools to manage agents. It has an agent factory to create agents. All the running agents have a registration entry in the naming service. The entry data will be removed after the agent is destroyed. The communication facilitator facilitates communications between agents. The service also provides tools to monitor the agent running and to generate reports from the agent runtime data or the historical data.

4) Environment monitoring service

The agent running environment is represented by environment variables. The environment monitoring service provides tools to monitor the variables and maintain their values. Agents sense the environment through the variables.

5) Server management service

The server management service provides user interface and tools to maintain the agent management server itself. For example, users can use this service to start or stop other

services, upgrade software components, manage the database or knowledge base, and so on.

In this section, we have presented a reusable architecture for designing and implementing goal autonomous agent systems. Each unit of the goal autonomous agent is independent of each other and can be developed as a reusable component. The whole agent design presents high reusability. The proposed framework enables faster development cycles of goal autonomous agents with better quality, decreased effort, and greater reusability.

4.7 A Reusable Multi-agent System Architecture

Based on the goal-oriented multi-agent system model, Figure 4.10 describes a re-usable multi-agent system architecture.

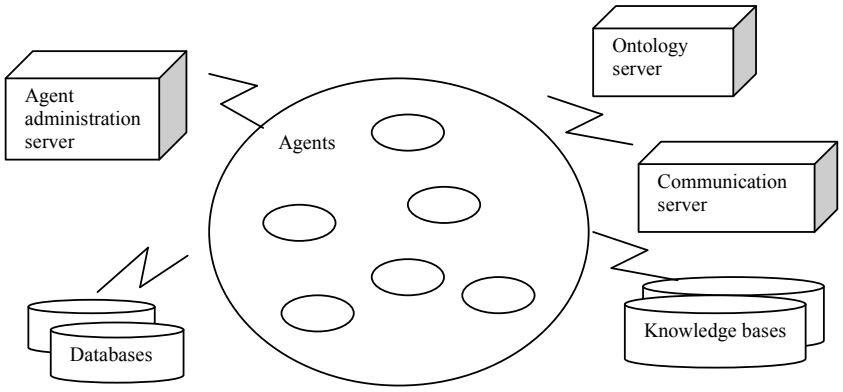

Figure 4.10 The multi-agent system architecture

As shown in this figure, a multi-agent system contains an agent management server, an ontology server, a communication server, a knowledge base, a database and a number of goal autonomous agents.

4.7.1 Agent Management Server

An agent management server provides services for managing agents. Such services include agent naming service, directory service, agent registration, information service, and job administration, etc.

When agent is running, the first thing it will do is to register itself with its personal information, such as goal model identifier, parent id, etc., to the agent administration server. The server will then create a profile for this agent and assign an agent name, a group identifier and a unique identifier to this agent. The agent name, identifier, and the reference of this agent are also registered into the naming service and directory service by the agent administration server automatically. When an agent stops running, its records in the agent administration server are deleted from the server. Obviously, the agents are organized in a group based on the underlying goal model hierarchical structure.

Attribute Name	Description
Name	Name of the agent, such as Agent1, Bob, Forecaster, etc.
Identifier	Unique ID of the agent
Location	Location of the agent, such as host name, IP address, URL, etc.
Reference	Reference of the agent, such as thread ID, process ID, descriptor, etc.
Start Time	Time of the agent starts running
End Time	Time of the agent stops running
Dynamic Info	Pointer to the information object in the information server
Status	Status of the agent, that is, busy or idle

Table 4.1 Agent Profile

Table 4.1 defines the agent profile. An *agent profile* is a description of agent in a multi-agent system. It contains information about an agent including name, identifier, reference, location, start time and end time of agent, etc. The agent profiles are maintained by the agent administration server. The naming service, directory service and the communication server use agent profiles to manage agents. Location and reference of an agent are also

generated by the agent administration server. But the representation formats are dependent on real implementation. There are other attributes recording dynamic information of an agent, such as, *status* indicating the running status of the agent, *busy* or *idle*; *dynamic info* is a pointer to the information object in the information center by which the jobs, goals, constraints currently the agent is taking are recorded.

The directory service provides agent information for other agents or other applications. Agents or other applications can search a particular agent through the directory service. Once the agent is found, the other agent or application can request service from or communicate with this agent by obtaining the agent reference from the naming service.

Information service manages all the information in the multi-agent system including agent information, system information and environment information.

- Agent information contains agent job types, goals, constraints and running state variables of current running agents.

- System information contains the number of agent groups, number of agents in each individual groups, names of communication channels, names of knowledge base and database, names of ontology server, agent administration server and name of communication server, user information, network protocols, message languages, etc.

- Environment information includes internal and external variables or factors specified by the designers. Agents will work closely with those variables and factors for goal selection and action selections.

Job administration manages jobs requested by users. When a user makes a request to the agent administration server, the agent administration server will search the knowledge base for the goal to handle the request and dispatch the goal to the master agent of a group who will then conduct its member agents to complete this job.

There are three types of jobs:

- One time job: The job is completed after the goal is reached.
- Repeatable job: The job will be started again after the goal is reached.
- Job with constraints: This type of jobs is a special case of the above two types of jobs. Together with indicating one time job or repeatable job, it also indicates some constraints for the job. For example, time constraint requests agents must complete the job within a certain period.

4.7.2 Communication Server

The communication server provides services for agent communication. The services include communication channels, user interaction, and communication administration.

Communication channels provide communication mechanism for agent communication. The channels can be TCP/IP network service, CORBA event channel, message queue, or their combinations. They are application dependent and can be decided in actual applications.

User interaction provides a mechanism for a user to communicate with agents. Agents communicate using specific language, Knowledge Query Manipulation Language

(KQML) [Finin, 93]. The user interaction service of communication server translates, converts and formats the messages between a user and agents. For example, a user wants to communicate with agents through a graphic user interface (GUI). The user interaction service provides such a GUI for users. When a message is given by a user, the communication server will translate this message to agent language through the ontology server, then convert it to KQML format and finally send it to the target agent. The process will be reversed when an agent sends a message to a user.

Communication administration provides general services for agent communication. For example, communication administration can have a connection pool to facilitate agent communication. It can provide language translation and conversion services to facilitate user interaction. It can also provide services to handle communication failures.

Recently agent development tools that provide agent management services, ontology service and communication service start to appear. For example, Java Agent DEvelopment framework (JADE) supports agent management, communication support and ontology server implementation. By choosing suitable development tools the multi-agent system addressed in this section can be easily designed.

4.8 Summary

This chapter proposed a goal-oriented agent model for designing goal autonomous agents with their goals modeled by Goal Net. To demonstrate the proposed model is not only promising but also practical, the agent design and the agent development framework were also presented. A multi-agent system model, multi-agent system architecture as well as a

multi-agent system development environment have also been presented for implementing agent-oriented systems.

In the next chapter, a goal-oriented methodology for engineering agent-oriented systems will be presented.

CHAPTER 5

GOAL-ORIENTED METHODOLOGY FOR DEVELOPING AGENT-ORIENTED SYSTEMS

Systems that consist of interactive agents represent a new software engineering paradigm. Despite the significant progress in the field of agent research and development, to date, there is still no widespread development and deployment of agent systems and multi-agent systems. This is because there is a lack of practical formal methodologies for developing agent-oriented systems. In Chapter 4, we proposed a multi-agent development environment (MADE) for facilitating the design and development of agent-oriented systems. In this chapter, we describe a practical Goal-Oriented (GO) methodology that assists the whole life cycle of multi-agent system development based on Goal Net.

5.1 Review of Existing Methods

A large majority of the current efforts on agent modeling and development still employ object-oriented methodologies, which model an agent as an extended object or so-called "smart object". Agents are goal-oriented. Agents-oriented modelling should starts from goals instead of objects. Recently, several agent-oriented software engineering (AOSE) methodologies have been proposed [Bauer, 2001; Wood, 2001; Caire, 2002; Bresciani, 2001]. The emerging of AOSE methodologies is an important step towards the widespread development of agent-oriented systems. However, there are some important issues that are not addressed by the existing AOSE methodologies:

- There are gaps between requirements to agent design, as well as between agent design and implementations. For instance, the well-known AOSE methodology Gaia is a high level agent-oriented methodology. Gaia neither deals with requirement capturing and modeling, nor deals with implementation issues. One important purpose of the agent design is to guide the implementation. There is a need to establish a precise connection between the agent design and implementation. Another important purpose of the design is to facilitate the realization of the requirements in an implemented system. There is also a need to build the connection between the requirement and the agent design. Hence, there is a need for an agent-oriented modeling tool that aid designers throughout the whole life cycle of agent-oriented system development.

- Most of the current AOSE methodologies focus on the static entity relationships such as goal and role, and lack of the modeling of dynamic relationships among agent goals, agent environment and agent behaviors.

- There is a gap between agent mental models and agent implementation. A lot of efforts have been put into research on agent mental models. Few efforts have been put in linking the agent mental models to the agent implementations.

- There is a need to support the flexible integrations with other systems including traditional systems. Software is becoming more and more integrated. The emergence of web services, grid computing, peer-to-peer network makes it possible to link worldwide applications and integrate them with each other. Every system might become a part of other systems. To our best knowledge, none of the current AOSE methodologies addressed the integration issues with traditional systems.

- There is a need to support the decomposition and combination throughout the lifecycle of agent-oriented system development, including requirements, design, and implementations. The need of worldwide service integration results in the fact that no one has control over the whole system. We are now moving to a world that software systems need to be easily decomposed, modified and re-composed/re-combined "on the fly" by different people who are not under the control of the designer of the systems.

- There is a gap between user's own expectation and the software. Hence, there is a need to take into account about user's own preference/expectations of the software in the design as well as to allow users to gain control of the software.

Although the autonomous and de-centralized nature of interactive agents makes multi-agent system a promising solution for the new generation of software, to date, there is still a lack of agent-oriented methodologies that can bridge the above gaps for assisting the

whole development life cycle towards the widespread development and deployment of multi-agent systems. In this chapter, we present a goal-oriented methodology for bridging the above gaps and for developing multi-agent systems.

5.2 Overview of the Proposed Methodology

The proposed Goal-oriented modeling methodology starts from the requirement capturing and analysis phase. The life cycle of MAS development using proposed methodology includes four phases: Goal-oriented Requirement Analysis phase, Agent Organization and System Architectural Design phase, Detailed Design phase and Implementation phase.

5.2.1 Goal-oriented Requirement Analysis

The requirement analysis phase is the initial phase in many software engineering methodologies. In the proposed GO methodology, Goal Net serves as a requirement capturing and analysis tool. The objective of requirement analysis is to produce the preliminary high level Goal Nets. Given a problem statement, to model the problem, and to design /implement a system for solving the problem, we analyze what are the goals, what are the possible behavior for achieving the goals, and what are the relationships among the goals. Hence, a complex problem is decomposed to many small goals that are associated with some possible behaviors for reaching the goals and the interactive relationships between goals. The whole process includes:

- Identifying the goals: This involves goal capturing, identification, the decomposition of a global goal into sub-goals as well as the combination of sub-goals into a high level goal.

- Modeling the environments: This includes the elicitation of the environmental variables for representing dynamic changing environment during the goal pursuing process.
- Analyzing the goal transitions: This involves identifying the relationships and interactions among goals and the possible tasks/actions that lead to the transitions between the goals.
- Forming the Goal Net: A preliminary Goal Net is formed through top-down decomposition, bottom-up combination or the mixed top-down and bottom-up i.e. sandwich approach.

The output of the goal-oriented requirement analysis is a preliminary Goal Net. The output of the requirement analysis phase will be the input of the agent organization and system architectural design phase as well as the detailed design phase.

5.2.2 Agent Organization and System Architecture

The agent organization and system architectural design phase includes:

- Agent identification: This involves defining agent identification rules, i.e. agent-goal binding rules. After the agent-goal binding has been performed on the Goal Net, an agent hierarchy will be formed. Each agent is bound with a decomposed Goal Net.
- Agent model: This defines an agent model for using Goal Net to direct its behaviors.

- Agent coordination: This specifies the agent coordination mechanisms. In the formed agent hierarchy, the higher level agent will act as a coordinator of the agents in the lower level.
- Communication protocols: This defines the communication protocols, communication languages, and ontology between identified agents.
- Multi-agent system architecture: This defines the components of the multi-agent system that include agents, agent server, ontology server, communication facilitator, database/knowledge base, etc.

The main output of this phase includes the agent hierarchy, agent communication protocols, and multi-agent system architecture.

5.2.3 Detailed Design

The detailed design includes:

- Goal refining. For each identified agent, refine the goals and sub-goals that are bound to the agent.
- Tasks and actions design. For each transition, identifying the task/action selection mechanism, environmental variable definitions, and constraints definition, etc.
- Reaction and message handler design. This defines how the agent will react to the unexpected events and received messages.
- Environment perception design. This defines how the environment will be monitored and how agents can be notified with the changes.
- Knowledge design. This involves specifying domain knowledge for the particular problem.

The output of the detailed design includes refined Goal Nets as well as detailed specifications of the agent models for each identified agent.

5.2.4 Implementation

In the implementation phase, the detailed development will follow the detailed design specifications to map the implementation platform to the detailed design notions.

The detailed implementation includes:

- Goal Net construction. This includes constructing the Goal Nets designed through previous phases and storing them into a database.
- Tasks/Actions development. This includes developing the tasks/actions identified in the detailed design specifications.
- Task selection mechanism development. This includes developing the task/action selection mechanisms for goal transitions.
- Knowledge implementation. This includes the implementation of the domain knowledge, such as a neural network based forecasting model for business forecasting.
- Agent implementation. This includes the implementation of agents based on the agent model.

Figure 5.1 depicts the life cycle of the multi-agent system development using the proposed GO methodology. We have developed a Multi-agent Development Environment (MADE) for supporting the proposed agent-oriented methodology and for building multi-agent systems (refer to Chapter 4). The main components of MADE includes a Goal Net Designer, an Agent Creator and a Knowledge Loader. A three-simple-step for developing an agent using MADE includes, defining the Goal-Net using Goal Net Designer, creating

a dummy agent using the Agent Creator, and loading the specified Goal Net via the Knowledge Loader.

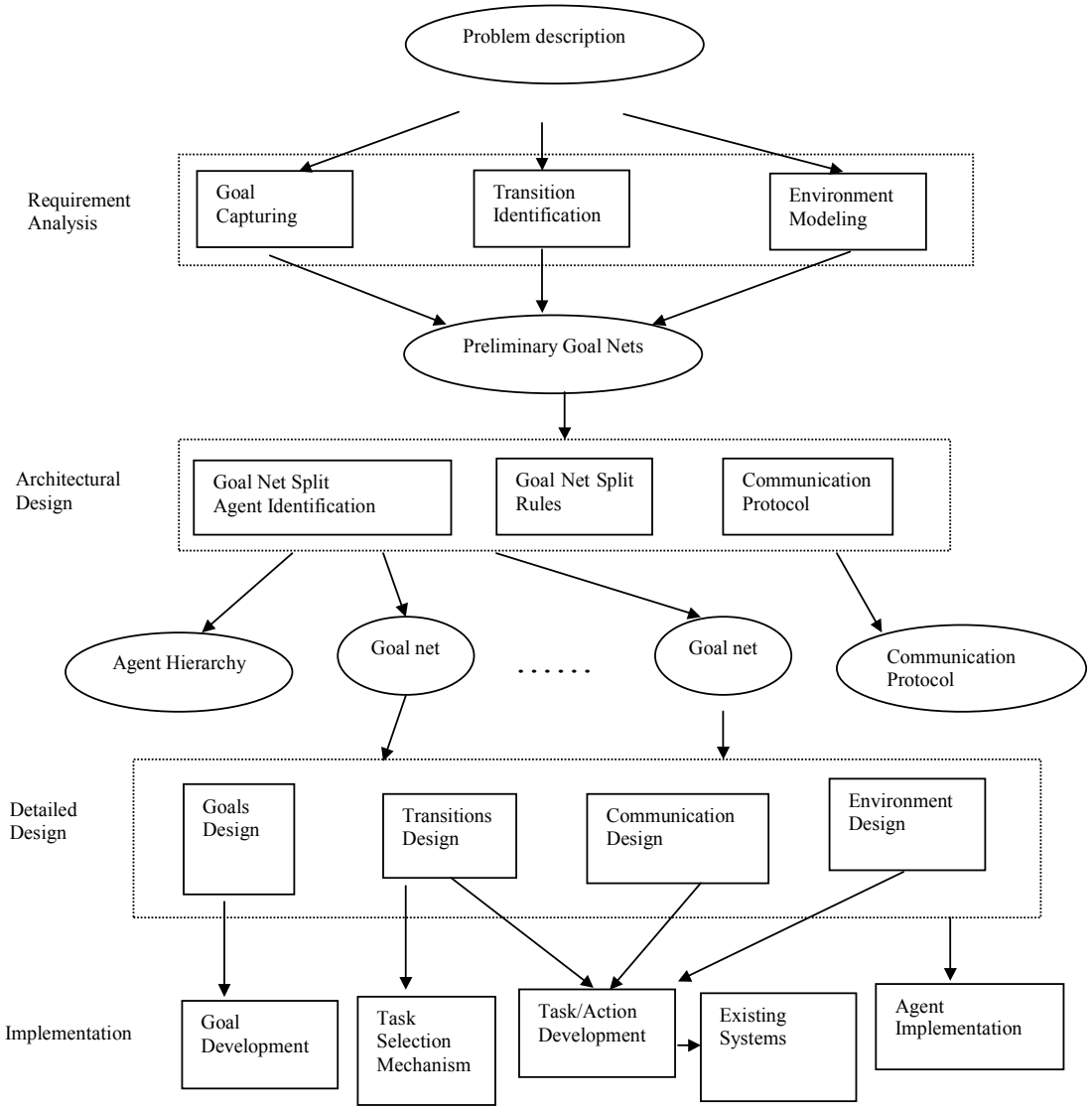

Figure 5.1 The MAS development life cycle

5.3 Example Problems

Before we go through the details of the methodology, in this sub section we introduce two example problems that will be considered as case studies to illustrate the proposed methodology and to show how the proposed methodology is used in the real-world applications.

5.3.1 E-Forecasting

Business forecasting is vital to the success of business. Moreover, nowadays, e-forecasting plays a more and more important role in people's every day life, such as stock forecasting, foreign currency forecasting, market forecasting etc. There has been an increased need for providing various e-forecasting services. And people want to access such e-forecasting services from anywhere, at anytime and in any form (via cell phone, PDA, Laptops etc.).

As an example problem, we consider a multi-agent system as a solution for providing the anywhere, anytime, and any form e-forecasting services. Basically, forecasting covers processes of data discovery/collection, data preparation, forecasting method training and generating forecasting results etc. [Armstrong, 2000; Allen, 2000].

(a) Data discovery and collection: Data is essential to the forecasting process. The data sources need to be discovered. According to the forecasting principles, the data needs to be collected as recently as possible and should be from diverse sources, in the application domain.

(b) Data preparation: The collected data is real data in the application domain. It cannot be used directly. It needs to be normalized. The relationships or associations between data should be also identified.

(c) Method training: The forecasting model needs to be trained by historical data before it can be used to do forecasting.

(d) Reasoning and Forecasting: The trained model and well-prepared data are used to reason and generate forecasting results.

To provide e-forecasting services, the system needs to satisfy all the above goals. Moreover, to allow users to access e-forecasting services from anywhere, at any time and in any form, the system needs at least to:

a) Always be ready to interact with users in order to collect users' requests and preferences;

b) Adapt to the environment changes, as users may request the services from different environments;

This naturally leads to a multi-agent system (MAS) in which each type of agent (personal agents, data discovery agents, data collection agents, data preparation agents, training agents, forecasting agents) caters for certain goals. The e-Forecasting MAS lives in an open distributed environment. New users from anywhere may request the services at any time. New data sources may be discovered, and new forecasting models may be added. The total number of user agents is unknown in the deployment time. The total number of data sources and the number of data collection agents are unknown and so are the total number of training agents and forecasting agents. Agents in the e-Forecasting system are de-centralized; each of them has its own environment, and pursues its own goal. Meanwhile, they work together towards a common goal for providing high efficient e-forecasting services.

5.3.2 E-Learning Grid Services

e-Learning provides learners distributed learning that includes digital content, and is experienced through Internet-enabled technology interfaces. In recent years, the knowledge-based economy makes e-learning services an important concern of most major organizations. Grid computing enables the sharing, selection, and aggregation of geographically distributed resources [Buyya, 2001a; Buyya, 2001b; Cao, 2002]. As e-Learning services need to address learning resources sharing and reuse, interoperability and various modes of interactions, grid technology is appropriate for providing learning services on the grid. This is commonly known as the e-Learning grid.

Given a requirement for providing e-learning services to each team that will be formed for newly tendered software projects in a software company, we consider a multi-agent system approach. For each role in a specific project, the multi-agent e-learning system should be able to suggest a list of team candidates based on the role requirements and the employees' profiles stored in the knowledge base of the organization. The best candidate will then be selected by the project manager and human resources department manager through a pre-assessment. The e-learning system should also be able to figure out the skills that the selected candidate lacks for fulfilling the roles of the project and to create a personalized learning path to train the selected candidate with the skills required by the project. Moreover, the e-learning system should be able to refine the learning path according to feedback from the learner in order to optimize the acquisition of needed competencies. Based on the learning path, the training courses and material should be delivered to the learner site on his/her demand. The system should also be able to assess the results of self-learning of the employee and readapt the learning path to cater for the

progress the employee has made. The self-learning cycle will be repeated until the assessment of the learning results meets the requirements of the role in the project.

Similar to the e-forecasting system, the multi-agent e-learning system is an open system. In the deployment time, the total number of the projects that will be formed is unknown. As a result, the number of the e-learning service agents for providing various services is also unknown.

5.4 Goal-oriented Requirement Analysis

Problem modeling and analysis, i.e. requirement analysis is the first phase of developing any software systems. Goal Net can serve as a problem modeling and analysis tool from the beginning of requirement analysis.

Goals are seen to have substantial promise in aiding the elicitation and elaboration of requirements. For example, the KAOS methodology [Dardenne, 93] uses goal as the central concept in requirements acquisition. Anton also uses goals as the main guiding concept in developing requirements specifications [Anton, 96; Anton, 97].

In the requirement analysis phase, the goal is to analyze the problem description and derive the preliminary Goal Net by identifying goals (*what*), analyzing how the goals can be pursued (*how*), and the environment that might affect how goals are pursued (*situation*). In a Goal Net, each composite state represents a goal pursuing process for solving a problem. The outputs of this phase are the preliminary Goal Nets.

5.4.1 GET Card

To simplify the requirement capturing and analysis process as well as to derive the preliminary Goal Nets, we introduce an easy-to-use index card, which is called Goal-Environment-Task (GET) card. GET card is an index card that is used to represent the Goals, the Task candidates for reaching the goal, and the Environment variables that represents the environment situation during the goal pursuit. The three elements capture the essential dimensions of goal-oriented modeling based on Goal Net. GET card is an easy practice approach to goal-oriented modeling. The cards can be created by the designers, and also the customers who are not familiar with Goal Net modeling.

Figure 5.2 and Figure 5.3 shows two simple examples of a GET card, which specify the goal, the task candidates for reaching the goal, and the environment variables during the goal pursuing process.

Goal: Arrive in lecture theater before 9 am	
Environment Variables	**Tasks**
Time Weather	Take a taxi Take a campus shuttle bus Ride a bicycle Walk

Figure 5.2 The first example of the GET card

Goal: Select a Java course from e-learning grid	
Environment Variables	**Tasks**
Price Course flexibility Reputation Number of students Accessibility Online delivery quality	Select course from NTU Select course from NUS Select course from ISS

Figure 5.3 The second example of the Get card

The above GET card shows that to achieve the goal "to be in Lecture Theater before 9am", a student may take one of the tasks in the task list. The task selection is based on the perception of the environment. For instance, if it's raining and there is not much time left, the student may *take a taxi*.

5.4.2 Goal Identification

Using Goal Net, the requirements analysis can be revealed in the process of goal capturing/identification and elaborating/structuring goals into a goal hierarchy. Following a knowledge-based top-down approach, the analysis starts from the overall goal (represented by a composite state in the root) to solve a complex problem in the real world. It is further decomposed to a set of sub-goals that encapsulate the goal pursuing processes for solving each decomposed problem. A sub-goal for solving a decomposed problem may need to be further decomposed to a number of sub-goals. The goal decomposition is continued until all sub-goals can be easily implemented and represented by GET cards. As a result, a hierarchical goal model described by a Goal Net is created.

Another approach is bottom up approach. Assuming that during the goal capturing, a set of GET cards have been collected, by asking *why*, the sub-goals can be aggregated to the higher level goals that forms a goal hierarchy.

In the e-forecasting example, the goal is to provide e-forecasting services. By asking how to achieve this top goal, it can be further decomposed to a set of goal pursuing processes, that is, sub goals. 1) Forecasting process. It generates a forecasting result based on a forecasting model. 2) Training process. The forecasting model needs to be trained before it can be used to produce forecasting results. 3) Data collection process. The data that is needed for training and forecasting needs to be discovered and collected. 4) Data preparation process. The collected data needs to be normalized or prepared from the raw data according to the requirement of the forecasting model. If all the above can be realized, we can collect requests from the users and provide them the e-forecasting services.

Accordingly, we now have identified four goals: collecting data, preparing data, training forecasting model, and generating forecast results. These four goals are sub-goals of the overall goal *providing e-forecasting services*.

By asking how to achieve individual goals, the four sub-goals can be further decomposed. For instance, a business forecasting model usually requires different types of data from various data sources. To achieve the goal *collecting data*, we need to collect data from the different data sources for different types of data. We then have sub goals of the goal *collecting data*. This process will be repeated until the decomposed goal can be represented by a GET card. Figure 5.4 gives the goal hierarchy.

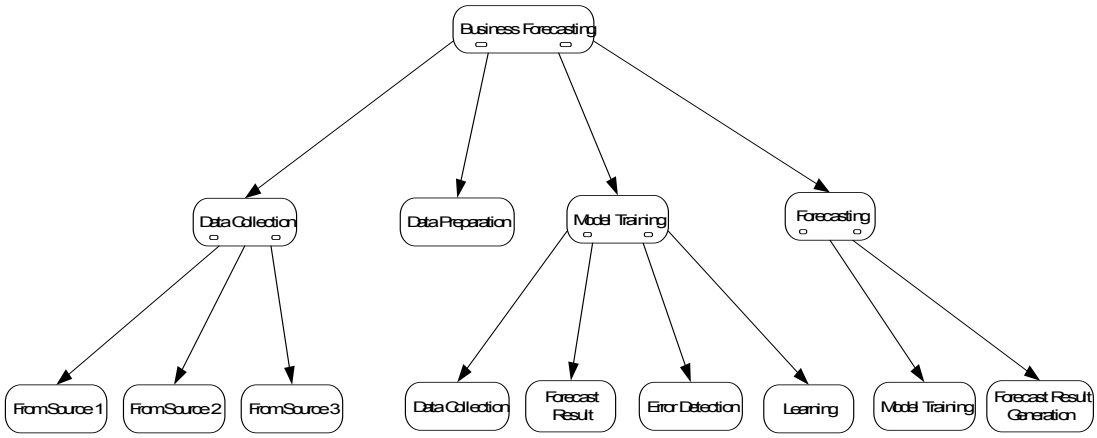

Figure 5.4 The goal hierarchy of e-forecasting

In the example e-learning grid services, the overall goal of the e-learning system is to provide e-learning services for a team that will be formed for a particular software project. By asking how to provide such services, the e-learning service can be decomposed to two sub goals (processes): learner assessment process and learning process. The learner assessment process can be further decomposed to role identification process, employee selection process and pre-assessment process. Learning process also can be further decomposed to learning path generation process, course delivery process and post-assessment process. The processes and their goals are listed in Table 5.1.

Process	Goal
E-learning service	Employee is well trained
Learner preparation	Learner is ready to learn
Learning	Learning is proceeded
Role identification	Role in a project is identified
Employee selection	Employee is selected for a specific role
Pre-assessment	Feedbacks from the employee are obtained

Learning path generation	Learning path is generated for the particular employee
Course delivery	Courses are delivered
Post-assessment	Results of self-learning are measured

Table 5.1 E-learning processes and their goals

The e-learning service process is the main process that provides flexible training services to individual employees. It provides a user interface to the organization. The services are completed through the layered sub-processes.

- The learner assessment process is to find employees who need to be trained and to make assessment for the employees selected.
- The learning process is to generate a personal learning path for individual employees and complete the training processes.
- The role identification process is to identify roles that are made up for a project team according to the project specification. The responsibilities of each role and the technical requirements for each role are defined.
- The employee selection process is to select employees for each role identified in the previous process. The selection is done by matching the technical requirement of each role to the user profile of each employee. Each role may have one or many candidates. The most suitable one should be selected based on all the factors that affect the availability of employees. In the case that no employee is selected for a particular role within the organization, request for recruitment should be made.
- The pre-assessment process is to obtain feedback from each of the selected employees. The assessment is made based on the user profile, technical

requirement for the role in the project and the training course materials. User profile may not contain complete information about an employee and may not be up-to-date. This process is necessary to gain an optimized learning path.

- The learning path generation process is to generate learning paths for all the employees selected for the project. It is done according to the user profile and feedback of each employee and the course materials. The generated learning path will then be stored in the user profile of individual employees.
- The course delivery process is to deliver the corresponding course materials to the employee site on his demand over the Internet or intranet. It is done according to the learning path generated for this employee. Employee can make his own arrangement for the training courses.
- The post-assessment process is to measure the achievement of the employee by the training courses. If the results have met the requirements of all the courses, the training for this employee is finished. His user profile will be updated accordingly. If the employee fails to pass any of the courses, a new learning path will be generated. He needs to continue the course work until he has passed all the courses in the learning path.

By structuring the identified goals, a goal hierarchy can be generated for modeling the whole learning processes using the proposed Goal Net, based on which a multi-agent system for assisting the whole learning process can be derived. Figure 5.5 shows the goal hierarchy.

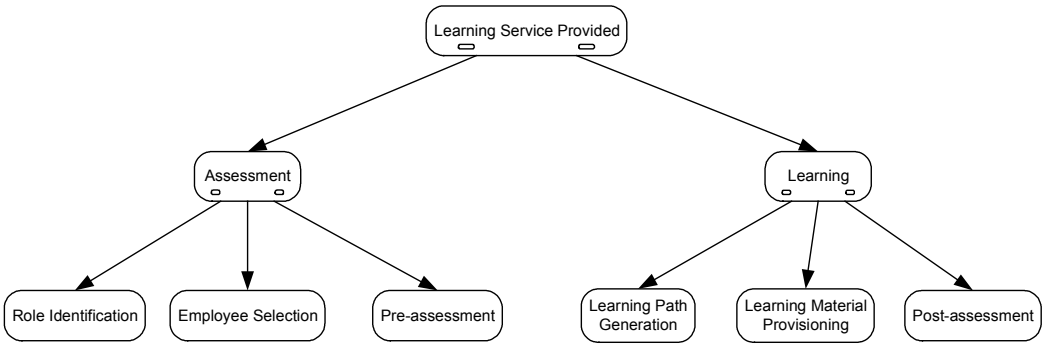

Figure 5.5 The goal hierarchy of E-learning

5.4.3 Environment Model

An agent works in its environment towards its goal. The agent working environment includes a physical environment, such as computers, networks, printers, etc.; a software environment, such as operating systems, database systems, network protocols, communication facilitators, agent management systems, etc.; and an application environment, such as application domains, problem scenarios, application data, and related external factors, etc. In the problem analysis stage we concentrate on the application environment because the physical environment and software environment can be decided during the detailed design phase and implementation phase. It is difficult to provide a general model for the environment because different applications have different environments and different designs use different methods. However, it is common to use variables to represent environment sources and the values of the variables to indicate the changes of the environment.

The environment variables or factors are important for agent behavior. Agents decide their actions based on the current values of the environmental factors. Goal Net is to model the dynamic behaviors of an agent towards its goal. With a Goal Net, the agent can

select its next goal and the task to reach the next goal dynamically according to the current environmental situation. So environment modeling/monitoring is an important stage for Goal Net construction. The environment modeling includes three steps:

- Identify what environment data are available or easily sensed

There could be many source of environment data in an application domain. Some are sensable, some are not. We need to find thoes sensable data and make use of them to detect environment changes so that the best solution can be selected based on the current situation. In the example of e-forecasting the environment data could be time, data status in data sources, forecasting model status, etc. In the example of e-learning, the environment data could be time, status of learning objects, etc.

- Analyze environmental data and decide how to sense the data and percieve the changes

After the environment variables are identified, the data need to be analyzed. This will include the data type, data values, and the representation of the data. We also need to decide how to sense the data and perceive the data changes and how an agent can detect the changes.

- Associate environment variables with goals and transitions in a Goal Net

The last step is to associate the environment variables to the elements of the Goal Net. When we define the Goal Net, we have defined the environment variables for goal selection and task selection based on the current values of such environment variables. We need to match the environment variables to the variables we just identified in the previous steps. If there are un-matched variables we need to adjust the Goal Net so

that all the environment variables defined in the Goal Net are monitorable in the real application system environment.

5.4.4 Transition Identification

One of the differences between Goal Net and other goal modeling methods is that after a complex goal is decomposed to sub-goals, the Goal Net can not only represent the hierarchical relationships but also interactive relationships among the sub-goals. Transitions of Goal Net are used to describe the interaction relationships between goals. In another words, transition of Goal Net models show how agents can transit to their next goal from their current goal in which goal pursuit paths can be decided dynamically at runtime.

In the last stage, we have identified the goals and sub-goals for solving a problem. In this stage we shall define how to transit between two goals. So we need to go through the following steps:

- Identify transitions

  For each goal hierarchy, we start from the root goal. First we add a state to indicate the initial state of the root goal and a state to indicate the final state when the root goal is reached. Then we start from the initial state to find the next goal to be pursued. If there is only one next goal to pursue, add a transition between the initial state and the selected goal. If there is more than one goal to pursue, there are two situations:

  - Only one goal is selected based on runtime environment. In this case, add a transition between the initial state and each of the selected goals respectively. The type is choice. They are graphically represented by the

triangular arrows. The rule to select the next goal needs to be defined then. There are two kinds of rules:

- Conditional rule: in this case, the next goal will be selected according to runtime conditions.
- Computational rule: in this case, the next goal will be selected according to the goal selection algorithms defined in the Goal Net model.

- All the selected goals are pursued in any sequence or in parallel. In this case, add a transition between the initial state and each of the selected goals respectively. The type is concurrent. They are graphically represented by the diamond arrows.

For each selected goal, we select this as the initial goal to repeat the above procedure to identify the transitions to its next possible goals. The procedure will be continued until the end state is reached.

- Define transitions

A transition indicates that an agent transits from one goal to another by finishing a task defined in the transition. Each transition is associated with a task list which contains all the possible tasks for reaching the goal. An environment variable list is also associated with a transition that represents the situations of the agent environment. An agent may select different tasks for reaching the same goal based on different environment situations.

In this stage, we need to define the tasks for each transition. The task to transit from one goal to another may not be unique. A transition may have a list of tasks, each of which might be fulfilled individually to make the transition to the next goal. Action

selection mechanisms defined for the transition will select suitable tasks to execute according to the current environment situations. So we need to find out all the possible tasks for each transition and then decide the task selection mechanism.

An agent will select different tasks in different environment situations. Modeling and monitoring environment is important for successful goal pursuing. For each sub-goal in a Goal Net, a transition can be identified through the GET card. For a specific transtion, a suitable action selection mechanism needs to be defined in a real application.

Once the goals, transitions and the environment variables are identified, a preliminary Goal Net can be formed. In the example of e-forecasting, based on the goal hierarchy depicted in Figure 5.4, a preliminary Goal Net can be formed as Figure 5.6. As shown in the figure, the four sub-goals identified need to be pursue in order. The next goal of data preparation could be model training or forecasting depending on whether the forecasting model is well trained. The sub-goals of data collection can be pursued in parallel. They will be synchronized before the final goal *data collection* is reached.

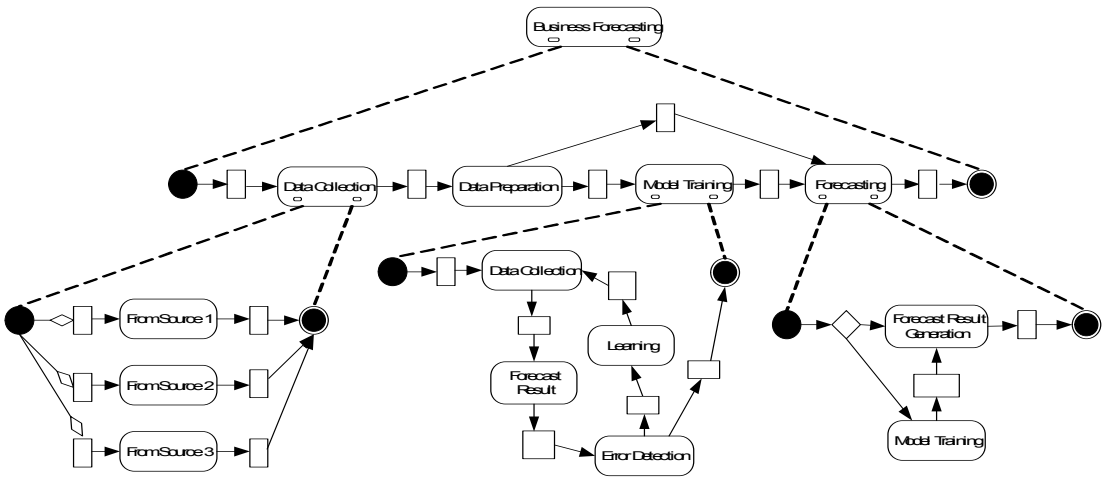

Figure 5.6 The Goal Net of e-forecasting

Similarly in the example of e-learning, based on the goal hierarchy depicted in Figure 5.5, a preliminary e-learning Goal Net can be formed as Figure 5.7. As shown in the figure, the sub-goals must be pursued in order. However, after the goal post-assessment is reached, it will be based on the result of assessment and the course completeness to decide its next goal, completing the learning process or continuing the learning process.

Figure 5.6 and Figure 5.7 also show that the interaction between goals only happens in the same decomposition of the super goal. This increases the modularity and encapsulation of Goal Net.

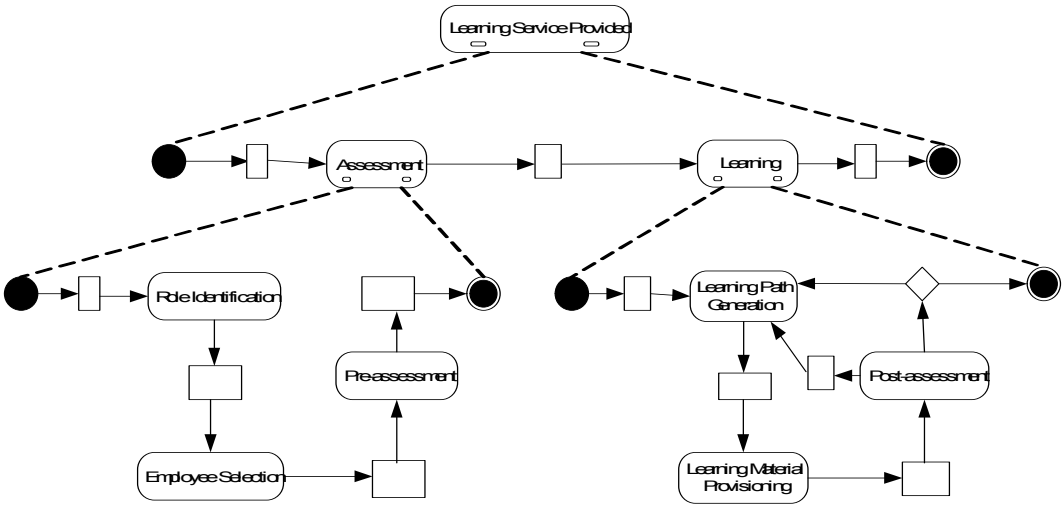

Figure 5.7 The Goal Net of e-learning

To summarize the problem analysis phase, the preliminary Goal Nets that model the problem are constructed. A Goal Net presents a hierarchical structure with goals, sub-goals and transitions between goals. In the next phase, a multi-agent system will be constructed.

5.5 Agent Organization and Systme Architectural Design

5.5.1 Agent Identification Rules and MAS Derivation

Goal Net method decomposes a real world problem into a Goal Net from which a set of agents can be identified to form a multi-agent system. A Goal Net can be split to a number of sub-nets based on the algorithm presented in the Section 3.4.1. The rules or strategies to identify agents will be decided during design time. There are many factors affecting the agent identification rules, including:

Modularity – each split Goal Net solves a sub-problem,

Reusability – each split Goal Net can be used to take part in a new composition for solving a particular problem,

Location – each split Goal Net is for a distributed located agent,

Role-based organization – each split Goal Net is corresponding to a role in an organization, and

Load balancing – Goal Net is split into sub Goal Nets to balance the load of agents.

A Goal Net is organized in a hierarchical structure. Each sub-net identified according to the Agent Identification Rules corresponds to an agent. Therefore, a complex Goal Net can derive a multi-agent model. The agents are organized in the hierarchical structure derived from the original Goal Net. The higher level agent becomes a coordinator of its lower level of agents in the agent hierarchy.

In the example of e-forecasting, the original Goal Net, depicted in Figure 5.6, can be split to three sub-nets (Data Collection, Model Training, and Forecasting) with consideration of four factors: modularity, reusability, role and load balancing. The three sub-nets correspond to three agents. Figure 5.8 shows the split Goal Nets and generated agent hierarchy.

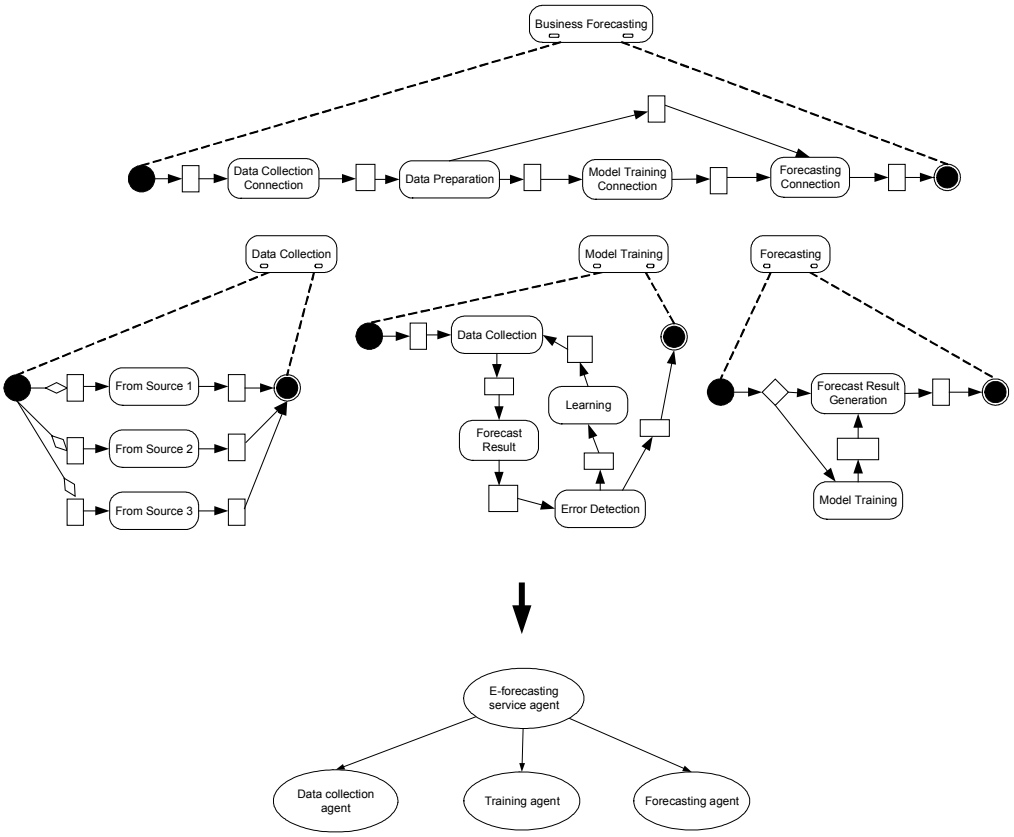

Figure 5.8 The multi-agent model derivation

The e-forecasting service agent (formed by the original Goal Net without split Goal Nets) becomes the coordinator of data collection agent (formed by the Goal Net data collection) and the training agent (formed by the Goal Net model training) and the forecasting agent (formed by the Goal Net forecasting). The connection states in the original Goal Net

become the synchronization points between e-forecasting service agent and other agents respectively.

5.5.2 Agent Communication Protocols

Once a multi-agent model is derived, communication protocols are required for the agent interaction. The protocols defined here are conceptual because the real protocols rely on the technology used and the agent running platform which will be decided in the detailed design phase and implementation phase. The conceptual protocol defines the messages required for agent communication. The format of a message is application dependent. For example, a message consists of following attributes:

- Message ID: a unique identifier for the message
- Message Source: the agent identifier who sent this message
- Message Target: the agent identifier whom the message was sent to
- Message Type: the type of the message
- Content Type: the type of the content contained in the message
- Content: the data

The category of message types and content types are application dependent. The receiver will interpret the data based on the message type and the content type. In the e-forecasting, we defined the protocols as shown in Table 5.2 for the communication between forecasting agent and other agents.

Message ID	Source Agent	Target Agent	Message Type	Content Type	Content
	Forecasting Agent	Data Collection Agent	Request	Info	
	Data Collection Agent	Forecasting Agent	Reply	Info	
	Forecasting Agent	Training Agent	Request	Info	
	Training Agent	Forecasting Agent	Reply	Info	
	Training Agent	Forecasting Agent	Request	Data	
	Forecasting Agent	Training Agent	Reply	Data	
	Training Agent	Forecasting Agent	Request	Data	
	Forecasting Agent	Data Collection Agent	Reply	Data	
	…	…			

Table 5.2 The sample communication protocols

The message ID will be generated at runtime. The message type *request* indicates this is a request message. The receiver needs to send a reply message to indicate the its response: accepted or rejected. The message type *reply* indicates the message is a reply to previous request, so the previous message ID is included. The content type *info* means the message contains information for cooperation whereas content type *data* means the message contains data for processing that is required by the receiver agent. Usually the contents of type *data* are exchanged after the two involved agents have made agreement by having exchanged the contents of type *info*. Of course, a more detailed and complete protocol should be defined in a real application development.

5.6 Detailed Design

In this phase, all the Goal Net entities, including goals, transitions, tasks, and actions will be designed; communication protocols and communication language between agents will be defined; and environment operation and management will be designed as well. The output of this phase should be able to guide the actual implementation of the agent system that will happen in the next phase.

5.6.1 Goal Design

The detailed goal design includes defining the attributes of goals and functions of goal, such as worth function, initialization function and goal selection function.

The worth function computes the achievement of the goal pursuit. It is important to set the expected goal achievement. Sometimes goal is difficult to achieve but partial goal is usually easier. Therefore, a threshold needs to be set for decision-making.

The goal selection function decides the selection of the next goal based on goal selection algorithm of Goal Net. The factors affecting the goal selection include:

- The current goal achievement – Examples include: The current goal is achieved or is not achieved; the current goal is partially achieved but it is greater than the threshold or the current goal is partially achieved and it is lower than the threshold.
- The environment situation – The next goal is selected based on environment changes.
- The time, cost, constraint etc.

So, the goal selection functions together with the factor variables need to be defined.

The initialization function needs to be defined to set the initial values of all the attributes and variables so that the functions defined on them can be run properly.

5.6.2 Transition Design

Transition design includes task selection mechanism design and task design. In this step, the task selection mechanism is designed in details. The environment variables, time, cost and other factors that are used by the functions need to be defined in the design. There may be a list of tasks defined in a transition. The task selection mechanism will be used to select suitable tasks to pursue the goal. All the tasks need to be identified and designed. However, a task can be associated with one or more transitions. So tasks are transition independent, that is, can be reused.

A Goal Net is the knowledge about solving a problem while a task is the execution of a solution. The agent may have the knowledge to solve a problem but it may not execute the solution if the task is not implemented. Once the task is implemented and becomes available to the agent, the agent can then execute the solution to solve the problem. So, a task can be implemented after the agent is running, in which case, the agent will not select this task during its goal pursuit. This requires that each transition must have at least one task implemented in order to run an agent.

5.6.3 Protocol Design

In this step, the protocols for agent communication are enumerated and designed. The language for agent communication is also selected, for example, ACL (Agent Communication Language) or KQML (Knowledge Query Manipulation Language). Usually the communication language selection is based on the agent development technology and platform. The designed protocols should comply with the requirements of the selected language and agent development platform. For example, in the e-forecasting

example and the e-learning example, if we adopted JADE (Java Agent Development Environment) as our agent platform, which uses ACL as agent communication language, all the protocols defined in the two examples should be designed in JADE environment.

5.6.4 Environment Design

We have identified and defined environment variables for goals and transitions of the Goal Net. In this step, we need to design

- the environment variables – what are their data types and what are their possible values;
- the environment interface – how an agent gets and sets the value of environment variables; and
- the environment management – how the environment variables are changed to reflect their current values and how the environment is monitored.

5.7 Implementation

In this phase we have all the detailed specifications of the system-to-be. The system is ready for implementation and deployment. The procedure for the system implementation and development is listed as follows:

- Build up the knowledge base. This step is to implement the Goal Nets designed and store them into the knowledge base. When an agent runs, the particular Goal Net will be loaded into the agent running environment, the tasks will then be executed by the agent.

- Develop intelligent actions. The agent development framework provides an interface for intelligent actions. The intelligent actions developed separately can be plugged into the framework through the interface. In the business forecasting example, the forecasting model is based on fuzzy neural network. The fuzzy neural network was designed and developed separately. Through the framework interface, the intelligent actions based on the fuzzy neural network implementation are bound into the agent at runtime. Similarly, the existing systems can also be wrapped in this way for integration.

- Develop goal autonomous agents and environment management facilities. In this step, the designed environment facilities for environment variable management are developed. With the agent development framework, the goal autonomous agent can be developed by extending the basic agent abstraction defined in the framework.

- Develop designed tasks and actions. In this step, the designed tasks and actions are implemented. The agent will check the availability of tasks at runtime. If the task selected by the task selection mechanism is not available, the agent will try the next selected task. If there is no task available at this time, it will send message to the user for advices.

We have developed a multi-agent development environment (MADE) for supporting the proposed agent-oriented methodology and for building multi-agent systems. The main components of MADE includes a Goal Net Designer, an Agent Creator and a Knowledge Loader. Three simple steps for developing an agent includes, defining the Goal-Net using

Goal Net Designer, creating a dummy agent using the Agent Creator, and loading the specified Goal Net via the Knowledge Loader.

The Goal-Net Designer is a tool to provide a GUI-based environment for designing Goal Nets. The Goal Nets designed by Goal-Net Designer will be stored into the knowledge base. When an agent runs, the particular Goal Net will be loaded to the agent by the Knowledge Loader so that the agent can run autonomously in its environment.

The Agent Creator is built on top of the agent development framework described in Chapter 4. It is used to facilitate the agent development and deployment. The designed tasks, actions and protocols, etc. can be implemented in this environment. The Agent Creator works with the Knowledge Loader. During deployment, the Agent Creator will create an agent and load a Goal Net through the Knowledge Loader.

5.8 Evaluation of the methodology

To compare the proposed GO methodology with existing AOSE methodologies, we adopt Dr. Kavi's comparison framework [Kavi, 2004] with the properties in the following four Categories:

1) Concepts and Properties

2) Notations and Modeling Technique

3) Software Engineering Process

4) Pragmatics

The objective of this comparison is to evaluate criteria concerning building blocks of both from software engineering process and from agent-oriented characteristics. In this AOSE comparison framework, four leading AOSE methodologies including GAIA, MaSE, AgentUML, and Tropos, have been compared against the properties classified by the above four categories. Following we include the proposed GO methodology into the comparison framework and make comparison to the related methodologies.

- Concepts and Properties

Features	Gaia	MaSE	Agent UML	Tropos	GO
Behavior autonomy	Yes	Yes	Yes	Yes	Yes
Goal autonomy	No	No	No	No	Yes
Mental Mechanism	No	Yes	Yes	Yes	Yes
Adaptation	Yes	No	Yes	Yes	Yes
Concurrency	Yes	Yes	Yes	Yes	Yes
Collaboration	Yes	Yes	Yes	Yes	Yes
Agent-Oriented	Yes	Yes	Yes	Yes	Yes

Table 5.3 The comparison of the methodologies for the first category

Features	GO Methodology	Justification
Behavior autonomy	Yes	The key notion of GO methodology is Goal Net. As an agent goal model, Goal Net supports flexible action selection mechanism that enables agent to act towards its goal autonomously. Therefore the agent is able to present behavior autonomy.
Goal autonomy	Yes	Goal Net enables the agents to present both behavior autonomy and goal autonomy. Goal Net facilitates the dynamic goal selection through various goal selection mechanisms with anytime goal selection feature.
Mental Mechanism	Yes	Using Go methodology, each agent carries a Goal Net as its brain. Goal Net provides the mental mechanisms for agents.
Adaptation	Yes	Goal Net supports flexible learning /reasoning mechanisms for action/goal selection that enables agents to be able to adapt to the environment.

Concurrency	Yes	Goal Net models various temporal relationships including concurrency. It allows an agent to perform multi tasks at the same time.
Collaboration	Yes	Goal Net serves as a multi-agent identification/organization/coordination model. Agents are able to cooperate with other agents to achieve a common goal.
Agent-Oriented	Yes	The design of the GO methodology originated from the consideration of agent-oriented ways to model, design, and implement software as a collection of agents.

Table 5.4 The justification of the GO methodology for the first category

Compared with the other related methodologies, GO methodology is the only one that supports the goal autonomy.

- Notations and Modeling Technique

Features	Gaia	MaSE	Agent UML	Tropos	GO
Decomposition	Yes	Yes	Yes	Yes	Yes
Design Re-combination	Low	No	No	Low	High
Goal-Oriented	No	No	No	Yes	Yes
Dynamic relationships between agent's goal, agent environment and agent behavior	No	No	No	No	Yes
Incorporate flexible action selection mechanisms	No	No	No	No	Yes

Table 5.5 The comparison of the methodologies for the second category

Features	GO Methodology	Justification
Decomposition	Yes	The key notion of GO methodology is Goal Net. Using GO methodology, a complex Goal Net can be decomposed into a number of sub-Goal Nets, each of which will be carried by an agent. Within a Goal Net, a goal can be further decomposed to sub goals.
Design Re-combination	High	A Goal Net can be further decomposed into sub Goal Nets. Each sub-Goal Net can be re-combined with

		other Goal Nets to form new Goal Nets.
Goal-Oriented	Yes	The design of the GO methodology originated from the consideration of goal-oriented ways to model, design, and implement a software as a collection of agents, i.e. multi-agent systems (MAS). Goal Net is a goal oriented model. Goal Net serves as a goal-oriented requirement and modeling tool, and a multi-agent identification, organization and coordination model. As a novel goal oriented model, Goal Net assists in all the phases of the life cycle for development of agent oriented applications.
Dynamic relationships between agent's goal, agent environment and agent behavior	Yes	Goal Net models an agent in a dynamic goal pursuing process. It models dynamic relationships between agent's goal, agent environment and agent behavior.
Incorporate flexible action selection mechanisms	Yes	Goal Net incorporates flexible action selection mechanisms including rule based action selection, probabilistic action selection and direct action selection.

Table 5.6 The justification of the GO methodology for the second category

Compared with the other related methodologies, GO methodology is the only one that models dynamic goal relationships and incorporates flexible action selection mechanisms.

- Software Engineering Process

Features	Gaia	MaSE	Agent UML	Tropos	GO
Life-cycle Coverage	Partial	Partial	Partial	Full	Full
Implementation Toolkits	No	No	Yes	Yes	Yes
Connection between Mental Model and Implementation	No	No	No	No	Yes
Deployment	Yes	Yes	No	No	Yes
Integration with existing system	No	No	No	No	Yes
Collaborative Design	No	No	No	No	Yes

Table 5.7 The comparison of the methodologies for the third category

Features	GO	Justification
Life-cycle Coverage	Full	GO methodology covers the whole software development life cycle of engineering agent-oriented systems from goal-oriented requirement and modeling, through multi-agent architecting, organization to agent design and implementations. As a novel goal oriented software engineering methodology, GO methodology assists in all the phases of the life cycle for development of agent oriented applications. The GO methodology provides guidelines from analysis, design to implementation.
Implementation Toolkits	Yes	GO methodology is supported by the multi-agent development environment (MADE) developed in this research.
Connection between Mental Model and Implementation	Yes	Through Goal Net and MADE, GO methodology builds up seamless connection between agent metal model and implementation.
Deployment	Yes	The GO methodology provides a way for practical deployment of agents through goal autonomous agent design model and multi-agent development environment MADE.
Integration with existing system	Yes	The GO methodology separates the design of agent brain and agent body. The multi-agent development environment, MADE has been integrated with JADE, a leading agent development tool.
Collaborative Design	Yes	The GO methodology allows multiple developers to do collaborative design using multi-agent development environment, MADE.

Table 5.8 The justification of the GO methodology for the third category

Compared with the other related methodologies, GO methodology is the only one that supports seamless connections between agent mental models and implementations.

- Pragmatics

Features	Gaia	MaSE	Agent UML	Tropos	GO
Required expertise	High	High	Medium	High	Low
User Preference in Design	No	No	No	No	Yes
Tools available	No	Yes	Yes	No	Yes
Model suitability	No	No	BDI	BDI	No
Automated AOSE	No	No	Partial	Partial	Partial

Table 5.9 The comparison of the methodologies for the fourth category

Features	GO	Justification
Required expertise	Low	The GO methodology is supported by an easy to use multi-agent development environment, (MADE). To provide initial evidence that a developer with no prior experience in agent development can apply the methodology and use MADE for agent development, a small group of Final Year Project computer engineering students have completed intelligent agent development related to their projects using the proposed GO methodology, Goal Net and multi-agent development environment, MADE.
User Preference in Design	Yes	The GO methodology is supported by an easy to use multi-agent development environment, (MADE). MADE has three main components, Goal Net Designer, Agent Creator, and Goal Net Loader. In Goal Net Designer, the developer designs the Agent Brain according to user's own preferences.
Tools available	Yes	The GO methodology is supported by an easy to use multi-agent development environment, (MADE).
Model suitability	No	The methodology is not based on a specific architecture.
Automated AOSE	Partial	The GO methodology is supported by an easy to use multi-agent development environment, (MADE). MADE has three main components, Goal Net Designer, Agent Creator, and Goal Net Loader. In Goal Net Designer, the developer design the Agent Brain. Using Agent Creator, the Agent Body can be automatically created. The Knowledge Loader can load Agent Brain to Agent Body.

Table 5.10 The justification of the GO methodology for the fourth category

Compared with the other related methodologies, GO methodology is the only one that requires low agent expertise.

The GO methodology consists of a goal oriented model, Goal Net, a goal autonomous agent model, and goal autonomous agent design model which are based on the Goal Net, and is supported by a multi-agent development environment, MADE. In proposing this novel goal oriented software engineering methodology GO, we carefully studied and evaluated the related agent oriented software engineering (AOSE) methodologies. Motivated by the common drawbacks of existing AOSE methodologies, we specifically designed and implemented the features summarized in the above tables in Goal Net, goal autonomous agent model, goal autonomous agent design model, as well as in MADE to address both software engineering and agent oriented properties.

In summary, GO methodology covers the whole life cycle of MAS development with strong notion of goal orientation, and bridges the gaps from agent mental models to agent implementations. GO methodology is a practical methodology supported by an easy to use MADE for development of MAS systems. We have applied the GO methodology and MADE in a few applications including e-learning, e-forecasting and agent mediated grid services. The agent development and experiments have demonstrated the above features designed in GO methodology.

A number of research papers derived from this research that address the most important features of GO methodology have been published in international journals and leading conferences. The key notion of the GO methodology, Goal Net, was presented in IAT 2004 (International Conference on Agent Technology 2004, acceptance rate 22%). The paper presents the goal oriented modeling and goal autonomous agent based on Goal Net,

and has gained well recognition in the agent community. The detailed goal autonomous agent design has been presented in the paper accepted by the 28th annual international conference on computer software and applications (COMPSAC 2004).

A number of well published AOSE methodologies (e.g. [Bussmann, 2004], etc.) use the computer science students to conduct third party review of the methodologies. The proposed GO methodology is targeted to ordinary software developer. We took a similar approach. To collect feedback that whether developers with no prior experience in agent development can apply the GO methodology and use MADE for agent development, we selected a small group of computer engineering students to use GO methodology, Goal Net and MADE for intelligent agent system modeling and development in their final year projects. They successfully developed goal autonomous agents in their individual systems. Compared with JADE, a leading java agent development tools, MADE provides Goal Net designer for designing agent mental model and supports seamless connections from agent mental model to agent implementations. Using JADE, the students found there is a big gap from implementing a dummy agent to make the dummy agent an intelligent agent.

CHAPTER 6

AGENTS MEDIATED E-LEARNING

The past decade's boom in computer technology rapidly advanced the e-learning industry and quickly changed how people learn, from traditional lecture style learning to web based learning, e-learning. In the last five years, e-learning has become an important concern of most major organizations. In 2002, a survey by the American Management Association has shown that 80% of the American companies were implementing or using some form of e-learning services in their organizations. Most of the e-learning materials are now focused on information transmission. While this is undoubtedly useful, a shift to knowledge intensive learning environments has yet to be made in order to build significant learning services enabling learners to achieve real competency gains. This chapter presents an agent-based e-learning approach for providing personalized, adaptive e-learning services based on the proposed goal-oriented modeling methodology.

6.1 E-learning Necessitates Personalized and Intelligent Agents

With the development of Internet applications, E-commerce/business revolutionized the way companies sold their products and services and the way businesses addressed customer needs and concerns [Wilderman, 2000]. These applications increased productivity and reduced cost while increasing customer satisfaction and maintaining their competitiveness. E-learning, as another type of Internet application, is now becoming very popular as companies rethink almost every aspect of the way their employees work in the enterprise [Osmar, 2002]. In this era of rapid change, large amounts of new product, market, and competitive information are emerging. Employees are expected to learn frequently so as to compete effectively. However, employees usually have different skill sets and have different learning requirements. Traditional instructor-led training and on-line training cannot scale to meet these new learning challenges. E-learning, defined as Internet-enabled or Internet-enhanced learning, aims to provide the tools to create personalized learning path and to be able to dynamically readapt learning paths according to user feedbacks and environment changes in order to optimize the acquisition of needed competencies [Osmar, 2002; Garro, 2002].

Unlike on-line training where thousands of static pages of content were posted on the web, E-learning sites contain a variety of media and learning objects, from many different subject matter experts. Employees or learners will be able to choose what, when, where, how, and how much they are ready to undertake in their requirements. A learning object is a self-contained, tagged object. These learning objects will be targeted to learners when they need them and only to those who need them. Pre-assessments will identify the gap

between what learners already know and what they need to know to effectively do their jobs. Post-assessments will confirm if they retained the knowledge. In short, e-learning provides individualized learning roadmaps for employees or learners to track learning progress based upon business objectives. E-learning is targeted to move from a content-centric model such as on-line training, to a rich, personalized, learner-centric model that will touch everyone associated with the enterprise, including partners and customers.

However, most of the existing e-learning systems are still in the style of "information transmission" [Tuker, 2002]. One of the major limitations is that: ***E-learning systems are often too disconnected from the learner's current preferences and goals.*** For instance, E-learning systems often propose a learning path that does not reflect the current user's needs, interests, etc. E-learning researchers found that [Osmar, 2002]:

- ***E-learning should be learner centric***: E-learning systems should put the user/learner at the centre, and also become a key component for managing individual knowledge. In particular, e-learning systems should help the learner in continuously assessing the state of their knowledge, and recommending an effective learning path.

- ***E-learning should be highly personalized***: E-learning systems should develop a very good knowledge of the learner in order to personalize the learning experience, therefore maximizing the effectiveness of learning. In particular, e-learning systems should take into account the learner's learning style, interests, preferences, current activities and goals.

Providing knowledge intensive e-learning systems has remained a challenge. Recent advances in the field of intelligent agents have shown potential for providing personalized

adaptive e-learning services in web based e-learning environments. Agent technologies are well suited to carry out the main activities involved in e-learning [Osmar, 2002; Garro, 2002; Silveira, 2002]. In fact, most of the e-learning systems are distributed systems. Those activities involved in e-learning systems require communications between distributed learning objects, sensing and monitoring of the environment and autonomous operations. E-learning agents have the ability to learn and reason. They are proactive, interactive, adaptive and autonomous. They are able to perform complex operations based on their goals, messages received and environment changes.

In general, e-learning is the delivery of education and training courses over the Internet and/or Intranet. It can be defined as a mixture of content (on-line courses or courseware) and communication (reaching online, emails, discussion forums). But it is not just about placing classes online to address *training* issues. E-learning encompasses training, education, information, communication, collaboration, knowledge management and performance management. It addresses *business* issues such as reducing costs, providing greater access to information and accountability for learning, and increasing employee competence and competitive agility. Therefore, E-learning is a critical element of any enterprise workforce optimization initiative.

Goal-oriented modeling method proposed in this research has the rich ability to model the pedagogical goals underlying the learning situation: in any given learning situation there are specific objectives defining what is to be learned and the desirable level of competence that is to be achieved. This affects which learning objects are relevant and how learning objects should be selected, and how they must be adapted for learner's preferences.

In this chapter, we explore the potential of proposed goal-oriented modeling methodology and goal autonomous agents in an e-learning environment. The challenge is: how to model the agents' knowledge, goals, behaviors, etc. so that the agents can provide knowledge intensive learning, in other words, personalized adaptive learning services to help learners to achieve their learning goals. In the following, we describe an e-learning scenario to show how the proposed methodology can be used to model and develop the agent-based e-learning systems.

6.2 A Multi-agent System for an E-learning Problem

In Chapter 5, we described an E-learning example, and then analyzed it using proposed goal-oriented methodology. Given a requirement for providing e-learning services to each team, which will be formed for every newly tendered software project in a software company, and get the team members be ready for the roles they will be played in development of the project we consider a multi-agent system approach. The multi-agent e-learning system is an ***open*** system. In the deployment time, the total number of the projects that will be formed is unknown. As a result, the number of the e-learning service agents for providing various services such as learning-path-generation etc. is also un-known. Unlike traditional software system, the multi-agent e-learning system is a ***service-oriented*** system in an e-learning grid environment. It may compose other e-learning services (e.g. learning courses) provided in the e-learning grid into the system [Silveira, 2002]. As normally, it is not possible for a software company to design and provide all the e-learning courses by the company itself. The e-learning grid environment makes the e-learning services provided by various e-learning providers interrelated. Every

e-learning system in the grid might need to compose some services from other e-learning services in the grid.

A preliminary Goal Net was constructed based on the analysis and was depicted in Figure 5.7. As shown, the goal of the e-learning service was decomposed to two sub-goals. Then each of the two sub-goals was further decomposed to three sub-goals respectively. The six sub-goals at the bottom layer can be further decomposed in a real e-learning application. As described in Chapter 4, for each composite state together with its decomposition, a sub-net can be identified as an agent. So based on the Figure 5.7, and the agent identification rules, an e-learning system has been organized with at least nine types of agents: learning service agent, learner preparation agent, learning agent, role identification agent, learner selection agent, pre-assessment agent, learning path generation agent, course delivery agent and post-assessment agent.

In this case study, we focus on two sub-goals of the e-learning systems, the personalized learning path generation and learning object delivery. We assume that the roles of a project are known; the learners have been decided; the pre-assessment is a list of questionnaires. Furthermore, the post-assessment for each course is prepared by the course providers. The scenario for this case study is described as the following:

A company needs to train an Oracle developer to be a database administrator (DBA) for a coming project via e-learning services. This is because with e-learning services, the employee can learn the courses in the company and he can still be on duty for some other projects. The other reason to select an e-learning service is that they hope the e-learning service can provide a personalized training for the employee since he has many years of

experience in Oracle database development. So the duration for the training could be reduced.

In this scenario, we want to provide a personal e-learning assistant to assist the employee, i.e. the learner, for the Oracle database training. The requirements are that 1) the agents should provide the courses based on the company's budget, duration restriction and the learner's technical expectation; 2) the agents should provide a personalized learning path to meet the learner's current technical skills.

Suppose there are three related course providers in the e-learning grid. All the e-learning providers provide Oracle courses for DBA training. However, the course organization, price, duration, volume of the technical contents are all different from each provider. And they might be changed by the service provider at anytime in a dynamic environment. Table 6.1 lists the details of the courses provided by the three course providers.

Course Provider	Course	Price (S$)	Duration (hrs)	Technical Grade
Provider 1	Quick DBA	1100	5 * 8 = 40	3
Provider 2	PL/SQL	400	5 * 8 = 40	1
	DBA fundamental I	650	5 * 8 = 40	3
	DBA fundamental II	400	5 * 8 = 40	4
	Performance Tuning	400	5 * 8 = 40	6
Provider 3	DBA I	700	5 * 8 = 40	2
	DBA II	500	5 * 8 = 40	4

Table 6.1 The courses list

The courses exist in the form of learning objects on an e-learning grid. A course provider stores a learning object on multiple e-learning grids to increase the quality of service. A course is usually split into one or more learning objects (LOs) to increase reusability and flexibility. Each e-learning provider has metadata to describe the courses it provides. The metadata includes the information about price, duration, prerequisite courses, and the information of the learning objects, etc. For example, the course *PL/SQL* is split to two LOs, *Standard SQL* and *PL/SQL*. Every course provider provides a series of courses to

meet a certain requirement. In order to provide personalized learning service the learning assistant agents should be able to compose the courses from the three learning providers.

The courses have relations with other courses in terms of prerequisites. For example, in the Oracle database courses, one of the prerequisites of the course *DBA Fundamental I* is the course *PL/SQL*. The learning paths for individual learners will be generated based on the relationships. For example, a course relationship diagram is shown in Figure 6.1. The node g_n is the course a learner wants to learn while g_0, g_1, g_2 and g_3 are the courses the learner must learn before he can learn g_n. In our system, each course is regarded as a goal. So, in order to learn the course g_n, that is, reach the goal g_n the learner must reach the goals g_0, g_1, g_2 and g_3 respectively.

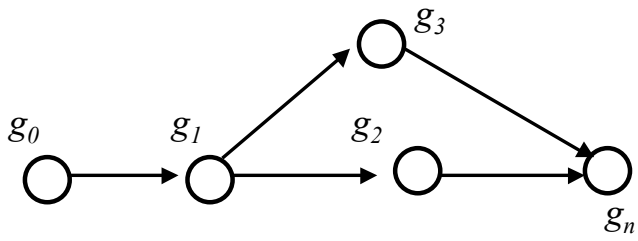

Figure 6.1 The course relationships

According to the metadata of the courses from the three course providers, we know the Oracle courses have the following relationships respectively as shown in Figure 6.2.

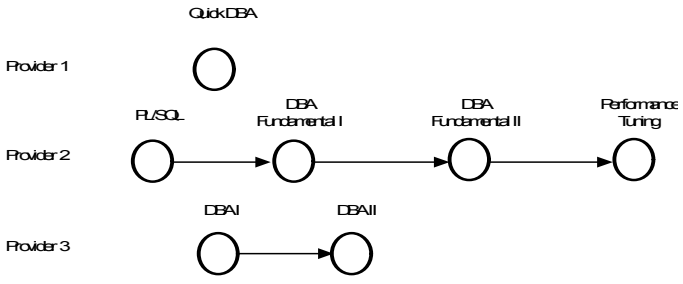

Figure 6.2 The Oracle course relationships

In the next section, we show the detailed Goal Net design for modeling the learning path generation agent and course delivery agent.

6.3 Modeling Personalized E-learning Services Using Goal Net

The goal of the learning assistant agent is to assist a learner to complete the DBA courses. To provide a personalized assistant to the particular learner, the agent must get the results of pre-assessment. The collected pre-assessment information will be used to select the learning objects (LOs) of a course particularly for the individual learner. To complete the DBA training, the learner must complete the courses provided by any of the three learning providers. However, we need to compose the courses from the three learning providers, so that we can have more ways to finish the courses based on different requirements from individual learners. So all the courses from the three learning providers will be considered together and each course becomes a sub-goal towards the final goal, *complete the DBA courses*. When a learner requests service, the agent should obtain the learner's current skills or the results from pre-assessment and then obtain the requirement or constraints about the courses from the learner to decide the learning path for him. After the learner finishes the courses, the agent should conduct a test for the learner to evaluate his achievement. Table 6.2 lists the sub-goals identified.

Goal	Sub-goal/State	ID	Type
DBA Course Learned	Start state	g_s	Atomic
	Learner skills obtained	g_1	Atomic
	Course information obtained	g_2	Atomic
	PL/SQL learned	g_3	Composite
	DBA fundamental I learned	g_4	Composite
	DBA fundamental II learned	g_5	Composite
	Performance tuning learned	g_6	Composite
	DBA I learned	g_7	Composite
	DBA II learned	g_8	Composite
	Quick DBA learned	g_9	Composite
	DBA test finished	g_{10}	Atomic
	End state	g_e	Atomic

Table 6.2 The identified goals for the learning assistant agent

According to the metadata of the courses, course provider 1 provides only one course *Quick DBA*; course provider 2 provides four courses and they must be taken in order of *PL/SQL*, *DBA Fundamental I*, *DBA Fundamental II*, and *Performance Tuning*; course provider 3 provides two courses and they must be taken in order of *DBA I*, and *DBA II*. After investigation of the courses provided by the three learning providers, we have new knowledge about the courses: 1) after the course *PL/SQL* is taken from the course provider 2, the learner can take the course *DBA I* from the course provider 3 or take the course *Quick DBA* from the course provider 1; 2) after the course *DBA Fundamental I* is taken, the learner can take the course *DBA II* from the course provider 3; and 3) after the course *DBA I* is taken, the learner can take the course *DBA Fundamental II* from the course provider 2. After the required courses are completed, the learner needs to take a test for a post-assessment. Then the training is finished. So by doing this, the learner can connect to different learning course provider from one course to the next course. The prerequisite relationships among courses become transitions among the sub-goals. Table 6.3 lists the transitions identified.

Transition	From	To	ID	Task
Get pre-assessment results	g_s	g_1	t_1	Get pre-assessment results
Get course information	g_1	g_2	t_2	Get course information
Connect to the course	g_2	g_3	t_3	Connect to the course
Connect to the course	g_3	g_4	t_4	Connect to the course
Connect to the course	g_4	g_5	t_5	Connect to the course
Connect to the course	g_5	g_6	t_6	Connect to the course
Connect to the course	g_2	g_7	t_7	Connect to the course
Connect to the course	g_3	g_7	t_8	Connect to the course
Connect to the course	g_7	g_8	t_9	Connect to the course
Connect to the course	g_4	g_8	t_{10}	Connect to the course
Connect to the course	g_3	g_9	t_{11}	Connect to the course
Connect to the course	g_7	g_5	t_{12}	Connect to the course
Take test	g_9	g_{10}	t_{13}	Take test
Take test	g_6	g_{10}	t_{14}	Take test
Take test	g_8	g_{10}	t_{15}	Take test
Finish	g_{10}	g_s	t_{16}	Finish

Table 6.3 The identified transitions for the learning assistant agent

Then we have the Goal Net as depicted in Figure 6.3.

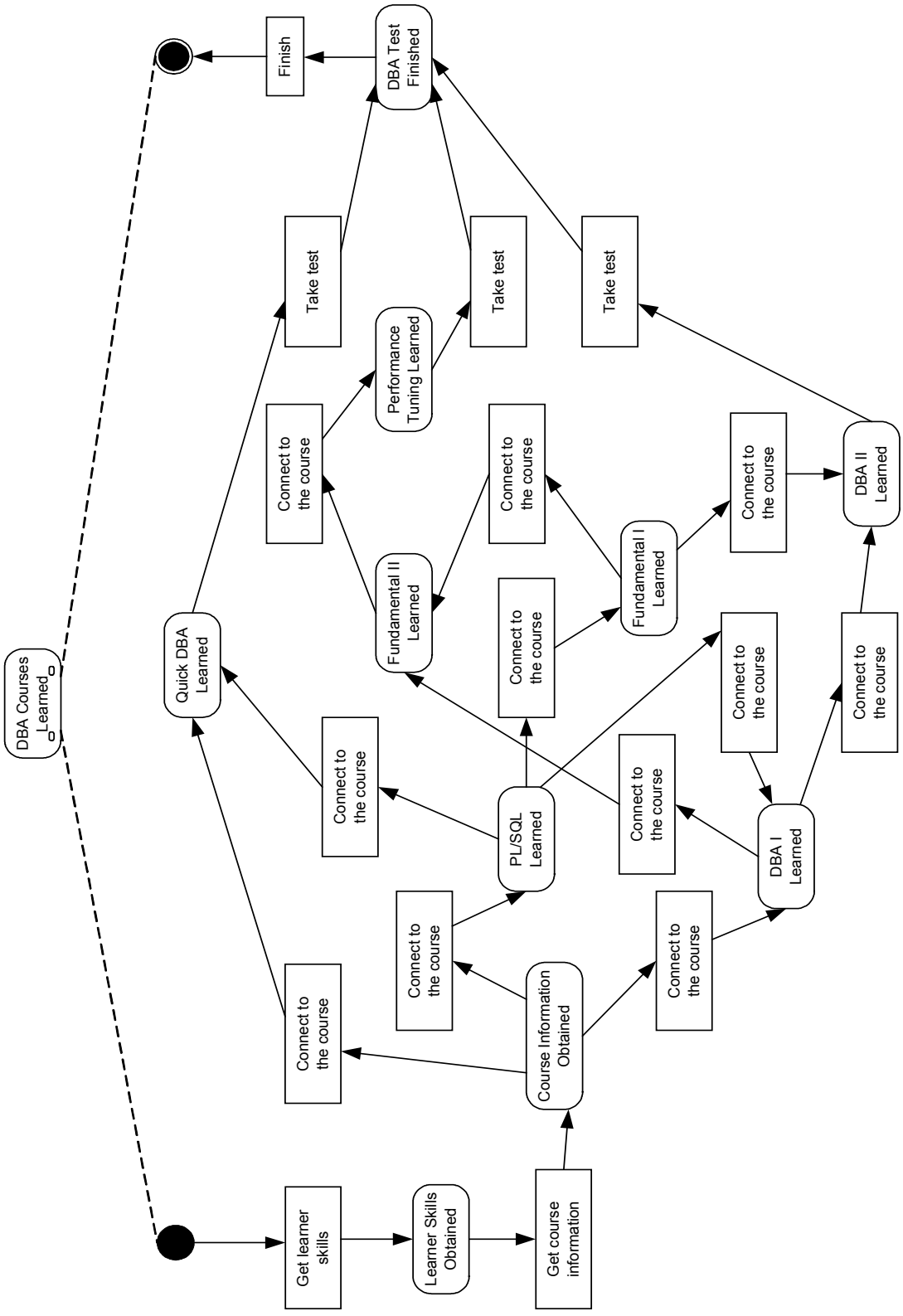

Figure 6.3 The Goal Net for the learning path generation agent

After the path generation agent selects the next course for the learner, it will guide the learner to proceed with the course and select suitable learning objects for the learner. To learn a course is also a complicate problem. A learner needs to register for the course and pay the fee. Then according to his current skills to select suitable learning objects. After a learning object is selected, the learning object should be delivered to the learner. After the learner finish studying with the current learning object, he should proceed to the next suitable one until he finishes all the learning objects. Table 6.4 lists the goals identified for learning the course PL/SQL while Table 6.5 lists the transitions between two goals.

Goal	Sub-goal/State	ID	Type
Learn a course	Start state	g_{3s}	Atomic
	Course registered	g_{31}	Atomic
	LO selected	g_{32}	Atomic
	LO learned	g_{33}	Atomic
	End state	g_{3e}	Atomic

Table 6.4 The identified goals for learning a course

Transition	From	To	ID	Task
Register a course	g_{3s}	g_{31}	t_{31}	Register a course
Select learning object	g_{31}	g_{32}	t_{32}	Select a learning object
Deliver the learning object	g_{32}	g_{33}	t_{33}	Deliver a learning object
Prepare for the next LO	g_{33}	g_{31}	t_{34}	Prepare for the next LO
Finish	g_{33}	g_{3e}	t_{35}	Finish

Table 6.5 The identified transitions for learning a course

Figure 6.4 gives the Goal Net for learning a course.

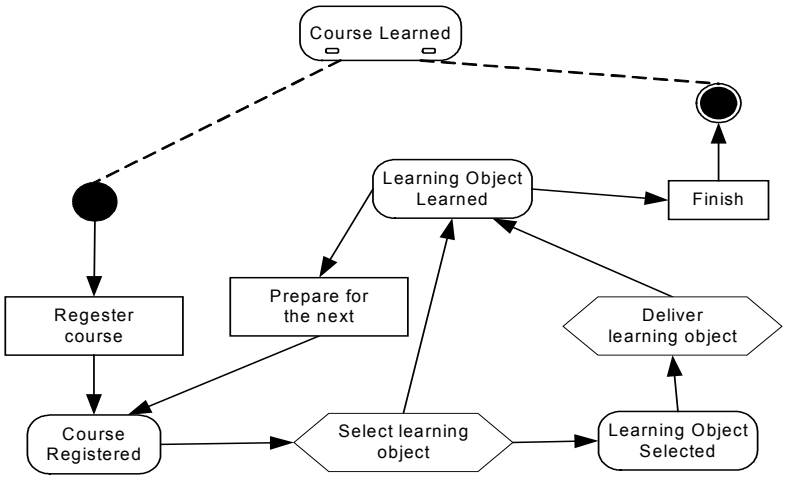

Figure 6.4 The Goal Net for learning a course

As shown in the figure, the action selection mechanisms defined here for selecting the learning objects and selecting the learning object servers are based on Bayesian networks. The detailed information about the task selection will also be elaborated in the experiment section.

In summary, in this case study, we design the detailed Goal Net for the learning path generation agent and the course delivery agent. The Goal Net shown in Figure 6.3 is a decomposition of the goal *learning path generation* in Figure 5.7 while the Goal Net shown in Figure 6.4 is a decomposition of the goal *course delivery* in Figure 5.7. So the two Goal Nets will be the goal of a learning path agent and the goal of a course delivery agent respectively.

6.4 E-Learning Agent System Design and Development

After the e-learning problem is modeled with the Goal Net, multiple e-learning agents are identified and thus a multi-agent e-learning system is generated. Figure 6.5 gives a multi-agent e-learning system architecture.

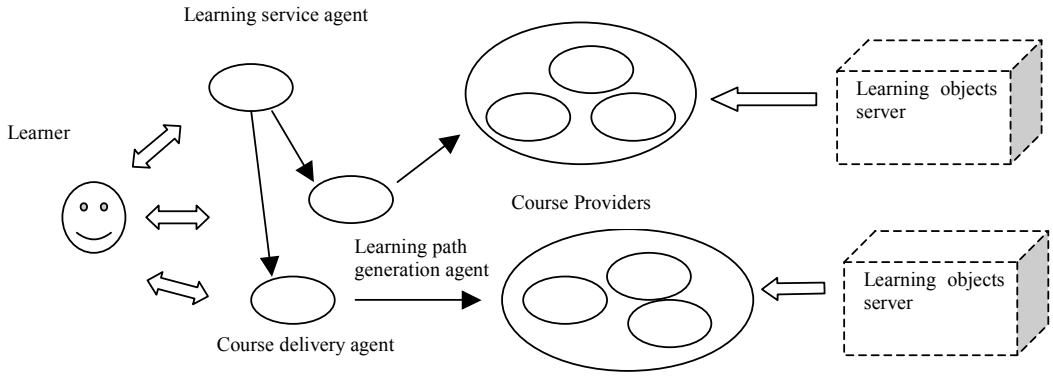

Figure 6.5 The e-learning system architecture

As showed in this figure, a multi-agent e-learning system contains agents identified using the Goal Net. The learning object servers store all the learning objects of courses, course metadata, and manage all the courses. It provides services to add new learning objects, delete learning objects or modify learning objects. So the availability of a learning object on a learning object server is dynamic. The course delivery agents need to access the learning object servers to retrieve required learning objects dynamically.

The multi-agent e-learning system shown in Figure 6.5 can handle many learners concurrently. When the e-learning service agent serves a learner, it will dispatch the work to the lower level agents: the learner agent and the learning agent who will dispatch the work further to their child agents, learning path generation agent and course delivery agent respectively. After the e-learning service agent dispatches the work, it can accept

the next learner. Each agent also can pursue another goal when its current goal pursuit is waiting for the feedback from other agents.

The Goal Nets designed for this case study were developed using the Goal Net Designer and they were save in the knowledge base. We used relational database Oracle as the knowledge base of the agents. We also used it as the learning object servers to simulate the course providers. The agents are developed using the agent framework with the Agent Creator. The system structure is depicted in Figure 6.6.

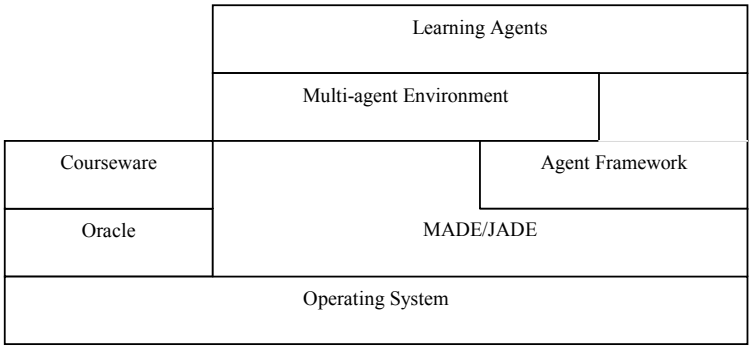

Figure 6.6 The system development structure

6.5 Experiments

The Goal Net models the dynamic relationships between two goals, which enables an agent can select the next dynamically and decide the suitable action dynamically according to the current situation. So the agent will present both goal autonomy and behavior autonomy. In this case study, we'll present the goal autonomy and behavior autonomy through two experiments. The first experiment will present the goal autonomy of an agent by showing how a learning path agent generates the learning path based on the learner's requirement and the environment changes dynamically. The second one will present the behavior autonomy by showing how a learning object delivery agent selects

the learning object according to the learner's current skills and how the agent decides the learning object server to retrieve the learning object.

6.5.1 The Experiments for the Learning Path Generation Agent

The courses provided by different learning grids have different technical grades. The technical grade is a measurement of the course content. Normally the longer a course duration is, the more the technical content, and therefore the higher the technical grade. For example, the technical grade of course from the course provider 1 is 2; the technical grade of course from the course provider 2 is 6; the technical grade of course from the course provider 3 is 4. Suppose a learner will get more achievement if he takes a course marked with higher technical grade. We let the personal achievement by finishing a course is the technical grade the course has. Table 6.1 also lists the achievement (technical grade) after the learner finishes the corresponding course. The price, duration and the technical grade are important factors to influence the course selection. With traditional learning system, a learner can select the course provider based on the value of factors he has. Once he starts learning, it is difficult to change to other learning providers without extra cost. With the help of the learning assistant agent, the learner can dynamically select the next course from a different learning provider when the values of factors have changed. As shown in Figure 6.3, a learner can learn the Oracle courses in different learning paths. For example, the learning path could be 1) g_3, g_4, g_5, g_6; 2) g_3, g_7, g_8; 3) g_3, g_9; and 4) g_3, g_4, g_8; etc. The real learning path will be generated dynamically during the learning process.

Suppose C is the budget, T is the maximum duration for the training, and A is the minimum achievement expected, we define:

1) $C(g_i) = Cost(g_i)$ $(1 \le i \le 10)$ is the price for the course defined by g_i;

2) $A(g_i) = Achievement(g_i)$ = technical grade value $(1 \le i \le 10)$;

3) $T(g_i) = Time(g_i)$ = duration for reaching g_i $(1 \le i \le 10)$;

4) $Cons(g_0, g_i) = Constraint(g_0, g_i) = C(g_i, g_i) \le C \wedge T(g_i, g_i) \le T \wedge A(g_i, g_i) \ge A$

   where $0 \le i, j \le 10$;

5) $Index(C(g_i, g_j), A(g_i, g_j)) =$

$$\alpha * \frac{C(g_i, g_j)}{C(g_i, g_j) + C} + (1 - \alpha) * \frac{A}{A(g_i, g_j) + A} \qquad (6.1)$$

   where α is a coefficient, and $0 \le \alpha \le 1$, $0 \le i, j \le 10$.

So, in the example the learning assistant agent will generate the learning path for the learner for the DBA courses dynamically during the learning process. In this case study, we consider the following four cases to test how the agent selects the next goal – generate the learning path dynamically. Assume the agent has achieved the sub-goal g_2.

Case 1: There is no restriction on budget and time. The learner expects more achievement.

In this case, there is no constraint. Assuming $C = 2000$, $A = 1$ and $\alpha = 0$, based on the algorithm GSA1, we have:

$G = \{g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9, g_{10}\}$, $S = \{\}$, $I = \{\}$,
The initial setting:
$AGS(g_2) = \{g_3, g_7, g_9\}$;
$C(g_2, g_3) = 400$, $A(g_2, g_3) = 1$, $\pi(g_3) = null$, $l(g_3) = 0$, $Index(g_2, g_3) = 0.5$;
$C(g_2, g_7) = 700$, $A(g_2, g_7) = 2$, $\pi(g_7) = null$, $l(g_7) = 0$, $Index(g_2, g_7) = 0.33$;
$C(g_2, g_9) = 1100$, $A(g_2, g_9) = 3$, $\pi(g_9) = null$, $l(g_9) = 0$, $Index(g_2, g_9) = 0.25$;
$C(g_2, g_2) = 0$, $A(g_2, g_2) = 0$, $S = \{g_2\}$, the index values of others are ∞;

$I = \{Index(g_2, g_3), Index(g_2, g_7), Index(g_2, g_9)\}$;
$Index(g_2, g_9) = \min\{Index(g_2, g_3), Index(g_2, g_7), Index(g_2, g_9)\}$;
So, initialize the path as $P_0 = \{g_9\}$;
Loop 1:
$S = \{g_2, g_9\}$, $(\min\{Index(g_2, g_3), Index(g_2, g_7), Index(g_2, g_9)\} = 0.25)$;
$I = \{Index(g_2, g_3), Index(g_2, g_9)\}$;
$AGS(g_9) = \{g_{10}\}$;
$C(g_2, g_{10}) = 1100$, $A(g_2, g_3) = 3$, $\pi(g_{10}) = g_9$, $l(g_{10}) = 1$, $Index(g_2, g_{10}) = 0.25$;
Generate the path:
$P(l(g_{10})) = P(1) = g_{10}$, $P(0) = \pi(g_{10}) = g_9$;
So, the path becomes $P_1 = \{g_9, g_{10}\}$;
Loop 2:
$S = \{g_2, g_9, g_7\}$, $(\min\{Index(g_2, g_3), Index(g_2, g_7)\} = 0.33)$;
$I = \{Index(g_2, g_3)\}$;
$AGS(g_7) = \{g_5, g_8\}$;
$C(g_2, g_5) = 1100$, $A(g_2, g_5) = 4$, $\pi(g_5) = g_7$, $l(g_5) = 1$, $Index(g_2, g_5) = 0.2$;
$C(g_2, g_8) = 1200$, $A(g_2, g_8) = 4$, $\pi(g_8) = g_7$, $l(g_8) = 1$, $Index(g_2, g_8) = 0.2$;
$I = \{Index(g_2, g_3), Index(g_2, g_5), Index(g_2, g_8)\}$;
Loop 3:
$S = \{g_2, g_9, g_7, g_8\}$, $(\min\{Index(g_2, g_3), Index(g_2, g_5), Index(g_2, g_8)\} = 0.2)$;
$I = \{Index(g_2, g_3), Index(g_2, g_5)\}$;
$AGS(g_8) = \{g_{10}\}$;
$C(g_2, g_{10}) = 1200$, $A(g_2, g_{10}) = 4$, $\pi(g_{10}) = g_8$, $l(g_{10}) = 2$, $Index(g_2, g_{10}) = 0.2$;
Generate the path:
$P(l(g_{10})) = P(2) = g_{10}$, $P(1) = \pi(g_{10}) = g_8$, $P(0) = \pi(g_8) = g_7$;
So, the path becomes $P_2 = \{g_7, g_8, g_{10}\}$;
Loop 4:
$S = \{g_2, g_9, g_7, g_8, g_5\}$, $(\min\{Index(g_2, g_3), Index(g_2, g_5)\} = 0.2)$;
$I = \{Index(g_2, g_3)\}$;
$AGS(g_5) = \{g_6\}$;
$C(g_2, g_6) = 1500$, $A(g_2, g_6) = 6$, $\pi(g_6) = g_5$, $l(g_6) = 2$, $Index(g_2, g_6) = 0.14$;
$I = \{Index(g_2, g_3), Index(g_2, g_6)\}$;
Loop 5:
$S = \{g_2, g_9, g_7, g_8, g_5, g_6\}$, $(\min\{Index(g_2, g_3), Index(g_2, g_6)\} = 0.14)$;
$I = \{Index(g_2, g_3)\}$;
$AGS(g_5) = \{g_{10}\}$;
$C(g_2, g_{10}) = 1500$, $A(g_2, g_{10}) = 6$, $\pi(g_{10}) = g_6$, $l(g_{10}) = 3$, $Index(g_2, g_{10}) = 0.14$;
Generate the path:
$P(l(g_{10})) = P(3) = g_{10}$, $P(2) = \pi(g_{10}) = g_6$, $P(1) = \pi(g_6) = g_5$, $P(0) = \pi(g_5) = g_7$;
So, the path becomes $P_3 = \{g_7, g_5, g_6, g_{10}\}$;
Loop 6:
$S = \{g_2, g_9, g_7, g_8, g_5, g_6, g_3\}$, $(\min\{Index(g_2, g_3)\} = 0.5)$;
$I = \{\}$;
$AGS(g_3) = \{g_4, g_7, g_9\}$;
$C(g_2, g_4) = 1050$, $A(g_2, g_4) = 3$, $\pi(g_4) = g_3$, $l(g_4) = 1$, $Index(g_2, g_4) = 0.25$;
$C(g_2, g_7)$ is no change because $Index(g_2, g_7) = 0.33$ is same as previous value;
$C(g_2, g_9)$ is no change because $Index(g_2, g_9) = 0.25$ is same as previous value;
$I = \{Index(g_2, g_4)\}$;
Loop 7:
$S = \{g_2, g_9, g_7, g_8, g_5, g_6, g_3, g_4\}$, $(\min\{Index(g_2, g_4)\} = 0.25)$;
$I = \{\}$;
$AGS(g_4) = \{g_5, g_8\}$;
$C(g_2, g_5)$ is no change because $Index(g_2, g_5) = 0.2$ is same as previous value;
$C(g_2, g_8)$ is no change because $Index(g_2, g_8) = 0.2$ is same as previous value;
Generate the path:
$P(l(g_{10})) = P(3) = g_{10}$, $P(2) = \pi(g_{10}) = g_6$, $P(1) = \pi(g_6) = g_5$, $P(0) = \pi(g_5) = g_7$
So, the path is $P_4 = \{g_7, g_5, g_6, g_{10}\}$.

In this test case, it shows the anytime feature of the goal selection algorithm. First we have a path P_0, then it is replaced by P_1 and finally the optimal path P_4 was obtained. So in the time sensitive environment the agent always has a "best for current" goal to pursue.

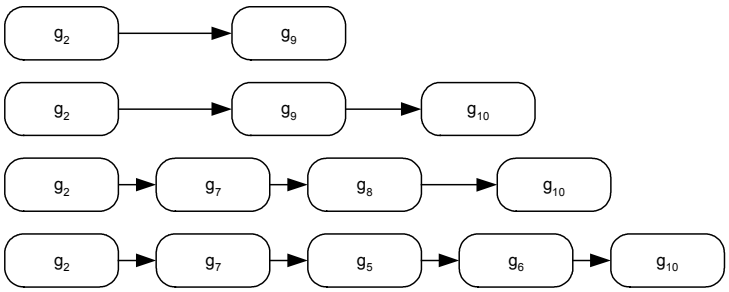

Figure 6.7 The goal selection process for Case 1

Figure 6.7 lists the paths generated during the optimization process. The last one is the optimal result. It indicates the learner will get the highest grade (achievement) by following this path based on the current situation.

Case 2: The budget is S$1300, there is no time limit. The learner expects more achievement.

In this case, C = 1300. Assuming A = 1 and α = 0.5, based on the algorithm GSA2, we have:

G = {g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9, g_{10}}, S = {}, I = {},
The initial setting:
AGS(g_2) = {g_3, g_7, g_9};
C(g_2, g_3) = 400, A(g_2, g_3) = 1, π(g_3) = null, l(g_3) = 0, Index(g_2, g_3) = 0.37;
C(g_2, g_7) = 700, A(g_2, g_7) = 2, π(g_7) = null, l(g_7) = 0, Index(g_2, g_7) = 0.342;
C(g_2, g_9) = 1100, A(g_2, g_9) = 3, π(g_9) = null, l(g_9) = 0, Index(g_2, g_9) = 0.355;
C(g_2, g_2) = 0, A(g_2, g_2) = 0, S = {g_2}, the index values of others are ∞;
I = {Index(g_2, g_3), Index(g_2, g_7), Index(g_2, g_9)};
Index(g_2, g_7) = min{Index(g_2, g_3), Index(g_2, g_7), Index(g_2, g_9)};
So, initialize the path as P_0 = {g_7};
Loop 1:
S = {g_2, g_7}, (min{Index(g_2, g_3), Index(g_2, g_7), Index(g_2, g_9)} = 0.342);
I = {Index(g_2, g_3), Index(g_2, g_9)};
AGS(g_7) = {g_5, g_8};
C(g_2, g_5) = 1100, A(g_2, g_5) = 4, π(g_5) = g_7, l(g_5) = 1, Index(g_2, g_5) = 0.33;

$C(g_2, g_8) = 1200$, $A(g_2, g_8) = 4$, $\pi(g_8) = g_7$, $l(g_8) = 1$, $Index(g_2, g_8) = 0.34$;
$I = \{Index(g_2, g_3), Index(g_2, g_5), Index(g_2, g_8), Index(g_2, g_9)\}$;
Loop 2:
$S = \{g_2, g_7, g_5\}$, $(\min\{Index(g_2, g_3), Index(g_2, g_5), Index(g_2, g_8), Index(g_2, g_9)\} = 0.33)$;
$I = \{Index(g_2, g_3), Index(g_2, g_8), Index(g_2, g_9)\}$;
$AGS(g_5) = \{g_6\}$;
$C(g_2, g_6) = 1500 > 1300$, the cost exceeds the constraint;
Loop 3:
$S = \{g_2, g_7, g_5, g_8\}$, $(\min\{Index(g_2, g_3), Index(g_2, g_8), Index(g_2, g_9)\} = 0.34)$;
$I = \{Index(g_2, g_3), Index(g_2, g_9)\}$;
$AGS(g_8) = \{g_{10}\}$;
$C(g_2, g_{10}) = 1200$, $A(g_2, g_{10}) = 4$, $\pi(g_{10}) = g_8$, $l(g_{10}) = 2$, $Index(g_2, g_{10}) = 0.34$;
Generate the path:
$P(l(g_{10})) = P(2) = g_{10}$, $P(1) = \pi(g_{10}) = g_8$, $P(0) = \pi(g_8) = g_7$;
So, the path becomes $P_1 = \{g_7, g_8, g_{10}\}$;
Loop 4:
$S = \{g_2, g_7, g_5, g_8, g_9\}$, $(\min\{Index(g_2, g_3), Index(g_2, g_9)\} = 0.355)$;
$I = \{Index(g_2, g_3)\}$;
$AGS(g_9) = \{g_{10}\}$;
$C(g_2, g_{10}) = 1100$, $A(g_2, g_{10}) = 3$, $Index(g_2, g_{10}) = 0.355 > 0.34$ (previous value);
Loop 5:
$S = \{g_2, g_7, g_5, g_8, g_9, g_3\}$, $(\min\{Index(g_2, g_3)\} = 0.37)$;
$I = \{\}$;
$AGS(g_3) = \{g_4, g_7, g_9\}$;
$C(g_2, g_4) = 1050$, $A(g_2, g_4) = 3$, $\pi(g_4) = g_3$, $l(g_4) = 1$, $Index(g_2, g_4) = 0.348$;
$C(g_2, g_7) = 1100$, $A(g_2, g_7) = 2$, $Index(g_2, g_7) = 0.396 > 0.342$ (previous value);
$C(g_2, g_9) = 1500 > 1300$, the cost exceeds the constraint;
$I = \{Index(g_2, g_4)\}$;
Loop 6:
$S = \{g_2, g_7, g_5, g_8, g_9, g_3, g_4\}$, $(\min\{Index(g_2, g_4)\} = 0.348)$;
$AGS(g_4) = \{g_5, g_8\}$;
$I = \{\}$;
$C(g_2, g_5) = 1450$, the cost exceeds the constraint;
$C(g_2, g_8) = 1550$, the cost exceeds the constraint;
Generate the path:
$P(l(g_{10})) = P(2) = g_{10}$, $P(1) = \pi(g_{10}) = g_8$, $P(0) = \pi(g_8) = g_7$;
So, the path is $P_2 = \{g_7, g_8, g_{10}\}$.

In this test case, it shows the anytime feature and constraint consideration of the goal selection algorithm. First we have a path P_0, then it is replaced by P_1 and finally the optimal path P_2 was obtained. However, the optimal path may not be still true due to some environment changes. In the following three test cases, the goal paths are dynamically adjusted according to the environment changes.

Case 3: There is no budget limitation and time restriction. The learner expects more achievement. However, after first week, the company limits the time to two weeks due to an urgent arrangement.

In this case, $T = 80 - 40 = 40$. Before the company limits the time, the optimal path can be obtained from the computation of case 1. So, the path is $P = \{g_7, g_5, g_6, g_{10}\}$. After the first week the path is $P = \{g_5, g_6, g_{10}\}$. Assuming the same setting as that in the case 1, that is, $C = 2000$, $A = 1$ and $\alpha = 0$, based on the algorithm GSA2 at the goal g_7, we have:

$G = \{g_7, g_5, g_6, g_8, g_{10}\}$, $S = \{\}$, $I = \{\}$,
The initial setting:
$AGS(g_7) = \{g_5, g_8\}$;
$T(g_7, g_5) = 40$, $A(g_7, g_5) = 4$, $\pi(g_5) = null$, $l(g_5) = 0$, $Index(g_7, g_5) = 0.2$;
$T(g_7, g_8) = 40$, $A(g_7, g_8) = 4$, $\pi(g_8) = null$, $l(g_8) = 0$, $Index(g_7, g_8) = 0.2$;
$T(g_7, g_7) = 0$, $A(g_7, g_7) = 0$, $S = \{g_7\}$, $I = \{Index(g_7, g_5), Index(g_7, g_8)\}$;
$Index(g_7, g_5) = \min\{Index(g_7, g_5), Index(g_7, g_8)\}$;
So, initialize the path as $P_0 = \{g_5\}$;
Loop 1:
$S = \{g_7, g_5\}$, $(\min\{Index(g_7, g_5), Index(g_7, g_8)\} = 0.2)$;
$I = \{Index(g_7, g_8)\}$;
$AGS(g_5) = \{g_6\}$;
$T(g_7, g_6) = 80 > 40$, the time exceeds the constraint;
Loop 2:
$S = \{g_7, g_5, g_8\}$, $(\min\{Index(g_7, g_8)\} = 0.2)$;
$I = \{\}$;
$AGS(g_8) = \{g_{10}\}$;
$T(g_7, g_{10}) = 40$, $A(g_7, g_{10}) = 4$, $\pi(g_{10}) = g_8$, $l(g_{10}) = 1$, $Index(g_7, g_{10}) = 0.2$;
Generate the path:
$P(l(g_{10})) = P(1) = g_{10}$, $P(0) = \pi(g_{10}) = g_8$;
So, the path becomes $P_1 = \{g_8, g_{10}\}$.

In this test case, it shows that the environment changes will affect the goal achievement and the Goal Net can adapt to the changes dynamically. A new goal will be generated towards the final goal achievement. In this test, the original path $\{g_5, g_6, g_{10}\}$ was changed to $\{g_8, g_{10}\}$.

Case 4: The budget is S$1300, There is no time limit. The learner expects more achievement. However, during the learner is learning the course *DBA I* from the Grid 3 to pursue the goal g_7, the price of the course *DBA Fundamental II* and the course *Performance Tuning* from the Grid 2 are changed to S$250 and S$300 respectively.

The original path obtained from the computation of case 2 is $P = \{g_7, g_8, g_{10}\}$. After the first week the path is $P = \{g_8, g_{10}\}$. So the next goal is supposed to be g_8. Assuming the

same setting as that in the case 2, that is, $C = 1300 - 700 = 600$, $A = 1$ and $\alpha = 0.5$, based on the algorithm GSA2, we have:

$G = \{g_7, g_5, g_6, g_8, g_{10}\}$, $S = \{\}$, $I = \{\}$,
The initial setting:
$AGS(g_7) = \{g_5, g_8\}$;
$C(g_7, g_5) = 250$, $A(g_7, g_5) = 4$, $\pi(g_5) = null$, $l(g_5) = 0$, $Index(g_7, g_5) = 0.247$;
$C(g_7, g_8) = 500$, $A(g_7, g_8) = 4$, $\pi(g_8) = null$, $l(g_8) = 0$, $Index(g_7, g_8) = 0.327$;
$C(g_7, g_7) = 0$, $A(g_7, g_7) = 0$, $S = \{g_7\}$, $I = \{Index(g_7, g_5), Index(g_7, g_8)\}$;
$Index(g_7, g_5) = \min\{Index(g_7, g_5), Index(g_7, g_8)\}$;
So, initialize the path as $P_0 = \{g_5\}$;
Loop 1:
$S = \{g_7, g_5\}$, $(\min\{Index(g_7, g_5), Index(g_7, g_8)\} = 0.247)$;
$I = \{Index(g_7, g_8)\}$;
$AGS(g_5) = \{g_6\}$;
$C(g_7, g_6) = 550$, $A(g_7, g_6) = 6$, $\pi(g_6) = g_5$, $l(g_6) = 1$, $Index(g_7, g_6) = 0.31$;
$I = \{Index(g_7, g_8), Index(g_7, g_6)\}$;
Loop 2:
$S = \{g_7, g_5, g_6\}$, $(\min\{Index(g_7, g_8), Index(g_7, g_6)\} = 0.31)$;
$I = \{Index(g_7, g_8)\}$;
$AGS(g_6) = \{g_{10}\}$;
$C(g_7, g_{10}) = 550$, $A(g_7, g_{10}) = 6$, $\pi(g_{10}) = g_6$, $l(g_{10}) = 2$, $Index(g_7, g_{10}) = 0.31$;
Generate the path:
$P(l(g_{10})) = P(2) = g_{10}$, $P(1) = \pi(g_{10}) = g_6$, $P(0) = \pi(g_6) = g_5$;
So, the path becomes $P_1 = \{g_5, g_6, g_{10}\}$;
Loop 3:
$S = \{g_7, g_5, g_6, g_8\}$, $(\min\{Index(g_7, g_8)\} = 0.327)$;
$I = \{\}$;
$AGS(g_8) = \{g_{10}\}$;
$C(g_7, g_{10}) = 500$, $A(g_7, g_{10}) = 4$, $Index(g_7, g_{10}) = 0.327 > 0.31$ (previous value);
Generate the path:
$P(l(g_{10})) = P(2) = g_{10}$, $P(1) = \pi(g_{10}) = g_6$, $P(0) = \pi(g_6) = g_5$;
So, the path is $P_2 = \{g_5, g_6, g_{10}\}$.

In this test case, it also shows how the environment changes affected the goal achievement and how the Goal Net adapted to the changes dynamically. A new goal path was generated towards the better goal achievement. In this test, under the same budget the learner can achieve the technical grade 6 instead of original 4.

Case 5: The budget is S$1300, There is no time limit. The learner expects more achievement. However, during the learner is learning the course *DBA I* from the Grid 3 to pursue the goal g_7, the price of the course *DBA II* is changed to S$750 and the technical grade is changed to 5.

In this test case, the original path obtained from the computation of case 2 is $P = \{g_7, g_8, g_{10}\}$. After the first week the path is $P = \{g_8, g_{10}\}$. So the next goal is supposed to be g_8. Assuming the same setting as that in the case 2, that is, $C = 1300 - 700 = 600$, $A = 1$ and $\alpha = 0.5$, based on the algorithm GSA2, we have:

$G = \{g_7, g_5, g_6, g_8, g_{10}\}$, $S = \{\}$, $I = \{\}$,
The initial setting:
$AGS(g_7) = \{g_5, g_8\}$;
$C(g_7, g_5) = 400$, $A(g_7, g_5) = 4$, $\pi(g_5) = null$, $l(g_5) = 0$, $Index(g_7, g_5) = 0.3$;
$C(g_7, g_8) = 750$, $A(g_7, g_8) = 5$, $\pi(g_8) = null$, $l(g_8) = 0$, $Index(g_7, g_8) = 0.36$;
$C(g_7, g_7) = 0$, $A(g_7, g_7) = 0$, $S = \{g_7\}$, $I = \{Index(g_7, g_5), Index(g_7, g_8)\}$;
$Index(g_7, g_5) = \min\{Index(g_7, g_5), Index(g_7, g_8)\}$;
So, initialize the path as $P_0 = \{g_5\}$;
Loop 1:
$S = \{g_7, g_5\}$, $(\min\{Index(g_7, g_5), Index(g_7, g_8)\} = 0.3)$;
$I = \{Index(g_7, g_8)\}$;
$AGS(g_5) = \{g_6\}$;
$C(g_7, g_6) = 800 > 600$, the cost exceeds the constraint;
Loop 2:
$S = \{g_7, g_5, g_8\}$, $(\min\{Index(g_7, g_8)\} = 0.36)$;
$I = \{\}$;
$AGS(g_8) = \{g_{10}\}$;
$C(g_7, g_{10}) = 750 > 600$, the cost exceeds the constraint;
Generate the path:
$P(l(g_{10})) = P(0) = g_{10}$;
So, the path becomes $P_1 = \{g_{10}\}$.

In this test case, it shows how the environment changes affected the goal achievement and how the Goal Net adapted to the changes dynamically. The original goal is g_8. The new goal will be generated towards the final goal achievement. In this test, the only new goal is the final goal which is g_{10}. However, $g_{10} \notin AGS(g_7)$ which means g_{10} is not achievable from g_7. This indicates that the Goal Net could not find a goal to pursue in the current situation. The agent will send a message containing the current values of the environment variables to the learner and recommend him to get help from his company. If the agent is not stopped, the agent will keep trying to find a next goal to pursue. To continue the test, we changed the price of the course *DBA II* back to S$500, the agent generated the path $\{g_8, g_{10}\}$, and sent a message to the learner to continue. Figure 6.8 lists the goal selection process.

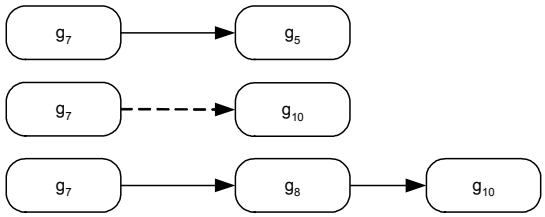

Figure 6.8 The goal selection process for Case 5

If we define a threshold value for the achievement (grade) of the learner, we can define the partial goal achievement. Similarly if we define a threshold value for the score of the course test using a fuzzy value, we can define the fuzzy goal achievement. For example, we define the goal is achieved if the learner gets grade 6. The threshold value is 0.5. Then for the above test cases, if the learner can finish the course following the optimal path, the learner can achieve 1, 0.67, 0.67, 1 and 0.67 of the goal respectively. Obviously, the achievement is greater than the threshold value, so the agent can proceed to pursue the next goal. Similarly if we define the threshold value for the score of the test as *satisfactory*, a fuzzification function can be defined to measure the fuzzy goal achievement.

6.5.2 The Experiments for the Learning Object Delivery Agent

As shown in Figure 6.4, the transition from the goal g_{31} to g_{32} consists of tasks to take individual learning objects that constitute the target course respectively. The selection mechanism is defined for the transition to select the tasks to take suitable learning objects based on the learner's skills. There are two cases for consideration here: 1) the learner has learned the learning object; 2) the learner has working experience related to the content of the learning object which means the learner has accumulated some skills with the content of the learning object through his working experience. For example, to learn PL/SQL of

Oracle, a learner must learn structured query language (SQL) first. A learner, who has not taken a course about SQL, can possess the skill about SQL if he has worked in a project using SQL for a certain period. So if he needs to learn PL/SQL, whether he needs to take the learning object about SQL is decided by how long he has used SQL for his work and the results of the pre-assessment. We assume that the longer he has used SQL, the higher the probability that he possesses the required skill. In this way, Bayesian networks can be used to decide whether a learner needs to learn the SQL learning object.

The factors with the affected learning object form a Bayesian network. So, in the transition, if the probability of having the skill that is related to the learning object is higher than a pre-set threshold, the task will not be selected. Otherwise, the learning object will be selected.

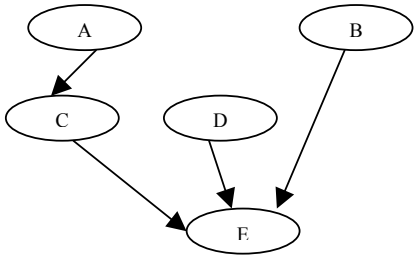

Figure 6.9 The Bayesian network for learning object

Figure 6.9 gives an example Bayesian network for the learning object. Table 6.6 lists the definitions of the nodes.

Node	Definition
A	used relational databases
B	learned SQL
C	used SQL
D	duration that SQL was used
E	learning object

Table 6.6 The node definitions of the Bayesian network

Table 6.7 lists a setting of the probabilities between the nodes. In Table 6.7, "y" indicates the value "yes" while "n" indicates "no".

Node	A=y	B=y	C=y	D=1 year	E=n
A=y	0.5				
B=y		0.5			
C=y	0.8		0.5		
D=1 year				0.9	
E=n		1	0.5	0.5	0.5

Table 6.7 The probabilities between the nodes of the Bayesian network

In this example, suppose the threshold for the probability not to select the learning object is 0.5, and a learner has working experience in database programming, if he has worked for more than one year, the probability is 0.8*0.5 + 0.5 = 0.9 > 0.5, which indicates that this learning object is not selected; if he has worked for less than one year, the probability is 0.8 * 0.5 = 0.4 < 0.5, which means that the learning object should be selected. In contrast, if the learner has learned the learning object, the learning object is not selected because the probability is 1 > 0.5.

The agent will try to deliver the learning objects in the order defined by the course provider. For a given learning object, the agent decides weather it is selected or not. If the learning object is selected, the agent reaches the state g_{32}. Otherwise, the agent reaches the state g_{33}. Once the agent selects the learning object, it will deliver the learning object to the learner for reaching the state g_{33}.

With the development of grid computing, many course servers are distributed in different locations. There are more attributes for each learning object in grid computing environment, such as, locations containing the learning object, available service time for the service from each locations, quality of service, reliability of service, busyness of the server, etc. The agents should get appropriate learning objects from one or more course

servers based on these attributes. However, the values of these attributes are changing dynamically. For example, one server may be busy during daytime in the local time while the other has better quality of service roughly between 9:00 am to 11:00am. So for the learning object delivery, another selection mechanism is defined to select the task that requests the learning object from the suitable course server to satisfy the learner's requirement. For example, Figure 6.10 (a) shows the learning objects in a grid environment. There are four nodes in the figure. One is the learning server on which a learning agent has registered to. Others are the course servers providing learning objects.

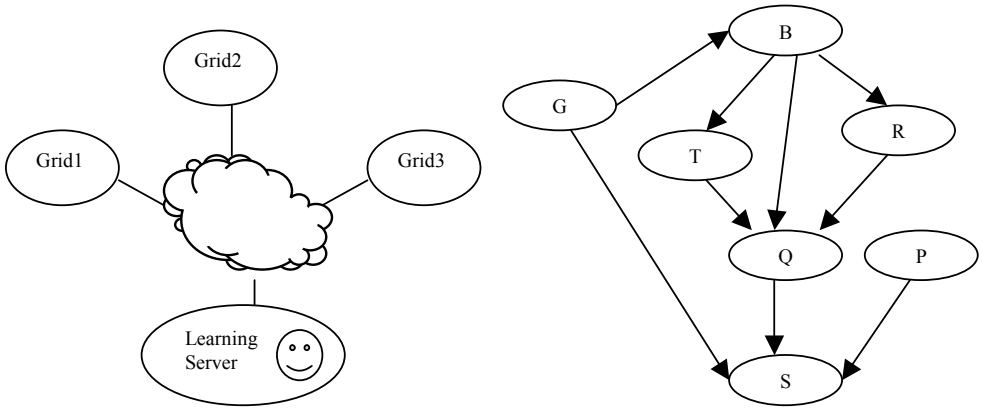

Figure 6.10 The example of course delivery

Figure 6.10 (b) shows the Bayesian network for task selection. Table 6.8 lists the definitions of the nodes.

Node	Definition
B	Busyness
G	Selected grid
P	Performance
Q	Quality of service
R	Reliability
S	Satisfaction
T	Time for service

Table 6.8 The node definitions of the Bayesian network

The Bayesian network shows that the selection of the grid affects the busyness of the grid which in turn affects the time of service, reliability and the quality of service; the selection of the grid, the quality of service and the performance of the service will finally affect the satisfaction of the learner. According to the action selection algorithm proposed in Chapter 3, the probability to obtain satisfaction of the service from the three grids can be computed respectively. The grid giving the highest probability of satisfaction will be selected by the learning assistant agent.

6.5.3 Conclusion from the Experiments

From the experiment for the learning path generation agent, it shows that the agent can 1) decide the next goal based on the environment changes, for example, course price changes, time constraint changes, etc.; 2) always have a solution for the next goal, it will have the optimal solution if computation time is allowed; 3) recommend the learner to take action if the agent could not find a solution to meet the learner's requirement at the current solution.

From the experiment for the learning object delivery agent, it shows that the agent can select the suitable task according to the current situation. It also shows how the agent handled the uncertainty with its current knowledge. In the experiments, it also shows that the goal *learn a course* can be reused. To learn a particular course can reuse the Goal Net by inheritance. The tasks defined in the transitions for connecting the course servers can also be reused. Because the tasks defined are similar. Only the course server address, course name, etc. are different which can be considered as parameters or environment variables. That is, a task defined for one transition can be reused for other transitions. This is a feature of Goal Net modeling method.

6.6 Summary

This chapter has explored the usage of the goal autonomous agents in an e-learning environment. We first analyzed the e-learning system using the goal modeling methodology. A goal model was built as the result of the analysis. The goal model was further refined by defining the tasks and environments between the goals. In particular, the complex problems with the learning path generation and learning objects delivery and delivery were discussed and solutions were proposed using the methods and methodology introduced in Chapter 3 to Chapter 5. Then we identified the agents from the goal model. Lastly, the multi-agent e-learning system architecture was given and the system was developed. Although the multi-agent e-learning system presented in this chapter is in its prototype system stage, the goal modeling methodology proposed in this research has been proven practical. The experiments show that the e-learning service agents are able to use flexible learning/reasoning mechanisms for providing learners personalized services that help learners to achieve their learning goals at anytime.

CHAPTER 7

AGENT-ORIENTED E-FORECASTING

Forecasting is vital to the success of a business in this knowledge-based economy. It also plays more and more important role to individual. There has been an increased need for providing next generation open e-forecasting services at anytime, from anywhere and in any form. However, the uncertain and complex nature makes it a challenging task to analyze design and implement open e-forecasting services.

Although agents and multi-agent systems have been applied successfully into many application domains, little work has been reported in the use of intelligent agent technology for e-forecasting purposes. This chapter explores why, where and how goal-oriented modeling and agent-oriented approach can be used to model and manage e-forecasting processes in the whole e-forecasting life cycle and in an open distributed environment. The main contribution of this chapter is that it presents an agent-oriented e-forecasting (AOEF) approach, which provides a generic solution for building open distributed e-forecasting (ODEF) services.

7.1 Why Agent-Oriented e-Forecasting?

Forecasting plays an important role in the economic world. It is concerned with the processes used to predict the unknown [Allen, 2000; Armstrong, 2000]. Moreover, nowadays, forecasting plays a more and more important role in people's every day life, such as stock forecasting, foreign currency forecasting, market forecasting etc. ***People want to access various e-forecasting services (stock, foreign currency etc.) from anywhere, at anytime and in any form (via cell phone, PDA, Laptops etc.).*** As a result, there has been an increased need for providing e-forecasting services in new environments, open distributed environment, pervasive environment etc. and in new forms. In fact, the widespread use of Internet based computing has brought a new perspective of software engineering to deliver software as open services instead of closed products in various application domains not limited to forecasting.

However, most of the current efforts on development of e-forecasting software systems are focused on implementation of specific forecasting methods for generating the forecasting result as a closed software product. While this is useful, it could not satisfy the new demands for providing e-forecasting services at anytime, from anywhere, and in any of the forms discussed above. Moreover, most of the current forecasting systems lack support for the whole forecasting life cycle. Using specific forecasting models for generating forecasting results is an important process in the forecasting life cycle, but there are some other important processes. The life cycle of e-forecasting covers processes of formulating a problem, obtaining information, selecting and implementing forecasting methods, evaluating methods, and using forecasts [Allen, 2000]. Each of the above processes may include a set of sub-processes. For instance, the "obtaining information" process has the following sub-processes, identifying data source, collecting data, and

preparing data etc. These sub-processes may contain their own sub processes. ***There is a great need for providing a software solution that can manage the processes throughout the whole life cycle of forecasting.***

Based on the above requirements, the new generation of e-forecasting systems need to have at least the following characters: 1) it is always running, i.e., it runs 24 hours a day and 7 days a week (24/7), users may request the e-forecasting at anytime 2) it lives in an open environment, users may request the e-forecasting services from anywhere 3) it allows new software components to be plugged in, new services to be composed "on the fly", for instance, new forecasting models need to be implemented and to be composed "on the fly" ; 4) it should be light-weight for residing in pervasive environment, for instance, users may want to access the services from mobile devices; 5) it needs to be personalized, as different users may access the services from different environments with different objectives. These characters lead us to consider a multi-agent system as a solution for managing the whole life cycle of e-forecasting processes and for providing the anywhere, anytime, and any form e-forecasting services.

Basically, as shown in Figure 7.1, forecasting covers processes of data discovery/collection, data preparation, forecasting method training and generating forecasting results etc. [Armstrong, 2000; Allen, 2000]:

- Data Discovery/collection: Data is essential to the forecasting process. The data should be collected as much as possible and should be from diverse sources, in the application domain.

- Data preparation: The collected data is real data in the application domain. It can't be used directly. It should be normalized. The relationships or associations between data should be also identified.

- Forecasting Model Training: The forecasting model needs to be trained using sample data in order to make the forecasting result as accurate as possible.

- Forecasting: The e-forecasting method is implemented in this process. It will use the trained forecasting model to generate a forecasting result.

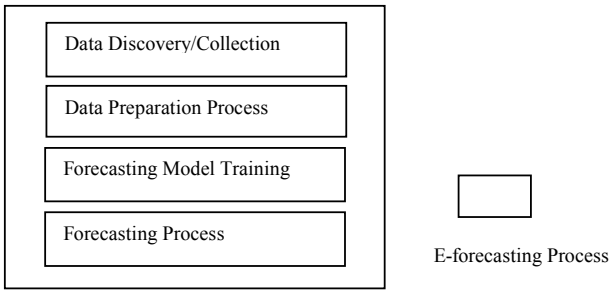

Figure 7.1 The e-forecasting processes

Traditionally, human beings have to be involved in each process and use different tools or manual work for managing each forecasting process in the life cycle as shown in Figure 7.2.

In fact, agents can be used for managing many processes in the e-forecasting life cycle, for instances:

- Agents Can Help to Identify Data Sources

  Nowadays, the Internet has evolved from an information space to a market space with millions of electronic storefronts, auctions and stock markets etc. Mobile agents can travel from one place to another to help people to identify data sources for e-forecasting autonomously.

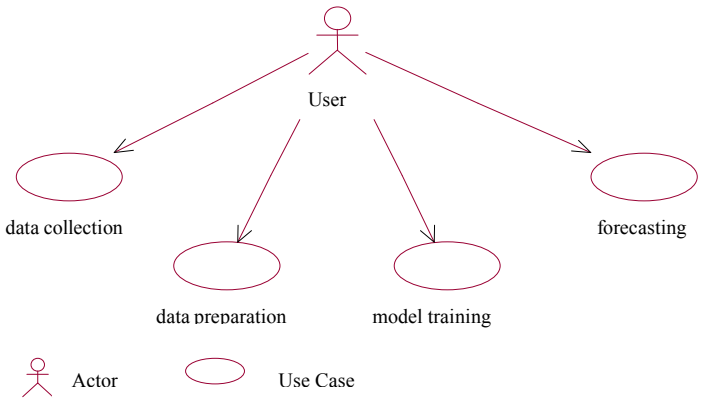

Figure 7.2 The traditional e-forecasting use cases

- Agents Can Help To Collect Most Recent Data

Data is important to the forecasting result and accuracy. Data are collected by human beings manually in traditional e-forecasting system. And they are not most recent data. The data collection agent can frequently visit the data resources to collect data and send the collected data back. They can work 24 hours a day, 7 days a week to get most up-to-date data from data sources on behalf of human beings.

- Agents Can Help to Prepare Data

The goal of the data preparation agent is to convert the original data to consistent data with the forecasting model and find out the relationship between the variables to generate scenarios. It keeps running continuously. Once it detects that new data is being sent back by the data collection agent, it will use its knowledge to prepare the data for other agents.

- Agents Can Help to Do Forecasting

The forecasting agent can be implemented based on the forecasting method or forecasting model. It trains the forecasting model and does forecasting computational reasoning to generate forecasting result. The forecasting agent is always running autonomously. Its goal is to generate forecasting result. Once it gets a task, it will give the forecasting result. If the result is not satisfactory, it will update its knowledge with its learning/training ability.

Knowing where agents can be used in the e-forecasting life cycle, we propose an agent-oriented e-forecasting approach to model, design e-forecasting processes by proposed goal-oriented modeling method and construct e-forecasting systems as a multi-agent system. Each agent inside the MAS manages some processes (sub-goals) in the e-forecasting life cycle, and the entirety of the MAS provides a complete software solution for assisting people to do e-forecasting (overall goal) in an open distributed environment.

In such a background, we explore how agents can be used to help people to do e-forecasting. An agent-oriented e-forecasting (AOEF) approach is presented based on the proposed GO methodology and multi-agent development framework.

7.2 Goal-Oriented Modeling for Open e-Forecasting Systems

7.2.1 Problem Description

Following the discussion from Section 7.1, we aim to model, design and implement a multi-agent system for providing open e-forecasting services. We assume that new users from anywhere may request the services at any time. New data sources may be

discovered, and new forecasting models may be added in "on the fly". The total number of user agents is unknown in deployment time. The total number of data sources and the number of data collection agents are unknown and so are the total number of training agents and forecasting agents.

In this case study, we limit the business forecasting models that will be implemented for generating forecast results to neural network based. This is due to the fact that neural network based forecasting models have become one of the most popular forecasting models for various forecasting purpose in different application domains.

Neural networks [Zurada, 92] have been used in a wide range of practical applications, such as classification, pattern recognition and e-forecasting etc. [Lin, 92; Azoff, 94]. Forecasting is done based on forecasting models [Armstrong, 2000]. Today's business environment is both complex and changing very fast. The relationships between the forecast variables and the input factors cannot always be expressed by a mathematical model. Compared with traditional forecasting methods, forecasting models based on neural networks provide a superior fit for the above new requirements. Neural networks provide a way to model the complex systems with a large number of input factors [Kosko, 92]. They are able to learn from previous experience and even acquire new knowledge by self-organised training [Rumelhart, 86]. Trained neural networks are faster than most of the available statistical forecasting techniques with at least the same degree of accuracy [Azoff, 94].

Forecasting is not an easy task and therefore has attracted many researchers to explore it. Many researchers have suggested that neural networks are suitable and serve as novel tools in e-forecasting. However it is very tedious to learn and train a neural network. The

automation of neural network based forecasting through agents is a challenging task that will lead to a new horizon for providing open e-forecasting services. The advantages include not only freeing human beings from the tedious forecasting procedures, but also improving the accuracy of the forecasting model by training the model using the most recent data or even up to moment data via autonomous cooperation of data collection agents and forecasting model training agent.

In such a background, there is a need for research efforts on proposing a generic software solution which enables software agents to represent various neural-network based e-forecasting models as their own knowledge and to carry out the learning, training and reasoning for providing e-forecasting services in various application domains.

7.2.2 Goal-oriented Requirement Analysis

The goal-oriented requirement analysis will be carried out according to the proposed GO methodology descried in Chapter 5. We start from a knowledge based top-down approach for identifying the goals. The ***objective or overall goal*** is to provide open e-forecasting services that can assist various e-forecasting processes in the whole e-Forecasting Life Cycle.

By asking HOW to achieve this objective, the overall goal of the open e-forecasting services can be broken into a set of e-forecasting processes (sub-goals), such as user interaction, implementing the forecasting model, training the model, collecting data, and normalizing the data according to the requirements of the forecasting model, and finally generating the forecasting result using the trained forecasting model. Therefore, the sub-

goals can be specified as “user interaction”, “model implementation”, “data collection”, “data preparation”, “model training”, and “forecasting”, etc.

Each sub-goal can be represented by a composite state which demonstrates the goal pursuing process. For a real world forecasting problem, the decomposition process could be very complex. For example, each sub-goal can be further decomposed by asking HOW to achieve it. This decomposition process will be repeated until all the leaf sub-goals are atomic goals.

Following we show a goal identifying procedure through a goal decomposition process.

- **Step 1**: The overall goal is decomposed into six sub-goals (by asking HOW) which include user interaction, model implementation, data collection, data preparation, model training, and forecasting generation.
- **Step 2**: We take one of the sub-goals, for example, forecasting generation, to illustrate the further decomposition. By asking HOW to generate the forecasting results, it can be further decomposed into a set of goal pursuing processes: model training and forecast computation. The forecasting model is not necessary to be trained every time. So a rule-based action selection should be defined to decide whether the forecasting model needs to be trained. If the model needs to be trained the forecasting agent should send request to the training agent to train the model. The process will be repeated for each sub-goal identified by step 1.
- **Step 3**: The process from step 1 to step 2 is a recursive process which means that if a new sub-goal is identified in the step 2, the process step 1 and step 2 will be repeated until all the leaf sub-goals are atomic goals.

Figure 7.3 depicts the goals identified. Compared with the figure in Chapter 5, two sub goals are added to cater for the needs of e-service and openness.

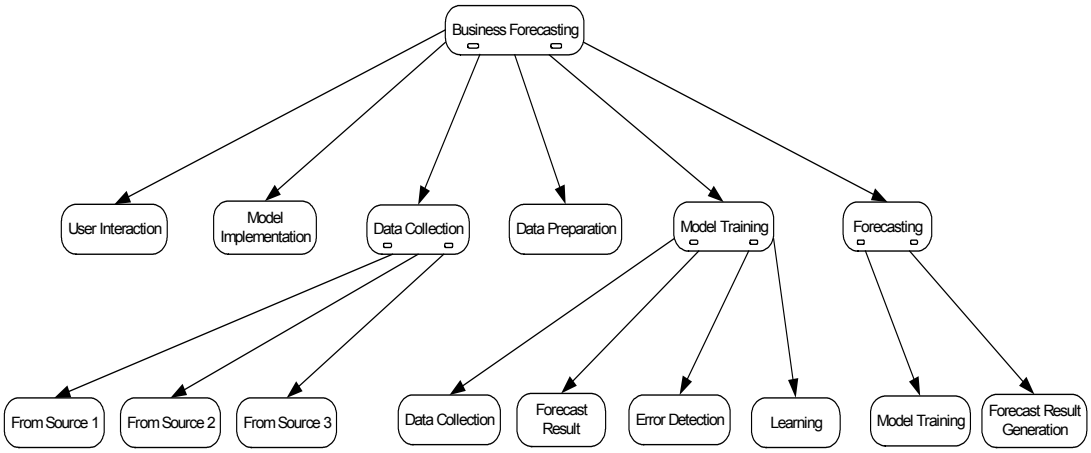

Figure 7.3 The identified goals for an ODEF system

After identifying the goals, for each goal we identify the transitions between the goals, and the environment for goal pursuing. The output of the goal-oriented analysis is a preliminary Goal Net that illustrating a goal hierarchy. A set of GET cards can be constructed for the goals in the hierarchy. For example, Figure 7.4 shows a GET card for the goal *data collection*. Figure 7.5 shows the Goal Net. In this case study, we focus on the sub-goals model implementation, data collection, model training, and forecasting. The decomposition of the sub-goal model implementation is not shown in this figure. It will be discussed in details in the following sections.

Goal: Collect data from data sources	
Environment Variables	**Tasks**
Data status on Source 1	Collect data from source 1
Data status on Source 2	Collect data from source 2
Data status on Source 3	Collect data from source 3

Figure 7.4 The GET card for the goal *data collection*

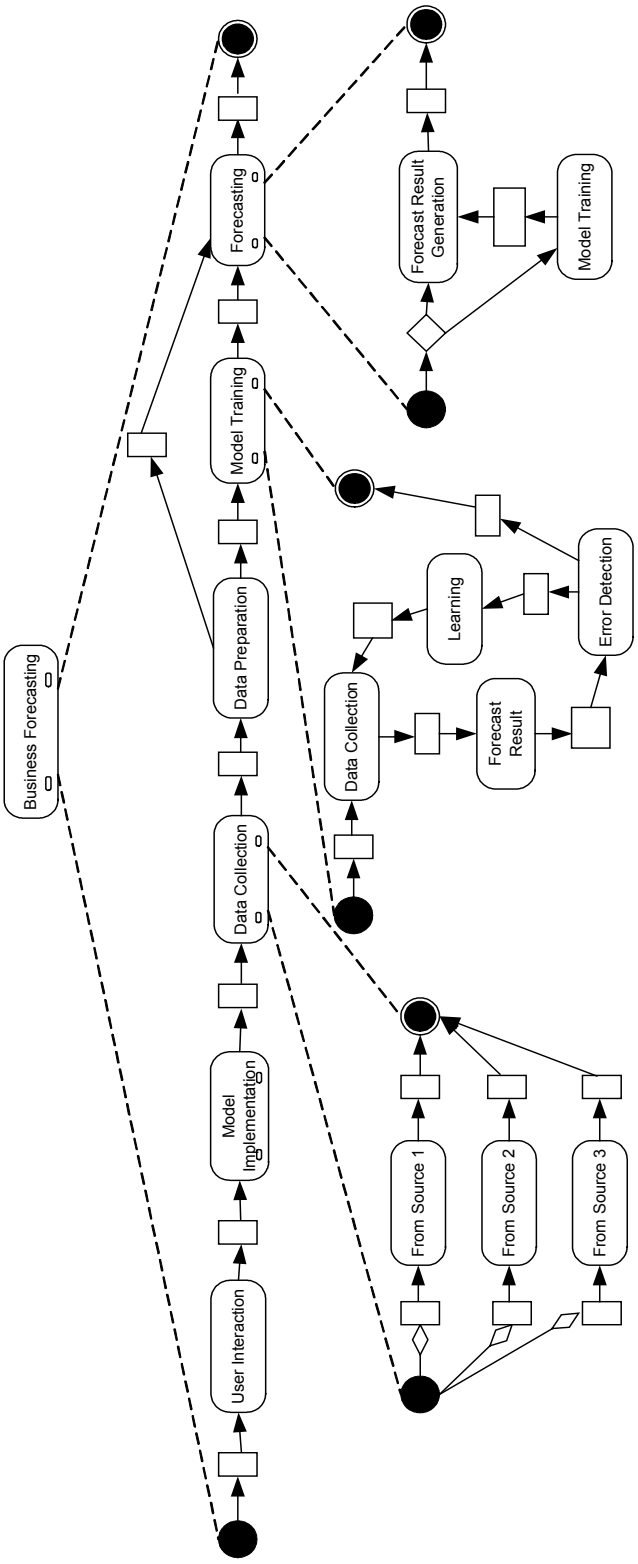

Figure 7.5 The Goal Net for an ODEF system

7.2.3 Agent Identification and Multi-agent System Organization

Agent identification for the forecasting agents can be done based on Goal Net. We adopted role based identification method to identify agents because the functionalities of different roles are easily isolated for this system.

With the identified goal hierarchy as shown in Figure 7.5, a multi-agent system can be identified and organized based on the possible roles. Figure 7.6 shows the identified agents and the agent hierarchy.

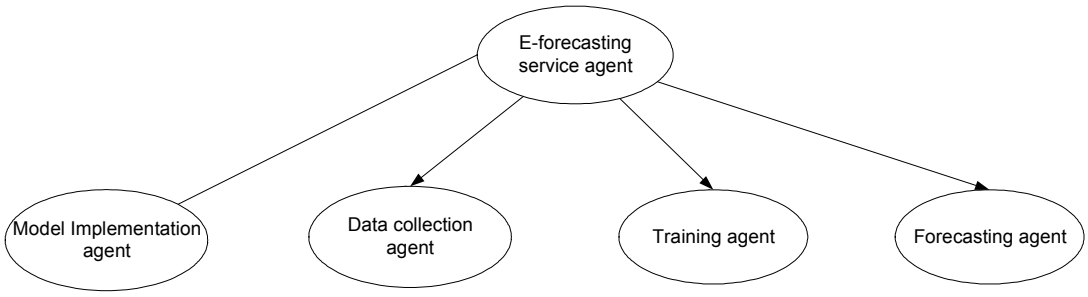

Figure 7.6 The agent hierarchy of e-forecasting agents

As shown in the figure, there are five roles agents can play in the business forecasting life cycle: e-forecasting service agent, model implementation agent, data collection agent, forecasting agent, and training agent. The e-forecasting service agent is the coordinator of the data collection agent, forecasting agent and the training agent according to Goal Net. If the e-forecasting service agent needs to process a forecast request, it will inform the data collection agent to collect data. After the data is collected, the e-forecasting service agent will normalize the data and use the data to compute the forecasting results by the forecasting agent. If the forecasting agent realizes the forecasting model needs to be re-trained before the forecasting results are computed, it will inform the training agent to

train the forecasting model using the historical data and update the forecasting model in the knowledge base. After that, the training agent will inform the forecasting agent to update its knowledge and proceed to compute the forecasting results.

The openness of such a system is evidenced in a few ways. New users may request the services at any time. New data sources may be discovered, and new forecasting models may be added in. The total number of user agents is unknown at the deployment time. The total number of data sources and the number of data collection agents are unknown and so are the total number of training agents and forecasting agents. So an agent generator is necessary to implement the forecasting model and create the forecasting agent on demand. In the figure, the model implementation agent will be the agent generator agent, which can create or stop an agent based on the request from the e-forecasting service agent. If the e-forecasting service agent needs to process a forecasting request, it will select a suitable forecasting model and request the agent generator to create a forecasting agent.

Agents in the e-forecasting MAS live in an open distributed environment. They are de-centralized; each of them has its own environment, and pursues its own goal. Meanwhile, they work together towards a common goal for providing high efficient e-forecasting services.

Unlike traditional e-forecasting style shown in Figure 7.2, Figure 7.7 shows the agent-oriented e-forecasting paradigm. Agents act as dynamic processes of data collection, model training and forecasting and corporate with each other during the whole e-forecasting lifecycle. The e-forecasting service agent is responsible for interaction with users. Users can interfere and take appropriate action during the e-forecasting life cycle

through the e-forecasting service agent. The forecasting agent generator is also an agent, which generates the forecasting agents dynamically.

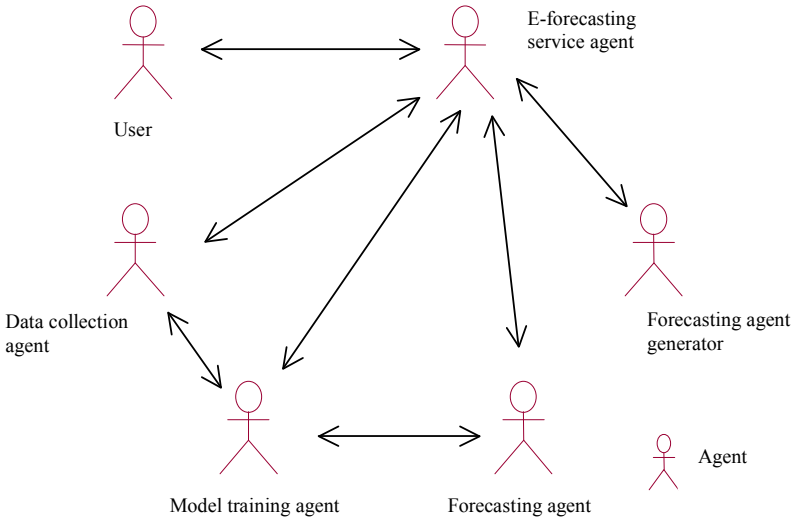

Figure 7.7 The agent-oriented e-forecasting

Formally, the multi-agent e-forecasting system can be defined as a tuple EFMAS = (A, C, O, K, E), where

- A = $\{agent_i \mid i = 1, 2, \ldots n\}$ is a set of agents for managing business forecasting processes;
- C is a set of communication channels;
- O is an ontology server;
- K is a knowledge base;
- E is the agent environment.

The constructed EFMAS will be developed using MADE described in Chapter 4. The Goal Nets for each identified agents as well as the forecasting models that the agent will

use to generate forecasting results are stored in the knowledge base. In the following sections, we demonstrate how the proposed agent-oriented e-forecasting system can be implemented based on MADE described in Chapter 4. Particularly, we focus on the forecasting agent generator to demonstrate how the agent is designed and implemented using the methodology proposed in Chapter 5 and the agent framework introduced in Chapter 4 and how the proposed agent model incorporates domain knowledge as the agent knowledge. More specifically, to construct a forecasting agent, the agent must represent the forecasting model as its own knowledge in order to generate the forecast result for its users. An agent knowledge model is proposed for representing various neural network-based forecasting models as its own knowledge.

7.3 Design of a Forecasting Agent Generator

In the above multi-agent e-forecasting system, we take one of the agents, business forecasting agent generator, as an example, to show how to develop an agent using the proposed MADE (Multi-agent Development Environment).

In Chapter 4, we have defined an agent model for goal autonomous agents logically and structurally. An agent development environment, MADE, including an agent development framework is also described.

With the MADE, using Goal Net Designer, the Goal Nets will be developed and stored in the knowledge base. Using the Agent Creator, a dummy goal autonomous agent will be created. When the agent starts to run, through the Knowledge Loader, the Goal Net stored in the knowledge base will be loaded into Process Unit, and the knowledge model (that is, the forecasting model) stored in the knowledge base will be loaded into the Knowledge

Unit. After Goal Net and Knowledge model are loaded, the dummy agent becomes a goal-oriented intelligent agent.

7.3.1 The Goal Net of Business Forecasting Agent Generator

Because of the dynamic demands, the number of agents for e-forecasting is unknown, an agent generator agent is required. A forecasting agent can be created dynamically when a forecasting request comes. An agent can be stopped also based on the request. To achieve this, the agent must be able to sense the knowledge base and wait for the request. After a new forecasting model is added into the knowledge base or a request is raised, the agent can create a dummy agent, an intelligent agent or stop an agent based on the request. After a dummy agent is created, a Goal Net should be assigned and then the created agent should be started whereas after an intelligent agent is created, apart from assigning a Goal Net, a forecasting model or other domain knowledge should be bound to the agent before the created agent is started. If the request is to stop an agent, the agent generator agent should be able to stop the specific agent. Based on the above analysis, we can have the Goal Net as shown in Figure 7.8.

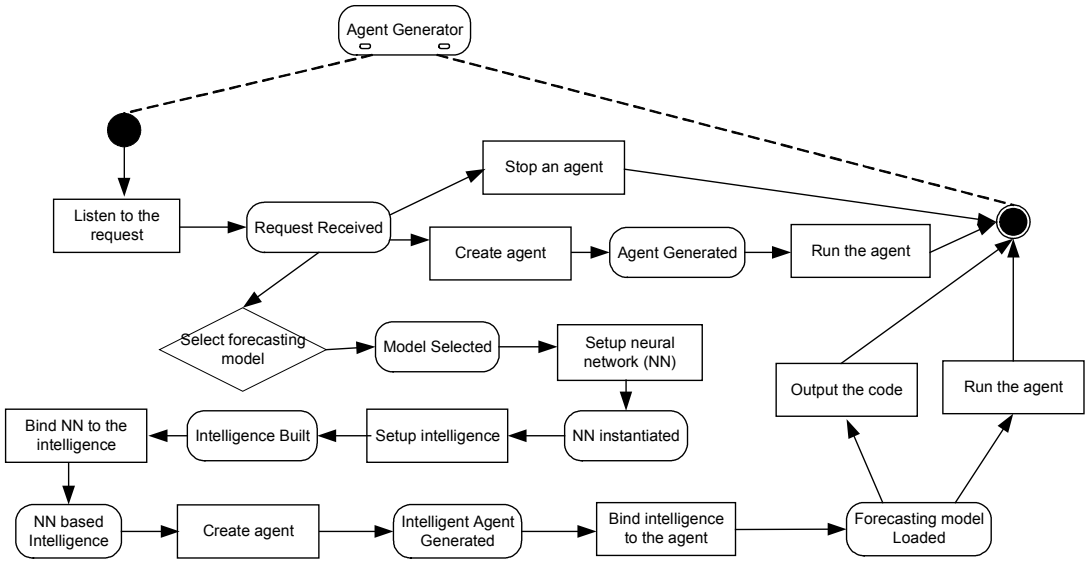

Figure 7.8 The Goal Net of the forecasting agent generator

To construct the forecasting agent, we must model a forecasting model as an agent knowledge model.

7.3.2 An Agent Knowledge Model

The agent knowledge model is represented as a layered and weighted directed graph as the neural network model, in which nodes denote *neurons* and weighted arrows denote *synapses*.

The layered structure of the model starts with an *input* layer, where each node corresponds to a predictor/input variable. The nodes in the input layer are connected to a number of nodes in a *hidden* layer. Each node in the input layer sends impulses through weighted arrows to every node in the hidden layer. The nodes in the hidden layer may be connected to nodes in another hidden layer, until the *output* layer is reached. The output

layer consists of a number of output nodes corresponds to response/output variables (i.e. the variables to be forecasted).

Figure 7.9 gives an example of the layered structure of the agent knowledge model.

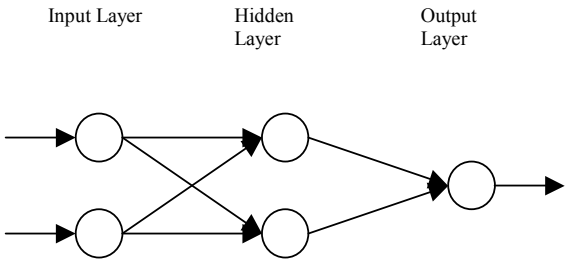

Figure 7.9 An agent knowledge model

Figure 7.10 shows a node *j* in layer *k* and the arrows connected to it. Every arrow has a weight to indicate how strong the impulse is. The impulse to a node from previous layer is called an input of the node and the impulse from a node to the next layer is called an output of the node. Typically a node has inputs from every node in the previous layer and outputs to every node in the next layer.

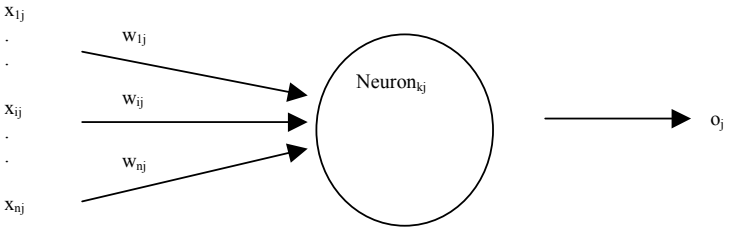

Figure 7.10 A node of the agent knowledge model

In Figure 7.10, subscript *i* represents the node number in the previous layer; x_{ij} represents the input from node *i* of previous layer to the node *j* of current layer; w_{ij} represents the weight of the input x_{ij}; and o_j represents an output of the current node. Since the outputs of a neuron have the same values, it can be considered that each node has only one output.

There are two functions associated with each node: *sum* function and *activation* function. The sum function adds all the input impulses to generate the net value of the node. The activation function generates output impulses based on the net value of the node.

The node performs a designated transformation on its inputs (x_{ij}) weighted by their respective weight values (w_{ij}) to generate an output (o_j). This is accomplished in two steps. The first step transforms the inputs into the net value net_j by the sum function:

$$net_j = F(x_{1j} \cdot w_{1j}, ..., x_{ij} \cdot w_{ij}, ..., x_{nj} \cdot w_{nj}) \tag{7.1}$$

where function *F()* is the sum function; subscript *i* represents the node in the previous layer that sends input to node *j* in this layer and subscript *n* represents the total number of inputs.

The second step generates the output o_j through the activation function *f()*:

$$o_j = f(net_j) \tag{7.2}$$

The output, when weighted by some weight values, will in turn become inputs to the relevant nodes in the immediate next layer according to the node connections in the network structure.

When all the layers are traversed, i.e., the output layer is reached, the outputs of the nodes in the output layer produce the forecasting results.

7.3.3 Knowledge Base

The knowledge base is designed to store the agent knowledge models. Figure 7.11 shows the entity relationship diagram of the knowledge base. The entity *Network*, *Node* and *Arrow* store the structure data of knowledge models. The entity *Parameter* stores the parameters of the knowledge models, such as learning rates, error limits and iteration numbers etc. The entity *Status* stores the state definitions of the knowledge models.

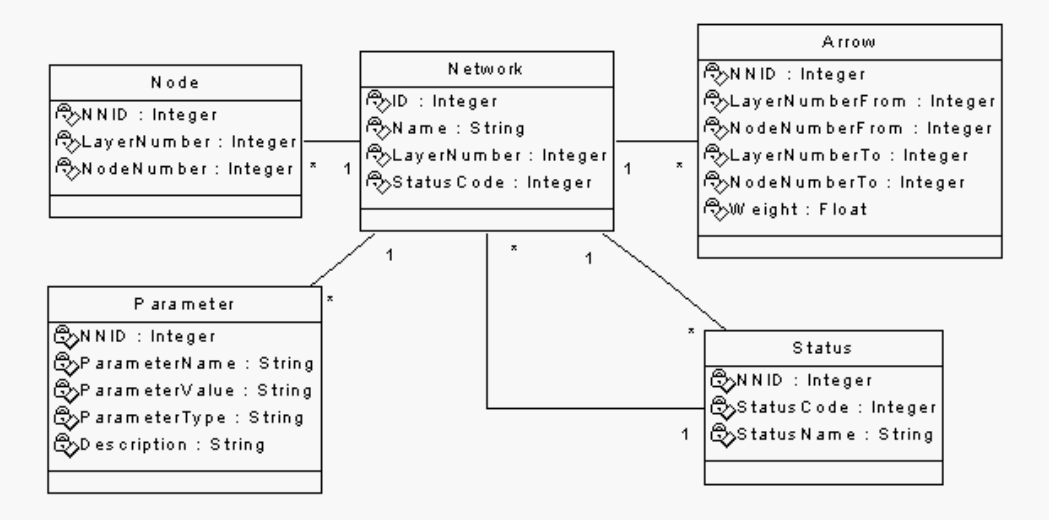

Figure 7.11 The structure of knowledge base

Knowledge base is important for knowledge reuse. It enables the agent to restore the knowledge from the knowledge base and starts service immediately.

7.3.4 Agent Knowledge Representation

To represent various neural network based forecasting models as agent knowledge models, the following classes have been designed.

The class *NeuralNetwork* defines the structure of the model. It has aggregation relationships with class *Neuron* and class *Synapse*. It has the properties that describe the structure of neural network such as number of layers, the number of nodes in each layer, the nodes and the weights between the nodes. The property *state* indicates the status of the networks. The Property *node* is a two-dimension array of *Neuron*, where the two dimensions indicate layer number and position number of the node in the layer. For example, *node[3][5]* indicates that the node is in the third layer counted from the left and it is the fifth node counted from the top in the same layer. The Property *arrow* is a four-dimensional array of *Synapse*. The first two dimensions indicate the node from the previous layer while the last two dimensions indicate the effected node. For example, *arrow[3][5][4][2]* indicates the arrow from *node[3][5]* to *node[4][2]*. The function *initNeuralNetwork()* initializes the network. Function *setupNode()* and function *setupArrow()* assign each node object and arrow object to node array and arrow array respectively.

The class *Neuron* defines node in the model. It contains the property *layerNumber* to indicate the layer number the node belongs to, the property *nodeNumber* to indicate the position of the node within the layer, and the property *inputNumber* to indicate the number of inputs to this node, etc. The function *sumFunc()* and *activation()* define the sum function and activation function respectively. Both of them are abstract functions. They need to be implemented by specific algorithms of neural networks based forecasting models during agent construction.

Similarly, the class *Synapse* represents the weighted arrow in the model. It contains the layer number *fromLayer* and the node number *fromNode* of the node that the arrow starts from and the layer number *toLayer* and the node number *toNode* of the node that the arrow ends at. The weight value *weight* indicates how strong the output impulse is from one node to another. The class diagram of the model is shown in Figure 7.12.

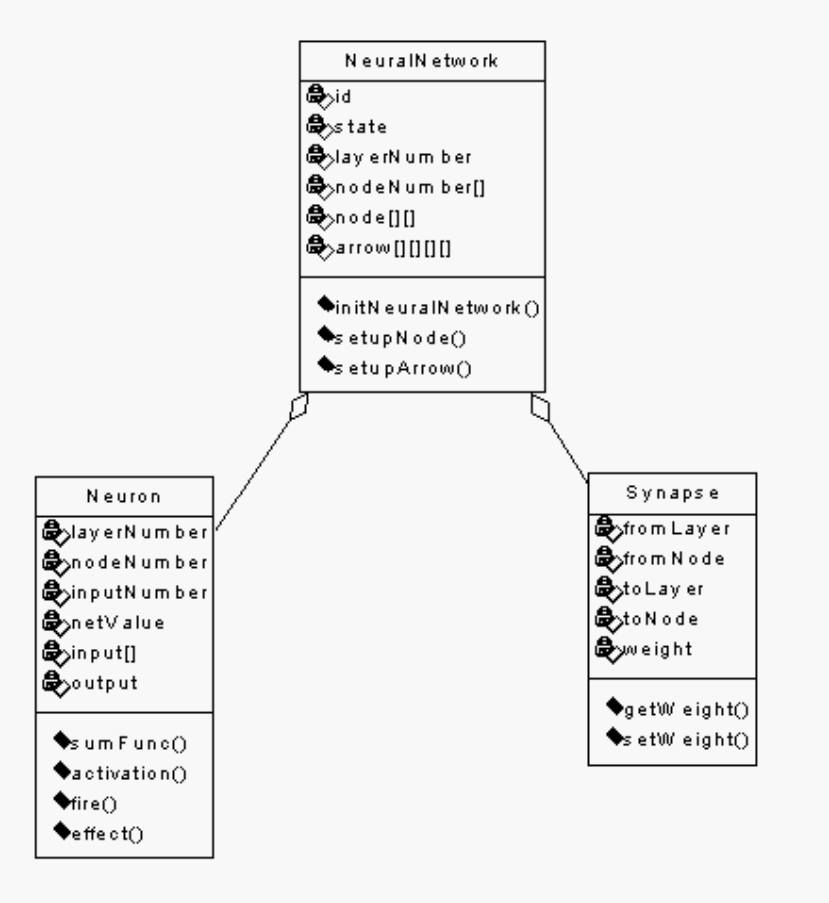

Figure 7.12 The class diagram of the agent knowledge model

To enable the classes defined in Figure 7.12 for representing neural network based forecasting model as agent knowledge model, the defined classes can be bound to the knowledge unit of the agent framework. As shown in Figure 7.13, class *intelligence* is an implementation of the class *KnowledgeUnit*. It contains a neural network object (an

instance of the class *NeuralNetwork*), which represents the agent knowledge model, i.e. the knowledge of an agent. The class *intelligence* also defines the functions, which present the intelligent behaviors of agents such as learning, reasoning, training and action planning, using the agent knowledge model.

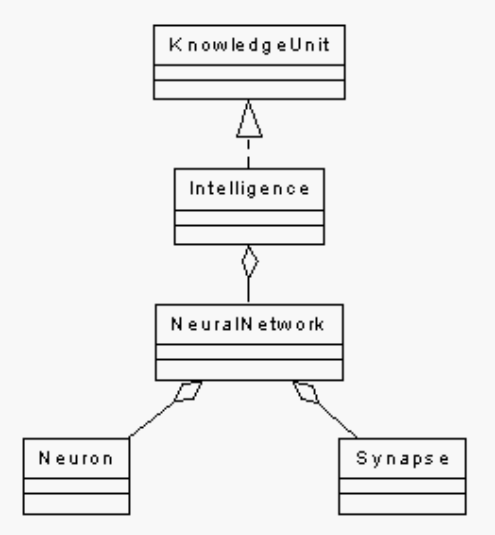

Figure 7.13 An implementation of knowledge unit

- Reasoning

Within the agent knowledge model, reasoning is to reason about what value of the output variable will be, based upon the values of input variables. The reasoning is carried out by the process starts from receiving the input impulses by the nodes in the input layer, translating and sending those impulses along arrows into nodes in the hidden layers, and ends after generating the outputs (i.e. reasoning results) of the nodes in the output layer.

Unlike symbolic logic based reasoning such as first order logic, neural network uses pure mathematics computation to infer the result instead of using logic induction. The function *reasoning()* takes all the inputs to infer the output using the agent knowledge,

i.e. the agent knowledge model. It traverses all nodes layer by layer to compute the final output of the neural network by the sum function and activation function. Figure 7.14 gives an implementation of reasoning mechanism.

```
public class intelligence implements knowledgeUnit {
  neuralNetwork nn;
  int state;
  int result;
  public void reasoning() {
    int i, j, k;
    int output=0;
    for (i=0;i<nn.layerNumber;i++)
      for (j=0;j<nn.nodeNumber[i];j++) {
        state = nn.node[i][j].sumFunc();
        output = nn.node[i][j].activation();
        if (i == nn.layerNumber - 1) continue;
        for (k=0;k<nn.nodeNumber[i+1];k++) {
          nn.node[i+1][k].effect(j, output*nn.arrow[i][j][i+1][k].getWeight());
        }
      }
    result = output;
  }
```

Figure 7.14 A reasoning function

- Learning

In order to generate the accurate forecasting result by the reasoning process, learning and training are fundamental to the model. Each neuron's output is determined by two things, the input and the weight of synapse that indicates how strong the input impulse is. Therefore, the learning mainly focuses on making changes to weights in order to improve the accuracy of the output result. The function *learning()* implements the learning algorithms to minimize the error of reasoning. Unlike reasoning, learning algorithms vary in different e-forecasting models. These are the three commonly used learning modes: supervised learning, reinforcement learning and unsupervised learning.

In the supervised learning mode, desired output value is given. The error between the desired output value and computed output value is used to modify the weights between the nodes. Reinforcement learning is similar to supervised learning except that the exact desired output value is not provided. Instead, only a "grade" is given to indicate how well the neural network is doing,. In contrast to the both, there is no information or feedback at all about its performance level in the unsupervised learning mode. The neural network is only presented with a series of input data. This learning mode is also called self-organization.

There is no restriction for the learning mode in the agent knowledge model. The function *learning()* can be implemented during the agent construction according to different learning algorithms.

- Training

Training is different from learning. Training is the procedure through which the network learns. It is external to the neural network. Function *training()* enables agent to learn from training data. It empowers the agent to have the self-learning ability. For every set of training data, it invokes function *learning()* internally to adjust the weights.

The design of the forecasting agent generator demonstrates a re-usable way for creating a forecasting agent based on a given forecasting model. The next section describes a specific intelligent e-forecasting model based upon fuzzy neural network and the knowledge representation of an intelligent e-forecasting agent using the proposed agent knowledge model. It is shown that the proposed agent knowledge mode is able to

represent and transform the forecasting model as the knowledge of the intelligent e-forecasting agent.

7.4 Implementing of a Specific Forecasting Agent Generator

In this section, we follow the design of the Forecasting Agent Generator to implement a specific Forecasting Agent Generator based on an intelligent forecasting model represented by a fuzzy neural network.

7.4.1 An Intelligent Forecasting Model

The intelligent forecasting model is proposed based upon fuzzy neural network (FNN) which integrates the basic elements and functions of a traditional fuzzy logic inference into a neural network structure [Kosko, 92]. It is a five-layered fuzzy rule-based neural network [Li, 99]. The structure is shown in Figure 7.15.

Layer 1: The nodes in this layer indicate the input variables. They transmit input values to the nodes in layer 2 directly. Thus for node i ($i = 1, 2, \ldots, p$) in layer 1, we define:

$$\text{net}_i = x_i, \quad o_i = f(net_i) = net_i \tag{7.3}$$

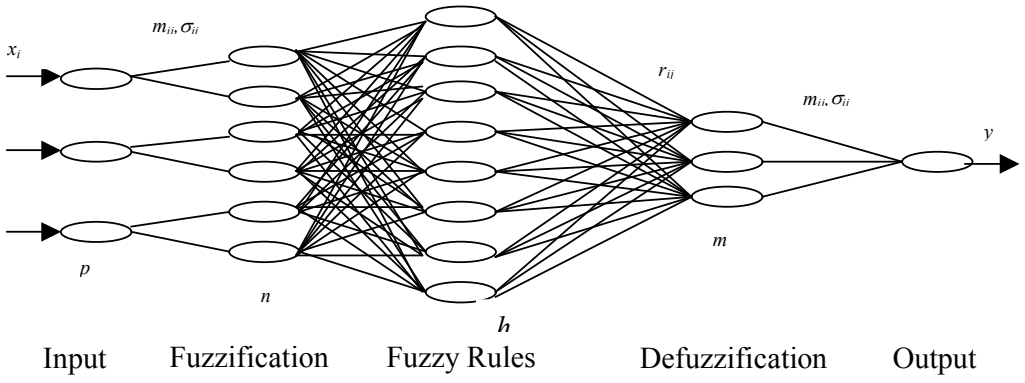

Figure 7.15 The forecasting model

Layer 2: Each node in layer 2 is a fuzzifier of an input variable in layer 1. It transforms a numerical input into a fuzzy set [Zadeh, 65]. Here we define the membership functions are normal distributions with a range of {0, 1}. For the node j ($j = 1, 2, \ldots, n$) in layer 2, we define:

$$net_j = -(\frac{x_{ij} - m_{ij}}{\sigma_{ij}})^2; \tag{7.4}$$

$$\text{and} \quad o_j = f(net_j) = e^{net_j} \tag{7.5}$$

In this expression, x_{ij} is the input from node i ($i = 1, 2, ..., p$) in layer 1 and m_{ij} and σ_{ij} are the mean and variance of the membership function of node j respectively.

Layer 3: Each node in this layer performs a fuzzy AND operation on its inputs. Thus we have:

$$net_j = min\{x_{ij} \cdot w_{ij}\} \tag{7.6}$$

$$\text{and} \quad o_j = f(net_j) = net_j \tag{7.7}$$

where $i = 1, 2, ..., n;\ j = 1, 2, ..., h;$ and weight w_{ij} is unity.

Layer 4: Each node in this layer performs a fuzzy OR operation on its inputs. Thus we have:

$$x_{cj} = max\{f(net_1),\ f(net_2),...,\ f(net_h)\} \tag{7.8}$$

$$net_j = x_{cj} \cdot r_{ij} \tag{7.9}$$

$$\text{and } o_j = f(net_j) = net_j \tag{7.10}$$

where $i = 1, 2,\ldots, h;\ j = 1, 2, ..., m;$ and $c \in \{1, 2, \ldots, h\}$. Node c in layer 3 is the "winner" node of the fuzzy min-max operation. The r_{ij} is the rule value. The rule values are either initialized with random values or assigned directly by domain experts. They are then fine-tuned in the ODEF by supervised learning.

Layer 5: The node in this layer indicates the output variable. It performs defuzzification to generate outputs. Here the Center Of Gravity method [Kosko, 92] is used, which utilizes the centroid of the membership function as the representative value. Thus if m_j and σ_j are the mean and the variance of the output membership function respectively, we have:

$$w_j = \sigma_j \cdot m_j \tag{7.11}$$

$$net = \sum_{j=1}^{m} w_j \cdot x_j = \sum_{j=1}^{m} \sigma_j m_j x_j \tag{7.12}$$

$$f(net\ ) = \frac{net}{\sum_{j=1}^{m} \sigma_j x_j} \tag{7.13}$$

where $j = 1, 2, ..., m$; x_j is the input from node j in layer 4 and w_j is its weight. Then the defuzzified output $\hat{y}(t)$ is given:

$$\hat{y}(t) = f(net) \tag{7.14}$$

There are three steps to set up the neural network: self-organized learning, identification of the fuzzy rules and supervised learning.

The self-organised learning process implements the Kohonen's Feature Maps algorithm [Kohonen, 98] to find the number of membership functions, their respective means and variances.

For each input variable and output variable, we initialize the mean values $m_1, m_2, \ldots, m_k$ based on the training data set X = $(x_1, x_2, \ldots\ldots, x_n)$, where k, the total number of membership function nodes, is decided by the domain experts; and

$$min\ (x_1, x_2, \ldots, x_n) < m_i < max\ (x_1, x_2, \ldots, x_n)$$

The data x_j $(j = 1, 2, ..., n)$ is then grouped around the initial mean m_c $(1 \le c \le k)$ according to:

$$| x_j - m_c | = \min_i \{ | x_j - m_i | \} \tag{7.15}$$

where $i = 1, 2, ..., n$.

The data groupings and the initial mean values are optimized by the following iterative process:

For each mean value m_c ($1 \le c \le k$), if data x_j ($j = 1, 2, \ldots, n$) belongs to the grouping of m_c, we have:

$$m_c(t+1) = m_c(t) + \alpha(t)[x_j(t) - m_c(t)] \tag{7.16}$$

where $x_j(t)$ and $m_c(t)$ are the value of x_j and the value of m_c at iteration t ($t = 0,1,2,\ldots$) respectively; $\alpha(t)$ ($0<\alpha(t)<1$) is a monotonically decreasing scalar learning rate.

Otherwise, if x_j does not belong to the grouping of m_c, we have:

$$m_c(t+1) = m_c(t) \tag{7.17}$$

The iteration stops when the condition $|m_c(t+1)-m_c(t)| \le \delta$ is satisfied, where δ is an error limit.

After the mean m_i ($1 \le i \le k$) is optimized, the variance σ_i of membership function i can be computed by following equation:

$$\sigma_i = \frac{1}{R}\sqrt{\frac{1}{p_i}\sum_{j=1}^{p_i}(x_j - m_i)^2} \tag{7.18}$$

where p_i is the total number of data samples in data grouping of m_i; x_j is the data sample and R is the overlap parameter.

After the membership functions are constructed, the total number h of initial rules in layer 3 is decided by

$$h = \prod_{i=1}^{p} k_i \tag{7.19}$$

where p is the total number of input variables in layer 1 and k_i is the number of membership functions of the i'th input variable. Thus a fully connected neural network structure is completed. The next process is to identify the fuzzy rules using the same sets of data samples. Based upon a fuzzy AND operation in layer 3, then followed by a fuzzy OR operation in layer 4, the winner node associated with this particular set of input data is identified by the fuzzy min-max operation.

Finally the rule values are assigned and fine-tuned by supervised learning process to minimize the output error. We define the output error E by the following equation:

$$E = \frac{1}{2}[y(t) - \hat{y}(t)]^2 \tag{7.20}$$

where $y(t)$ is the actual value of output variable and $\hat{y}(t)$ is the computed output.

From the back propagation algorithm of Rumelhart [Rumelhart, 86], we define:

$$r_{ij}(t+1) = r_{ij}(t) - \eta \frac{\partial E}{\partial r_{ij}} \tag{7.20}$$

where r_{ij} is rule value from the winner node i ($1 \le i \le h$) in layer 3 to the node j ($j = 1, 2, \ldots, m$) in layer 4; t ($t = 1, 2, \ldots$) is the time period and η is the assigned learning rate. Thus, the rule value r_{ij} can be fine-tuned by equation (7.27):

$$\frac{\partial E}{\partial r_{ij}} = \frac{\partial E}{\partial f(net_j)} \cdot \frac{\partial f(net_j)}{\partial (net_j)} \cdot \frac{\partial (net_j)}{\partial r_{ij}} \tag{7.21}$$

From equations (7.10-7.14), we have:

$$\frac{\partial E}{\partial f(net_j)} = \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial f(net\ )} \cdot \frac{\partial f(net\ )}{\partial x_j} \tag{7.22}$$

$$\frac{\partial E}{\partial f(net\ )} = \frac{\partial E}{\partial \hat{y}(t)} = \frac{\partial \{1/2[y(t) - \hat{y}(t)]^2\}}{\partial \hat{y}(t)} = -[y(t) - \hat{y}(t)] \tag{7.23}$$

$$\frac{\partial f(net\ )}{\partial x_j} = \frac{(\sum \sigma_j \cdot m_j \cdot x_j) \cdot \sigma_j \cdot m_j - (\sum \sigma_j \cdot m_j \cdot x_j) \cdot \sigma_j}{\sum \sigma_j \cdot x_j} \tag{7.24}$$

$$\frac{\partial f(net_j)}{\partial (net_j)} = 1 \tag{7.25}$$

$$\frac{\partial (net_j)}{\partial r_{ij}} = x_{ij} \tag{7.26}$$

From equations (7.20-7.26), we have

$$r_{ij}(t+1) = r_{ij}(t) + \eta x_{ij}[y(t) - \hat{y}(t)] \frac{(\sum_{j=1}^{m} \sigma_j u_j)\sigma_j m_j - (\sum_{j=1}^{m} \sigma_j m_j x_j)\sigma_j^5}{(\sum_{j=1}^{m} \sigma_j u_j)^2} \tag{7.27}$$

where $r_{ij}(t)$ is the value of r_{ij} at time period t; x_{ij} is the input to node j in layer 4 from node i in layer 3; m_j is the mean of output membership function j in layer 4; σ_j is the variance of output membership function j in layer 4; x_j is the input from node j in layer 4 to the node in layer 5; net_j is the net value of node j in layer 4 and net is the net value of the node in layer 5.

The learning process is iterated until an acceptable error between actual value of output variable $y(t)$ and computed output $\hat{y}(t)$ is achieved. Depending on the data samples, the rule values may not be trained very well or completed. So the model needs to be retrained when new data are available.

7.4.2 Agent Knowledge Representation

As it has been shown, the intelligent forecasting model is a five-layered FNN. The structure, nodes, and weights of the forecasting model can be represented by the classes defined in the agent knowledge model. The structure of the FNN is represented by the class *NeuralNetwork*. The weights are represented by the class *Synapse*. The nodes are represented by the class *Neuron*.

As the nodes in each layer of the intelligent forecasting model have the same *sum* functions and *activation* functions, nodes in each layer can be further represented by an extended class of the class *Neuron*. In detail:

The first layer of the forecasting model is the input layer. Nodes in this layer just transfer inputs to the next layer. So the function *sumFunc()* simply sets the net value of a node to the input value. The function *activation()* outputs the net value to the next layer directly.

The second layer of the forecasting model is the fuzzification layer. The number of nodes depends on the number of membership functions for each node of layer 1. The function *sumFunc()* and the function *activation()* of each node implement the two equations of equation (7.4) and equation (7.5) respectively. The mean value and variance value of each

member function are set in training process. The class that represents nodes in layer 2 is shown in Figure 7.16.

Similarly, the function *sumFunc()* and the function *activation()* of a node in layer 3 implement equation (7.7) and equation (7.8) respectively, whereas the function *sumFunc()* and the function *activation()* of a node in layer 4 implement equation (7.9) and equation (7.10).

```
Public class layer2 extend neuron {
   Float m;
   Float v;
   Public void sumFunc() {…}
   Public void activation() {…}
   Public void getMean() {…}
   Public void getVarian() {…}
   Public void setMean() {…}
   Public void setVarian() {…}
}
```

Figure 7.16 The node class in layer 2

Finally, the function *sumFunc()* and the function *activation()* of nodes in layer 5 implement equation (7.12) and equation (7.13) respectively. The weights between nodes in layer 4 and nodes in layer 5 are calculated through the implementation of equation (7.11). The mean value and variance value are set and fine-tuned by training process.

Similarly, the weights can be set in the instances of class *Synapse*. Since the input node in layer 1 only has relationship with its own membership functions nodes in layer 2, the weight between each input node in layer 1 and its membership function node in layer 2 is unity whereas the weight between each input node in layer 1 and membership function node of other input nodes in layer 2 is zero.

The weight between a node in layer 2 and a node in layer 3 is simply unity whereas the weight between a node in layer 3 and a node in layer 4 needs to be set and fine-tuned by

the training process. The weight between a node in layer 4 and a node in layer 5 is the production of the mean value and variance value of the member function.

7.4.3 Implementing the E-forecasting Agent Generator

After the forecasting model is represented by the agent knowledge model, it can be bound to the knowledge unit of the forecasting agent generator.

As shown in Figure 7.17, class *intelligence* implements class *knowledgeUnit*. Within the class *intelligence*, function *setState()*, *reasoning()*, *training()* and *actionPlan()* have been defined in the agent knowledge model, only the function *learning()* needs to be implemented during the construction of an OEF agent according to the specific forecasting model.

```
public class intelligence implements knowledgeUnit {
 neuralNetwork nn;
  int state;
  int result;
  public void setState();
  public void reasoning();
  public void learning();
  public void training();
}
```

Figure 7.17 An example of knowledge unit implementation

For this specific forecasting model, there are three types of learning algorithms implemented in the function *learning()*:

- Grouping

  Grouping implements equation (7.15) by which the inputs are separated into different groups based on each preset mean value.

- Adjusting mean values

  According to the equation (7.16) and (7.17), each mean value is adjusted. The number of iterations t and parameters α, δ are preset based on experience or the expert knowledge.

- Fine tuning rule values

  The fuzzy rule identification and equation (7.27) are implemented. For every set of input data, the output of the neural network is obtained and the fuzzy rule will also be identified by nodes in layer 4. Therefore, the fuzzy rule values related to the identified fuzzy rule are fine-tuned through iterative computation. The learning rate η is preset according to experience or the expert knowledge.

```
public class ifa extends IntelligentAgent {
  public ifa(String name) {
    super(name);
  }
  public static void main(String[] args) {
    ifa IFAgent = new ifa("intelligent e-forecasting agent");
    neuralNetwork nn = new neuralNetwork(0, 0);
    nn.initNeuralNetwork(0);
…
    intelligence brain = new intelligence();
    brain.gainKnowledge(nn);
    IFAgent.knowledge = brain;
    IFAgent.process.intelligence = 1;
    IFAgent.doWork();
  }
}
```

Figure 7.18 The construction of a forecasting agent

After the class *intelligence* is fully defined, an intelligent agent can be constructed by extending the class *intelligentAgent* of the agent framework. A portion of a forecasting agent program is shown in Figure 7.18. A neural network instance *nn* is initialized by

restoring the knowledge about the model from the knowledge base. Then, an instance of the implementation of the knowledge unit, *brain*, is created. Finally the function *gainKnowledge()* of knowledge unit binds the neural network instance *nn* into the instance *brain*. When the agent starts running, it has the knowledge about the intelligent forecasting model. It is ready to do forecasting.

The forecasting agent generator will automate the above procedure to generate the agent dynamically. The agent generator first instantiates a neural network and initialized the neural network with the structure of the forecasting model retrieved from the knowledge base. Then an object *intelligence* which implements the knowledge unit of the agent, will be built to accommodate the neural network. Finally an intelligent agent is created and the object *intelligence* will be bound to the knowledge unit of the agent. When the created agent is started, the Goal Net for the forecasting agent will be loaded by the Knowledge Loader. The code for the created agent can also be output to a file so that the developer can change the code based on the specific requirement.

We have shown how to create an agent, load the Goal Net and bind the knowledge using MADE (Multi-Agent Development Environment) step by step through the design and implementation of an agent generator. A proof of concept prototype of an open e-forecasting agent system has been developed which shows that the proposed Goal Net, GO methodology and MADE framework are not only promising but also practical.

7.5 Multi-agent e-Forecasting System Prototype and Experiments

We have shown the detailed process for design and implementing an agent generator in the multi-agent e-forecasting system. A prototype of multi-agent e-forecasting system has been developed to prove the re-usability, practicability and flexibility of the proposed Goal Net, GO methodology, MADE agent development framework.

The prototype multi-agent e-forecasting system includes a number of agents, such as e-forecasting service agent, data collection agents, forecasting agents, forecasting model training agents, and forecasting agent generator. Similar to generating the forecasting agent, we may also generate data collection agent, training agent, etc. The agent generators enable the e-forecasting system to provide open services at anytime, from anywhere and in any form.

7.5.1 The Multi-agent e-Forecasting System

The e-forecasting agents are working with a knowledge base, a database and the remote data sources. The data sources can be remote databases, remote data files or web pages on the WWW. Figure 7.19 shows a multi-agent e-forecasting system. The agents maintain their knowledge in knowledge base. The collected data from remote data sources are stored in database for computing forecast results, future reference or retraining agent.

The data collection agent regularly connects to remote data sources to check for new data set or extracts the data from web pages. After the data are retrieved from data sources, the

forecasting agent is notified. In turn, the forecasting agent will coordinate to compute the forecasting result using its knowledge.

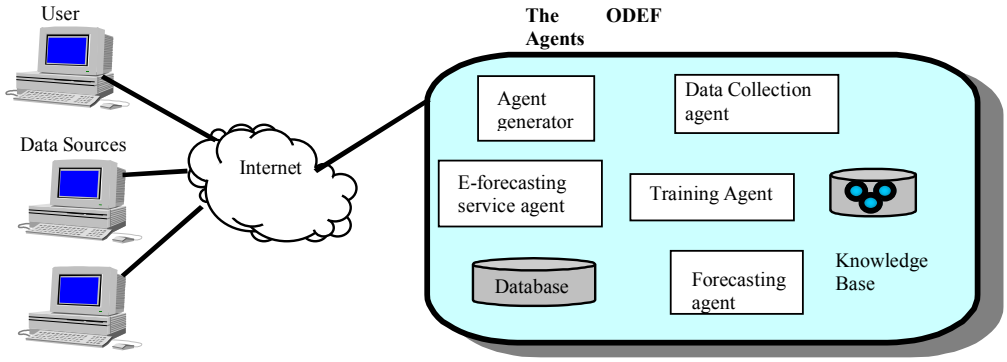

Figure 7.19 The system structure of the e-forecasting system

The perception unit of each agent senses the environment for any change so that the agent can work properly. For example, if the connection between the agent and the database is broken or the network between the agent and the remote resources has communication problems, the perception unit of the agent notifies the control unit about the changes. The control unit then suspends the goal reaching processes and sends messages to the user and other agents through the communication unit. Once the problems are fixed, the perception unit also notifies the control unit to resume the work.

7.5.2 Experiment

In the last section we described an intelligent e-forecasting model based upon fuzzy neural network and the knowledge representation of an intelligent e-forecasting agent using the proposed agent knowledge model to show that the proposed agent knowledge mode is able to represent and transform the forecasting model as the knowledge of the intelligent e-forecasting agent. In this section we'll conduct an experiment to show how

the agents in the prototype system cooperate together to generate forecasting results. Finally, the conclusions of the experiment are given.

In the experiment, we used the developed e-forecasting system to forecast the exchange rate of US dollar to Singapore dollar. The system architecture is shown in Figure 7.19. There are three input variables for the forecasting model [Li, 99]: Stock Exchange of Singapore Indices (x_1), Domestic Interest Rates (x_2), and Exports Value (x_3). A total of 69 data sets were used for training and testing. They were obtained from statistical reports [Yearbook of Statistics 1990-1995] published by the Department of Statistics, Singapore. We used the first 50 samples (from 01/1990 to 02/1994) for training and the rest for testing.

The data samples were first normalized to the range of {0,1}. The number of membership function nodes is 3 for x_1, 5 for x_2, 5 for x_3 and 3 for output y respectively. Therefore, the maximum number of possible fuzzy rules is 75 which equals to 3×5×5. Then we stored the meta-data of the neural network structure, preset parameter values and initial means, variances, weights including rule values into the knowledge base. The 50 training data sets were stored in the database. We created three simple web pages for the three input variables respectively to simulate the real world situation. Each page contains one set of testing data. Another program was running in the background updating the three web pages every 30 minutes using the other 19 testing data sets.

The data collection agent monitored and checked the web pages at intervals of 20 minutes. The forecasting agent would be trained by the training agent using the training data collected by the data collection agent in the database. Then the forecasting agent could generate the forecasting results using the collected testing data from the web pages.

Case 1: There are two forecasting models defined in the task list of transition *select forecasting model* for currency forecast. However they have different accuracy records based on the past evaluation. After each forecast, the selected forecasting model will be evaluated based on the real value when the value becomes appears. The forecasting model *X* has the accuracy ratio 0.5 and the other one *Y*, which is implemented based on the discussed fuzzy neural network, has the ratio 0.55. But in this test case, the model *Y* is occasionally unavailable.

In the transition load knowledge, the task is selected based on rules. In the tasks that meet the forecasting requirement, the task using the forecasting model with the highest accuracy ratio will always be selected. In this test case, we set the status of the task is unavailable. Then we printed out the log to see how the agent decided the task. Following is the output in the log file:

```
The transition is enabled
The task Y is selected (ratio = 0.55)
The task Y is not available
The task X is selected (ratio = 0.5)
The task X is available
```

This test case shows that the task availability does not affect the knowledge of the agent. The agent can dynamically select the suitable task for the goal pursuit.

Case 2: Following the case 1, now the model *Y* becomes available.

Now, when we re-send request to the agent, the output becomes:

```
The transition is enabled
The task Y is selected (ratio = 0.55)
The task Y is available
```

This test case shows that the task availability does not affect the knowledge of the agent. The agent can dynamically select the suitable task for the goal pursuit.

Case 3: Following the case 2, we want to test how the agents cooperate with each other to produce the forecasting results for a user. In order to monitor the coordination and the cooperation of the agents, we added a print statement in the agent class. Whenever an agent receives a message or sends a message, the content will be output to a flat file. Following are the list of the contents of the log files.

e-forecasting service agent.log:
send to agent generator
create a data collection agent
receive from agent generator
send to data collection agent
receive from data collection agent
send to agent generator
create a training agent
receive from agent generator
send to training agent
receive from training agent
send to agent generator
create a forecasting agent
receive from agent generator
send to forecasting agent
receive from forecasting agent

agent generator.log:
receive from e-forecasting service agent
Done
send to e-forecasting service agent
receive from e-forecasting service agent
Done
send to e-forecasting service agent
receive from e-forecasting service agent
The transition is enabled
The task Y is selected (ratio = 0.55)
The task Y is available
Done
send to e-forecasting service agent

data collection.log:
receive from e-forecasting service agent
Done
send to e-forecasting service agent
receive from training agent
Done
send to e-forecasting service agent
send to training agent

training.log:
receive from e-forecasting service agent
send to data collection agent
receive from data collection agent
receive from e-forecasting service agent

send to forecasting agent
Error
send to e-forecasting service agent
receive from e-forecasting service agent
send to forecasting agent
receive from forecasting agent
… (repeated, deleted)
send to forecasting agent
receive from forecasting agent
Done
send to e-forecasting service agent

forecasting.log:
receive from training agent
Done
send to training agent
… (repeated, deleted)
receive from training agent
Done
send to training agent
receive from e-forecasting service agent
Done
send to e-forecasting service agent

We analyzed the log files and found the coordination and cooperation among the five agents. The e-forecasting service agent sent a request to generate the data collection agent. The agent generator created the agent and notified the e-forecasting service agent. Then the e-forecasting service agent sent a request to the data collection agent. After the data were collected, the data collection agent informed the e-forecasting service agent. Then the e-forecasting service agent sent a request to the agent generator to generate the training agent. The agent generator created the agent and notified the e-forecasting agent. Then the training agent sent a request to the data collection agent. After the training agent got reply from the data collection agent it sent a request to the forecasting agent but got an error (the agent did not exist). Then the training agent sent the error message to the e-forecasting service agent. The e-forecasting service agent sent the agent generator to create the forecasting agent. Then the training agent sent request to the forecasting agent for forecast result. The forecasting agent generated the forecast result and sent back to the training agent. This procedure repeated until the last training data was used. Then the training agent sent a message to the e-forecasting service agent. The e-forecasting service

agent then sent a request to the forecasting agent to generate the forecast result for the user request. The forecasting sent back the forecast result to the e-forecasting service agent.

From this test case, we found the e-forecasting service agent coordinated the whole process. The training agent cooperated with the data collection agent and the forecasting agent towards its own goal. The agents worked collaboratively and finally generated the forecast result for the user. Figure 7.20 shows the collaboration diagram of the agents.

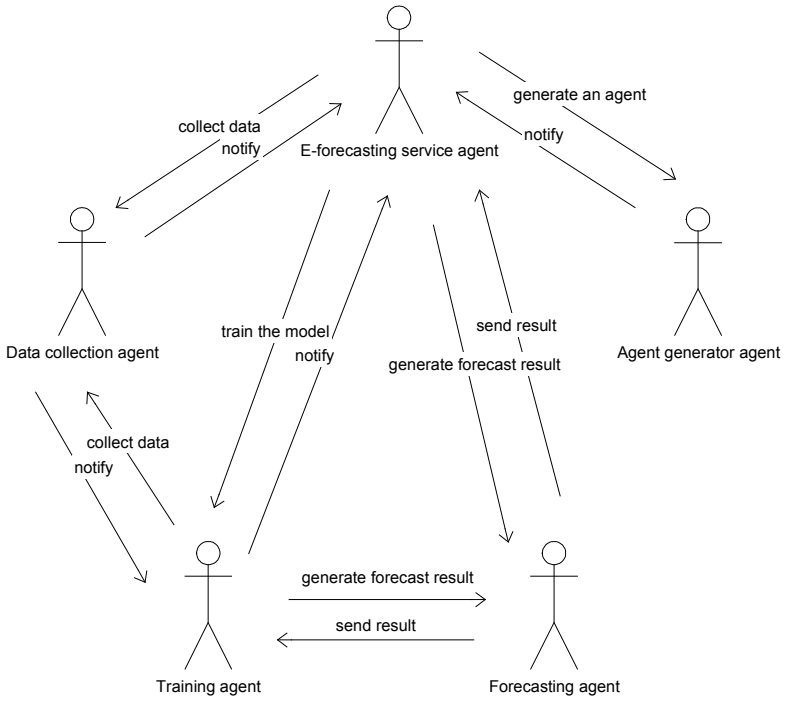

Figure 7.20 The agent collaboration diagram

Case 4: Following the case 3, we sent another forecast request to the e-forecasting service agent.

Following are the list of the contents of the log files.

e-forecasting service agent.log:

```
        send to data collection agent
        receive from data collection agent
        send to forecasting agent
        receive from forecasting agent

    agent generator.log:
        (empty)

    data collection.log:
        receive from e-forecasting service agent
        Done
        send to e-forecasting service agent

    training.log:
        (empty)

    forecasting.log:
        receive from e-forecasting service agent
        Done
        send to e-forecasting service agent
```

Form the log files we found that there is no new agent being created since all the agents were there already. The training agent was idle because the forecasting model was just trained.

Continuing the last test case, we repeated to send the forecast request until the last test data was used. Figure 7.21 listed the forecast results generated by the multi-agent forecasting system. It also lists the actual US dollar – Singapore dollar exchange rate and the result forecast by the forecasting agent.

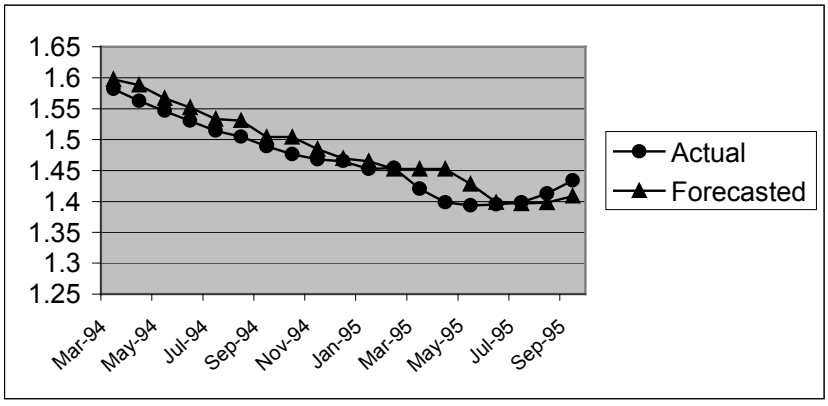

Figure 7.21 The trend of US dollar – Singapore dollar exchange rate

The example shows that a data collection agent can collect dynamic data from WWW efficiently and rapidly. And a neural network based e-forecasting agent can be easily constructed using proposed agent knowledge model and agent framework. The e-forecasting agent can learn and reason to produce the forecasting results using its knowledge, the e-forecasting model.

7.5.3 Conclusion from the Experiment

This experiment shows that: 1) agents in the multi-agent e-forecasting system are able to cooperate with each other, and manage the whole life cycle of forecasting processes, including forecasting model implementation, data collection/preparation, forecasting model training and forecasting result generations. 2) The multi-agent e-forecasting system is able to support open services in a distributed environment, new forecasting models can be added "on the fly" and new forecasting agents (services) can be autonomously created and composed in the system. 3) The Goal Net, GO methodology and MADE provide seamless connections between requirement and design, and between design and implementations. 4) Knowledge (Goal Net, Agent Knowledge Model) stored in the knowledge base are separated from the agent implementation, they can be dynamically loaded in runtime and can be highly re-used at both deployment time and runtime. The results of the experiment prove that the proposed goal-oriented agent development methodology is not only promising but also practical.

7.6 Summary

In summary, the successful development of the multi-agent e-forecasting system prototype shows that the agent-oriented e-forecasting approach leads to a new generation of intelligent e-forecasting systems which will not only automate the e-forecasting processes and free human beings from various tedious e-forecasting procedures but also provide open e-forecasting services that allow users to access the services from anywhere, at anytime and in any form. The goal-oriented (GO) modeling methodology as well as the goal autonomous agent model, and MADE framework have provided a practical way to build the new generation e-forecasting systems.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

The characteristics, environments and expectations of software systems have changed dramatically in the past few years. The environments of the software systems are moving to open, distributed, and dynamic operating environments. The traditional product-oriented software is moving to the service-oriented software rather than a closed product. Every service may include other services as its part, and may be included as a part of other services. Therefore research challenges have been raised on how to support service decomposition, integration and re-combination both at the deployment time and "on the fly".

Agents and multi-agent system are promising candidates for meeting the above mentioned challenges. However, the research on agent-oriented software engineering is relatively weak. Yet despite this intense interest, many concepts of the agent-oriented paradigm are

still not mature, and new methodology, especially the new techniques for agent modeling in practical use, have yet to be proposed.

This thesis presents a novel goal-oriented modeling approach, and agent-oriented software engineering methodology for modeling the complex goal of agents and for engineering agent-oriented systems in various application domains. Particularly, in this thesis we present 1) Goal Net, a goal-oriented modeling method; 2) Goal-oriented (GO) methodology for engineering agent-oriented systems; and 3) A multi-agent system development framework, namely, Multi-agent Development Environment (MADE).

The main contributions of this research include:

1) A novel goal oriented (GO) methodology for engineering agent-oriented systems for various application domains.

   GO methodology covers the whole development life cycle from goal-oriented requirement analysis, goal-oriented agent/multi-agent modeling, to design and implementation of goal autonomous agent/multi-agent systems. The current development of agent-based systems relies on different tools in different phases of the whole life cycle. The modeling and design of goal autonomous multi-agent systems using the proposed goal-oriented approach have demonstrated a novel agent-oriented software engineering paradigm for designing and developing complex software systems in open distributed environments (Chapter 5).

2) An original goal oriented modeling mechanism, Goal Net, for modeling and architecting MASs.

Goal Net serves as a goal-oriented agent modeling tool, an agent goal model, a goal-oriented requirement analysis tool, a multi-agent identification and organization model and an agent design/implementation model (Chapter 3).

- As an ***agent goal model***, Goal Net has a rich ability for describing various ***properties*** of agent's goals, such as fuzzy goal, partial goal, composite goal, temporal goal, etc. as well as the ***measurement*** of agent's goals. To date, few efforts have been reported for modeling agent's goals in a dynamic goal pursuing process and for facilitating such process by supporting fuzzy goals and partial goals. Goal Net models the ***dynamic goal relationships*** in a dynamic goal pursuing process in a changing environment. It ***supports flexible learning/reasoning mechanisms for dynamic goal pursuing***. A series of goal measurement, goal selection and action selection strategies and algorithms have been developed.

- Goal Net enables the agents to present both ***behavior autonomy and goal autonomy***. The autonomy currently used in agent literature is referable to the weaker notion of behavior autonomy (as opposed to goal autonomy) with the assumption that the goal of the agent is implicitly defined in agent behaviors. In a dynamic changing environment, agents must have control of its own goals and be able to decide its future goals autonomously. Without internal control of the agents' own goals, agents can irrationally pursue unrealistic goals. The goal selection and action selection mechanisms via flexible learning/reasoning

mechanism of Goal Net enable the agents to present both behavior autonomy and goal autonomy. (Chapter 3).

Compared with the existing agent goal models, such as task-oriented goal model, state-oriented goal model, etc., Goal Net presents advanced characteristics for facilitating dynamic goal pursuing via flexible learning/reasoning mechanisms in a changing environment and overcomes their limitations (Chapter 3).

- ***As a goal-oriented software modeling tool, Goal Net assists in all the phases of the life cycle for development of agent-oriented applications.*** Goal Net serves as a goal-oriented requirement analysis tool, a multi-agent identification and organization model and an agent design/implementation model. Goal Nets establish ***seamless connections between requirements, design and implementations***.

  Using Goal Net, an output of requirement analysis can be represented by a preliminary Goal Net that encapsulates the goals, the possible behavior for reaching the goal and the dynamic goal pursuing process, which can be further refined and implemented in the design and implementation stage. Goal Net supports both ***decomposition and re-combinations***. A complex Goal Net can be decomposed to a number of Goal Nets. A separate Goal Net can be combined with other Goal Nets and become a sub-Goal Net. The decomposition and re-combination features of Goal Net facilitate the decomposition and re-combination in requirements, design and implementation. Refer to the perspective of "delivering software as a service", every software might need to include other software as a part and might be included as a part for other software.

- Goal Net supports the multi-agent identification, organization through different agent identification/organization rules. Analyzing, designing and implementing software as a collection of interacting autonomous agents represents a promising approach for next generation of software. Agents in a multi-agent system act towards a common goal. In addition to an agent goal model, Goal Net also serves as a multi-agent identification, organization and coordination model. As described, Goal Net supports both ***decomposition and re-combinations*** via different rules (goal similarity, location, role, load balancing, security, etc.) Each decomposed Goal Net can be identified as an agent model for an individual agent in a multi-agent system. The Goal Net decomposition rules become the agent identification and organization rules.

3) A goal-oriented Multi-Agent Development Environment (MADE) for goal autonomous agent development which has integrated with the popular JADE, making MADE more powerful.

Despite the significant progress in the field of agent research and development, to date, there is still a lack of widespread development and deployment of agent systems and multi-agent systems. One of the major reasons is that research on narrowing the gap between agent formal models and agent implementation is rare. To bridge the gap, an agent model is proposed based on Goal Net and an agent development framework is developed. The proposed agent model separates agent body and agent brain, which makes Goal Net independent of agent implementation. A multi-agent development environment (MADE) has been developed for facilitating the agent development process (Chapter 4). Using the MADE, a dummy agent can be initialized

and a Goal Net can be dynamically loaded at run time. MADE has integrated with the popular agent development tool, JADE, which makes MADE more powerful.

In summary, the MADE not only eases the agent development but also enables the dynamic knowledge loading in runtime, and highly improves the reusability (knowledge re-use, task re-use).

In conclusion, the modeling, design and implementation of multi-agent systems using Goal Net, GO methodology, and MADE have demonstrated a novel agent-oriented software engineering paradigm for designing and developing complex software systems in open distributed environment. The goal-oriented approach proposed in this thesis not only enables a new goal-oriented modeling paradigm for agent-oriented software engineering but also bridges the gap between agent metal models and implementations, and facilitates the re-usable design and implementation of agent-oriented systems in various application domains.

8.2 Future Work

In spite of the reported contributions mentioned above, the current work can be extended in at least the following dimensions:

1) Further exploration of fuzzy goal, new goal derivation and goal measurement to extend and enhance the Goal Net.

2) Further exploration of the Goal Net for modeling mobile agents.

3) Investigation of design patterns for the goal autonomous agents development.

4) Further enhancement of the graphic Goal Net Designer for supporting collaborative design.

5) Exploration of various application domains of the proposed goal-oriented approach for modeling MAS in robot soccer, personal mobility and games.

8.2.1 Robot Soccer

A main challenge in autonomous robot soccer is how to enable robot to know what should do next to maximum the goal achievement of the whole robot soccer team in a dynamic changing situation. Behavior-based control is a popular paradigm, but current approaches to behavior design have some major limitations on team coordination. The aim of this work is to explore a goal-oriented approach based on Goal Net for modeling dynamic team strategies, team formation, team coordination as well as individual robot soccer's dynamic goal selection and action selection for a highly efficient team performance. A multi-agent robot soccer system is now being developed based on Goal Net, GO methodology and MADE framework.

8.2.2 Personal Mobility

In recent years, the research results of wireless communication have been successfully utilized to the commercial market. A broad range of network enabled mobile devices has been widely used and continues to play more and more important roles in our daily life. Mobile communication communities are currently preparing for various mobile data services to be accessed "anywhere, anytime, and in any form" [Gazis, 2002].

Today, mobile users already utilize a wide variety of mobile terminals ranging from simple mobile phones and Personal Digital Assistants (PDA) to high-end multimedia notebooks. Personal mobility services require an advanced architecture that integrates support protocol, mechanisms, and special functionality to dynamically reconfigure applications when users change from one terminal to another. This convergence raises the challenge of managing services while users roam from one device to another.

Goal-oriented modeling with Goal Net can be used to analyze and model complex personal mobility systems and goal autonomous agents can be used to fulfill the above demands in future generations of mobile communication systems. The advantages of using the goal-oriented modeling methodology and applying goal autonomous agents to the mobile communication systems can be identified as follows compared with traditional approaches:

(a) In mobile communication systems, agent identification and agent organization are two important issues. With the proposed goal-oriented modeling methodology, the service agents and their coordination are easily derived from the goal model.

(b) Mobile agents are able to move program logic to a remote host. In this case, services are no longer bound to a certain environment. Instead, they can be dynamically installed and used in exactly those places where they are required [Ghanea-Hercock, 99]. By extending the goal model for modeling mobile agents, mobile agents can be modeled and constructed.

(c) Moreover, the goal autonomous agents developed using the proposed methodology can reason their goals and appropriate actions autonomously in

the dynamic and even uncertain environment. They can learn from experience to improve their inference capabilities and adapt themselves to the changing environment.

We believe that the features of goal-oriented modeling methodology and the goal autonomous agents will bring a set of benefits for future generations of mobile communication systems.

8.2.3 Game Design

Game theory is defined as a theory of rational decision in conflict situations. It addresses the determination strategies for optimal play in a game, where a "game" is any situation involving multiple players and choice-dependent outcomes. Game models assume that each player is trying to maximize utility, and usually that the options and outcome utilities are knowledge common to all players. Each player's strategy determines one's course of action from a given position. The derivation of the "optimal" strategy is based on the mini-max concept, where each player maximizes the minimum values obtainable. It is also possible to try to anticipate the adversary's play and to select actions accordingly [Franken, 2003]. Goal autonomous agents are potential players for various game applications.

In summary, the research work carried out in this thesis, and the list of the future work to be done show that the research and design of goal-oriented modeling and goal autonomous agents are both interesting and challenging. Moreover, it can be applied into various application domains. The primary objective of this thesis, to present a new modeling diagram, goal-oriented modeling with the Goal Net, and a practical agent-

oriented software engineering methodology, from theoretical perspective to practical perspective, has been met.

References

[Aamodt, 94] A. Aamodt and E. Plazas, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches", AI Communications, Vol. 7, No. 1, pp. 39-52, 1994.

[Allen, 2000] P. G. Allen and R. Fildes, "Econometric forecasting", In Principles of Forecasting: A Handbook for Researchers and Practitioners, Kluwer Academic Publishers, Norwell, MA, 2000.

[Amandi, 97] A. Amandi and A. Price, "Towards Object-Oriented Agent Programming: the Brainstorm Meta-level Architecture", In Proceedings of the First International Conference on Autonomous Agents, pp. 510-511, 1997.

[Anton, 96] A. I. Anton, "Goal-based Requirements Analysis", In Proceedings of the Second IEEE International Conference on Requirements Engineering, April 1996.

[Anton, 97] A. I. Anton, "Goal Identification and Refinement in the Specification of Software-Based Information Systems", Ph.D. thesis, Department of Computer Science, Georgia Institute of Technology, June 1997.

[Armstrong, 2000] J. S. Armstrong, "Standards and practices for forecasting", In Principles of Forecasting: A Handbook for Researchers and Practitioners, Kluwer Academic Publishers, Norwell, MA, 2000.

[AWG, 99] Agent Working Group, "Agent Technology", Green Paper, Ver. 0.72, March 1999.

[Azoff, 94] E. M. Azoff, "Neural Network Time Series Forecasting of Financial Markets", John Wiley, New York, 1994.

[Baral, 2002] C. Baral, N. Tran and L. C. Tuan, "Reasoning about Actions in a Probabilistic Setting", AAAI/IAAI, pp. 507-512, 2002.

[Bauer, 2001] B. Bauer, F. Bergenti, P. Massonet and J. Odell, "Agents and the UML: A Unified Notation for Agents and Multi-agent Systems?", AOSE, pp. 148-150, 2001.

[Bellifemine, 99] F. Bellifemine, G. Rimassa and A. Poggi, "JADE - A FIPA-compliant Agent Framework", In Proceedings of the Practical Application of Intelligent Agents and Multi-Agents, London, UK, April 19-21, 1999.

[Boden, 72] M. Boden, "Purposive explanation in psychology", Cambridge, MA: Harvard University Press, 1972.

[Bonifacio, 2002] M. Bonifacio, P. Bouquet, R. Ferrario and D. Ponte, "Rationality, Autonomy and Coordination: the Sunk Costs Perspective", In the Third International Workshop on Engineering Societies in the Agents World (ESAW 2002), Madrid, Spain, September 16-17, 2002.

[Bradshaw, 96] J. M. Bradshaw, "Kaos: An open agent architecture supporting reuse, interoperability, and extensibility", In Tenth Knowledge Acquisition for Knowledge Based Systems, 1996.

[Bratman, 87] M. Bratman, “Intention, Plans, and Practical Reason”, Harvard University Press, Cambridge, MA, USA, 1987.

[Bresciani, 2001] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia and J. Mylopoulos, “Modeling Early Requirements in Tropos: A Transformation Based Approach”, AOSE, pp. 151-168, 2001.

[Brooks, 86] R. A. Brooks, “A Robust Programming Scheme for a Mobile Robot”, Proceedings of NATO Advanced Research Workshop on Language for Sensor-Based Control in Robotics, Castelvecchio Pascolu, Italy, September 1986.

[Brooks, 91] R. A. Brooks, “Intelligence Without Representation”, Artificial Intelligence Journal, Vol. 47, pp. 139-159, 1991.

[Busetta, 98] P. Busetta and K. Ramamohanarao, "An Architecture for Mobile BDI Agents", ACM Symposium on Applied Computing, 1998.

[Bussmann, 2004] S. Bussmann, N. R. Jennings and M. Wooldridge, “Multiagent Systems for Manufacturing Control – A Design Methodology”, Springer-Verlag Berlin Heidelberg, Germany, 2004.

[Buyya, 2001a] R. Buyya, D. Abramson and J. Giddy, “A case for economy grid architecture for service oriented grid computing”, In Proceedings of the 15th International Conference on Parallel and Distributed Processing Symposium, pp. 776 –790, April 23-27, 2001.

[Buyya, 2001b] R. Buyya and S. Vazhkudai, "Compute Power Market: towards a market-oriented grid", In Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 574 –581, May 15-18, 2001.

[Cao, 2002] J. W. Cao, et. al., "Agent-based resource management for grid computing", Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002, pp. 323 –324, May 21-24, 2002.

[Caire, 2001] G. Caire, W. Coulier, F. J. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. E. Kearney, J. Stark, R. Evans and P. Massoneta, "Agent oriented analysis using Message/UML", AOSE 2001, pp. 119-135, 2001.

[Carmel, 96] D. Carmel and S. Markovitch, "Learning Models of Intelligent Agents", AAAI/IAAI, Vol. 1, pp. 62-67, 1996.

[Castelfranchi, 94] C. Castelfranchi, "Guarantees for Autonomy in Cognitive Agent Architecture", ECAI Workshop on Agent Theories, Architectures, and Languages, pp. 56-70, 1994.

[Chapman, 86] D. Chapman and P. E. Agre, **"**Abstract reasoning as emergent from concrete activity**",** In Workshop of Reasoning about Actions and Plans, Portland, Oregon, June 1986.

[Corchardo, 98] J. M. Corchardo, B. Lees, C. Fyfe, N. Rees and J. Aiken, "Neuro-Adaptation Method for a Case-Based Reasoning System", IEEE World Congress on Computational Intelligence, Vol. 1, pp. 718, 1998.

[Curet, 96] O. Curet, M. Jackson and A. Tarar, "Designing and Evaluating a Case-Based Learning and Reasoning Agent in Unstructured Decision Making", 1996 IEEE International Conference on Systems, Man and Cybernetics, Vol. 4, pp. 2487-2492, 1996.

[Dardenne, 93] A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-Directed Requirements Acquisition", Science of Computer Programming, 20, pp. 3-50, 1993.

[Darimont, 98] R. Darimont, E. Delor, P. Massonet and A. van Lamsweerde, "GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering", In Proceedings of the 20th International Conference on Software Engineering (ICSE'98), Vol. 2, Kyoto, April 1998.

[DeMarco, 78] T. DeMarco, "Structured Analysis and System Specification", Yourdon Press, N.Y., 1978.

[Deugo, 99] D. Deugo, F. Oppacher, B. Ashfield and M. Weiss, "Communication as a Means to Differentiate Objects, Components and Agents", Technology of Object-Oriented Languages & Systems, IEEE, pp. 376-386, 1999.

[Dijkstra, 59] E. W. Dijkstra, "A note on two problems in connection with graphs", Numerische Mathematik, 1:269-271, 1959.

[Durfee, 96] E. H. Durfee, D. L. Kiskis and W. P. Birmingham, "The Agent Architecture of the University of Michigan Digital Library", In IEE Proceedings on Software Engineering, Vol. 144, No. 1, pp. 61–71, 1996.

[Faltings, 2000] B. Faltings, "Intelligent Agents: Software Technology for the New Millennium", Informatic/Informatique, Vol. 1, pp. 1-4, 2000.

[Ferguson, 92] A. Ferguson, "TouringMachines: An architecture for dynamic, rational, mobile agents", Ph.D. thesis, Comput. Sci. Lab., Univ. Cambridge, U.K., 1992.

[Fikes, 71] R. Fikes, "Monitored Execution of Robot Plans Producted by *STRIPS*", IFIP Congress (1), pp. 189-194, 1971.

[Finin, 93] T. Finin, et al., "Draft Specification of the KQML Agent Communication Language", External Interfaces Working Group, The DARPA Knowledge Sharing Initiative, June 1993.

[FIPA, 2001] FIPA00023, "FIPA Agent Management Specification", FIPA Specification 2000, August 2001.

[Foster, 2002] I. Foster, C. Kesselman, J. M. Nick and S. Tuecke, "The physiology of the Grid: An open Grid services architecture for distributed systems integration", http://www.globus.org/research/papers/ogsa.pdf, January 2002.

[Franken, 2003] N. Franken and A. P. Engelbrecht, "Evolving intelligent game-playing agents", Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology, September 2003.

[Franklin, 96] S. Franklin and A. Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", In Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, 1996.

[Gadomski, 93] A. M. Gadomski and J. M. Zytkow, "Paradigms, Foundations and Conceptualization Problems", Abstract Intelligent Agent, A. M. Gadomski, Ed., Rome, 1993.

[Garro, 2002] A. Garro and L. Palopoli, **"**An XML Multi-Agent System for e-Learning and Skill Management", In Proceedings of the 3rd International Symposium on Multi-Agent Systems, Large Complex Systems, and E-Businesses (MALCEB'2002), Erfurt, Germany, October 7-10, 2002.

[Gazis, 2002] V. Gazis, N. Houssos, A. Alonistioti and L. Merakos, "Evolving Perspectives on 4th Generation Mobile Communication Systems", IEEE 13th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2002), Coimbra, Portugal, September 2002.

[Genesereth, 94] M. R. Geneseretch and S. P. Ketchpel, "Software Agents", Communications of ACM, Vol. 37, pp. 48-53, 1994.

[Georgeff, 89] M. Georgeff and F. Ingrand, "Decision-Making in an Embedded Reasoning System", In Proceedings of the International Joint Conference on Artificial Intelligence, 1989.

[Ghanea-Hercock, 99] R. Ghanea-Hercock, J. C. Collis and D. T. Ndumu, "Cooperating mobile agents for distributed parallel processings", In Proceedings of the Third Annual Conference on Autonomous Agents, pp. 398–399, Seatle, WA, April 1999.

[Guessoum, 98] Z. Guessoum and J. Briot, "From Active Objects to Autonomous Agents", IEEE Concurrency – Special Series on Actors & Agents, 1998.

[Guttman, 98] R. Guttman, A. Moukas and P. Maes, "Agent-mediated Electronic Commerce: A Survey", Knowledge Engineering Review, Vol. 13, 1998.

[Hansen, 2001] E. A. Hansen and S. Zilberstein, "Monitoring and control of anytime algorithms: A dynamic programming approach", Artificial Intelligent, 126(1-2), pp. 139-157, 2001.

[Heckermann, 95] D. Heckermann and M. P. Wellman, "Bayesian Networks", Communications of the ACM, Vol. 38, No. 3, pp. 27-30, 1995.

[Hermans, 96] B. Hermans, "Intelligent Software Agents on the Internet: an inventory of currently offered functionality in the information society & a prediction of (near-)future developments", Tilburg, July 1996.

[Hongsoon, 2000] Y. Hongsoon, C. Kyehyun, K. Jongwoo and P. Sungjoo, "Architecture-Centric Object-Oriented Design Method For Multi-Agent Systems", In Proceedings of the Fourth International Conference on Multi-Agent System, pp. 469-470, 2000.

[Huhns, 97] M. N. Huhns and M. P. Singh, "Agents are everywhere!", IEEE Internet Computing, Vol. 1, Issue 1, pp. 87, January 1997.

[Huhns, 98] M. N. Huhns and M. P. Singh, "Readings in Agents", Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[Jacobs, 95] S. Jacobs and R. Holten, "Goal Driven Business Modelling - Supporting Decision Making within Information Systems Development", In Proceedings of Conference on Organizational Computing Systems, pp. 96-105, Milpitas, California, 1995.

[Janca, 95] P. C. Janca, “Pragmatic Application of Information Agents”, BIS Strategic Report, 1995.

[Jennings, 98] N. R. Jennings, K. Sycana and M. Wooldridge, "A Roadmap of Agent Research and Development", Autonomous Agents and Multi-Agent Systems, Vol. 1, Issue 1, pp. 7-38, 1998.

[Jennings, 2000] N. R. Jennings, "On Agent-Based Software Engineering", Artificial Intelligence, Vol. 117, No. 2, pp. 277-296, 2000.

[Jennings, 2001] N. R. Jennings, “An Agent-Based Approach for Building Complex Software Systems”, Communications of the ACM, Vol. 44, No. 4, April 2001.

[Kaelbling, 89] L. P. Kaelbling, “Intelligent Robots in the Real World”, IFIP Congress 1989, pp. 932-933, 1989.

[Kalakota, 2000] R. Kalakota and M. Robinson, “E-Business: Roadmap for Success”, Addison Wesley Longman Inc., 2000.

[Kavi, 2004] K. Kavi, “A Case for Agent-Oriented Software Engineering”, http://www.csrl.unt.edu/~kavi/Research/AOSE.pdf.

[Kendall, 98a] E. A. Kendall, “Agent Roles and Role Models”, In Proceedings of Intelligent Agents for Information and Process Management, 1998.

[Kendall, 98b] E. A. Kendall and M. Krishna, “Patterns of Intelligent and Mobile Agents”, In Proceeding of Autonomous Agents, pp. 92-99, 1998.

[Kim, 2000] J. Kim, K. Park and S. Park, “Agent Identification Method using Goal Modeling”, 2000.

[Kirsh, 91] D. Kirsh, "Foundations of AI: The Big Issues", Artificial Intelligence, Vol. 47, Issue (1-3), pp. 3-30, 1991.

[Kohonen, 98] T. Kohonen, "The self-organizing map", Neurocomputing, Vol. 21, Issue 1(3), pp. 1-6, 1998.

[Kolp, 2001] M. Kolp, P. Giorgini and J. Mylopoulos, "A Goal-Based Organizational Perspective on Multi-Agent Architectures", In Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), pp. 146-158, 2001.

[Kolp, 2002] M. Kolp, P. Giorgini and John Mylopoulos, "Organizational multi-agent architectures: a mobile robot example", AAMAS 2002, pp. 94-95, 2002.

[Kosko, 86] B. Kosko, "Fuzzy Cognitive Maps", International Journal of Man-Machine Studies, Vol. 24, pp. 65-75, 1986.

[Kosko, 92] B. Kosko, "Neural Networks and Fuzzy Systems", Prentice-Hall, Englewood Cliffs, NJ, 1992.

[Kotz, 99] D. Kotz and R. Gray, "Mobile Agents and the Future of the Internet", ACM Operating Systems Review, Vol. 33, pp. 7-13, 1999.

[Lamsweerde, 2001] A. V. Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour", In Proceedings of the 5th IEEE International Symposium on Requirements Engineering, pp. 249-262, Toronto, Canada, August 27-31, 2001.

[Lee, 89] T. W. Lee, E. A. Locke and G. P. Latham, "Goal setting theory and job performance", In L. A. Pervin (Eds.), Goal concepts in personality and social psychology, pp. 291-326, Hillsdale, NJ: Lawrence Erlbaum Associates, 1989.

[Lenz, 98] M. Lenz, H. D. Burkhard, B. Bartsch-Sporl and S. Wess, "Case-Based Reasoning Technology: From Foundations to Applications", Lecture Notes in Artificial Intelligence 1400, Springer Verlag, 1998.

[Li, 99] X. Li, C. L. Ang and R. Gray, "An Intelligent Business forecaster for Strategic Business Planning", Journal of Forecasting, 18, pp. 181-204, 1999.

[Lin, 92] C. T. Lin and G. Lee, "Neural-network-based fuzzy logic control and decision system", IEEE Transactions on Computers, 40, No. 12, December 1992.

[Liu, 2002] Z. Liu and L. Liu, "An Agent and Goal-Oriented Approach for Virtual Enterprise Modelling: A Case Study", Engineering Societies in the Agents World III: Third International Workshop (ESAW 2002), Madrid, Spain, September 16-17, 2002.

[Locke, 90] E. A. Locke and G. P. Lathman, "A theory of goal setting and task performance", Prentice-Hall, 1990.

[Macfadzean, 96] R. H. Macfadzean and K. S. Barber, "Reasoning about Autonomy in Multi Agent Systems", Semiotic Modeling for Sensible Agents Workshop, 1996.

[Maes, 91] P. Maes, "The Agent Network Architecture (ANA)", SIGART Bulletin 2(4), pp. 115-120, 1991.

[Maes, 94] P. Maes, "Agents that Reduce Work and Information Overload", Communications of the ACM, Vol. 37, No. 7, pp. 31-40, July 1994.

[Maes, 98] P. Maes, R. Guttman and A. Moukas, "The Role of Agents as Mediators in Electronic Commerce" Special Issue of Knowledge Engineering Review on Practical Applications of Agents, B. Crabtree, Ed., 1998.

[Mellor, 97] S. Mellor and R. Johnson, “Why Explore Object Methods, Patterns, And Architectures”, IEEE Software, Vol. 14, No. 1, pp. 27-30, 1997.

[Miao, 2002] C. Y. Miao, A. Goh and Y. Miao, “A Multi-Agent Framework for Collaborative Reasoning”, Intelligent Systems: Technology and Applications, Leondes, C. T., Ed., CRC Press LLC, 2002.

[Mylopoulos, 92] J. Mylopoulos, L. Chung and B. A. Nixon, “Representing and Using Nonfunctional Requirements: A Process-Oriented Approach”, IEEE Transaction on Software Engineering, Vol. 18, No. 6, pp. 483-497, 1992.

[Ndumu, 97] D. Ndumu and H. Nwana, “Research and Development Challenges for Agent-Based Systems”, In IEE Proceedings on Software Engineering, Vol. 144, No. 1, January 1997.

[Nwana, 99] H. Nwana and D. Ndumu, “A Perspective on Software Agents Research”, Knowledge Engineering Review, Vol. 14, No. 2, pp. 1-18, 1999.

[Odell, 99] J. Odell, “Objects and Agents: Is There Room for Both?”, Distributed Computing, November 1999.

[Odell, 2002] J. Odell, “Objects and Agents Compared”, Journal of Object Technology, Vol. 1, No. 1, pp. 41-53, 2002.

[Ohtani, 2000] S. Ohtani, M. Ishido and K. Shimohara, "Multi-agent based neural network as a dynamical brain model", in Proceedings of The Fifth International Symposium on Artificial Life And Robotics for Human Welfare and Artificial Life Robotics, 2000.

[OMG, 99] OMG Agent Working Group, "Agent Technology Green Paper", http://www.omg.org, 1999.

[OMG, 2000] OMG Agent Working Group, "Agent Technology Green Paper", Object Management Group, http://www.omg.org, March 2000.

[Osmar, 2002] R. Osmar and Zaïane, "Building a Recommender Agent for e-Learning Systems", In Proceedings of the International Conference on Computers in Education, pp. 55-59, Auckland, New Zealand, December 2002.

[Park, 2000] K. Park, J. Kim and S. Park, "Goal Based Agent-Oriented Software Modeling", In Proceedings of the 7th Asia-Pacific Software Engineering Conference (APSEC'00), Singapore, December 05-08, 2000.

[Rana, 2000] O. F. Rana and D. W. Walker, "'The Agent Grid': Agent-based resource integration in PSEs", In Proceedings of the 16th IMACS World Congress on Scientific Computing, Applied Mathematics and Simulation, Lausanne, Switzerland, August 2000.

[Rao, 92] A. Rao and M. Georgeff, "An Abstract Architecture for Rational Agents", In Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92), Morgan Kaufmann Publishers, 1992.

[Rao, 95] A. S. Rao and M. P. Georgeff, "BDI agents: From Theory to Practice", Tech. Rep. 56, Australian Artificial Intelligence Institute, Melbourne, Australia, 1995.

[Rentsch, 82] T. Rentsch, “Object-oriented Programming”, SIGPLAN Notices, Vol. 17, No. 9, pp. 51-57, September 1982.

[Rosenschein, 94] J. S. Rosenschein and G. Zlotkin, “Rules of Encounter: Designing Conventions for Automated Negotiation among Computers”, MIT Press, Cambridge, Massachusetts, USA, 1994.

[Rumbaugh, 91] J. Rumbaugh, et. al., “Object Oriented Modeling and Design”, Prentice-Hall, NJ, 1991.

[Rumelhart, 86] D. E. Rumelhart, G. E. Hinton and R. J. Williams, “Learning internal representations by error propagation”, Parallel Distributed Processing, Vol. 1, MIT Press, Cambridge, 1986.

[Russell, 95] S. J. Russell and P. Norvig, “Artificial Intelligence: A Modern Approach”, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1995.

[Sacerdoti, 74] E. D. Sacerdoti, “Planning in a Hierarchy of Abstraction Spaces”, Artifitial Intelligence, Vol. 5, No. 2, pp. 115-135, 1974.

[Sacerdoti, 75] E. D. Sacerdoti, “The Nonlinear Nature of Plans”, IJCAI 1975, pp. 206-214, 1975.

[Serrano, 2002] J. M. Serrano and S. Ossowski, “The design of agent communication languages: an organizational approach”, AAMAS 2002, pp. 555-556, 2002.

[Shen, 99] W. Shen and D. H. Norrie, "Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey", International Journal of Knowledge and Information Systems, Vol. 1, pp. 129-156, 1999.

[Silveira, 2002] R. A. Silveira, "Improving interactivity in e-learning systems with Multi-Agent architecture", P. De Bra, P. Brusilovisky, and R. Conejo (Eds.): Lecture Notes in Computer Science, Biarritz, Vol. 2347, pp. 466-471, Springer-Verlag Berlin Heidelberg, 2002.

[Simon, 67] H. A. Simon, "Motivational and emotional controls of cognition", Psychological Review, 74, pp. 29-39, 1967.

[Sloman, 78] A. Sloman, "The computer revolution in philosophy: Philosophy, science, and models of mind", Atlantic Highlands, NJ: Humanities Press, 1978.

[Sloman, 81] A. Sloman and M. Croucher, "Why robots will have emotions", In Proceedings of the Seventh International Joint Conference on Artificial Intelligence, pp. 197-202, Vancouver, B.C., Canada, 1981.

[Stephenson, 2000] T. A. Stephenson, "An Introduction to Bayesian Network Theory and Usage", IDIAP-RR 00-03, 2000.

[Stone, 2000] P. Stone and M. Veloso, "Multi-agent Systems: A Survey from a Machine Learning Perspective", Autonomous Robotics, Vol. 8, No. 3, pp. 345-383, 2000.

[Susan, 97] E. L. Susan, "Issues in Multi-Agent Design Systems", IEEE Expert, Vol. March-April, pp. 18-26, 1997.

[Sycara, 98] K. P. Sycara, "Multi-Agent Systems", AI Magazine, Vol. 19, pp. 79-92, 1998.

[Tucker, 2002] S. Tucker, A. Pigou and T. D. Zaugg, "e-Learning: making it happen now", Proceedings of the 30th annual ACM SIGUCCS conference on User services, November 2002.

[Webster, 2000] "Webster's English Dictionary", AM Productions, Ottawa, Canada, 2000.

[Wilderman, 2000] J. Wilderman, "The Es Have It: E-Business, E-Commerce, E-tailing and the Web", Gartner Group Research Report, 2000.

[Wood, 2001] M. Wood and S. A. DeLoach, "An Overview of the Multiagent Systems Engineering Methodology", Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Vol. 1957, P. Ciancarini and M. Wooldridge, Eds., Springer Verlag, Berlin, January 2001.

[Wooldridge, 95] M. Wooldridge and N. Jennings, "Intelligent Agents: Theory and Practice", Knowledge Engineering Review, Vol. 10, 1995.

[Wooldridge, 99] M. Wooldridge, N. Jennings and D. Kinny, "A Methodology for Agent-Oriented Analysis and Design", In Proceedings of Agent'99, 1999.

[Wooldridge, 2000a] M. Wooldridge, "Reasoning about Rational Agents", The MIT Press, July 2000.

[Wooldridge, 2000b] M. Wooldridge and P. Ciancarini, "Agent-Oriented Software Engineering: The State of the Art", *Agent-Oriented Software Engineering*, Lecture Notes in AI, Vol. 1957, Springer-Verlag, 2000.

[Wooldridge, 2000c] M. Wooldridge, N. R. Jennings and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design", Autonomous Agents and Multi-Agent Systems 3(3), pp. 285-312, 2000.

[Woodridge, 2001] M. Wooldridge, G. Wei, P. Ciancarini, "Agent-Oriented Software Engineering II", Second International Workshop, AOSE 2001, Montreal, Canada, May 29, 2001.

[Wooldridge, 2002] M. Wooldridge, "An Introduction to Multiagent Systems", Wiley, New York, 2002.

[Yu, 2001] E. Yu, "Agent Orientation as a Modelling Paradigm", Wirtschaftsinformatik, 43(2), pp. 123-132, April 2001.

[Zadeh, 65] L. A. Zadeh, "Fuzzy Sets", Information and Control, 8, pp. 338-353, 1965.

[Zambonelli, 2003] F. Zambonelli, N. R. Jennings and M. Wooldridge, "Developing multiagent systems: The Gaia methodology", ACM Transaction on Software Engineering Methodology, Vol. 12, No. 3, pp. 317-370, 2003.

[Zurada, 92] J. M. Zurada, "Introduction to Artificial Neural System", West Publishing Company, USA, 1992.

Author's Publications

The following work was published with respect to this research:

Journal Articles and Book Chapters

1. **Zhiqi Shen**, Robert Gay and Xuehong Tao, **Goal-Based Intelligent Agents,** *International Journal of Information Technology,* Vol 9, No. 1, pp. 19-30, 2003.

2. **Zhiqi Shen** and Robert Gay, **Agent Oriented Business Forecasting: A New Approach for Building Intelligent Business Forecasting Systems,** *International Journal of Intelligent Systems in Accounting, Finance and Management, to appear.*

3. **Zhiqi Shen**, Robert Gay and Yuan Miao, **Agent-based E-Learning System – a Goal-based Approach**, *Intelligent Knowledge-Based Systems: Business and Technology of the New Millennium*, Ed. C. T. Leondes, Kluwer Academic Press International, Vol. 3, Chapter 6, 2004.

4. **Zhiqi Shen**, Robert Gay, Xiang Li and Zhonghua Yang, **An Agent Approach for Intelligent Business Forecasting,** *Chapter of Intelligent Systems: Technology and Applications, Volume V: Manufacturing, Industrial and Management Systems*, C. T. Leondes Eds, CRC Press/Lewis Publishers, Boca Raton, FL, USA, 2002.

Conference Papers

5. **Zhiqi Shen**, Chunyan Miao, Robert Gay and Qiong Wang, **Goal-oriented Modeling for Agent Mediated E-Learning Grid,** *Proceedings of the 8th International Conference on Control, Automation, Robotics and Vision (ICARCV 2004)*, Kunming, Yunnan, China, December 6 - 9, 2004.

6. **Zhiqi Shen**, Chunyan Miao and Robert Gay, **Goal Autonomous Agent Architecture,** *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC 2004)*, Hong Kong, China, September 28 - 30, 2004.

7. **Zhiqi Shen**, Chunyan Miao and Robert Gay, **Goal Oriented Modeling for Intelligent Software Agents,** *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04)*, Beijing, China, September 20 - 24, 2004.

8. **Zhiqi Shen**, Chunyan Miao, Angela Goh and Robert Gay, **Agent Mediated Grid Services in e-Learning,** *Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2004)*, Chicago, Illinois, USA, April 19 - 22, 2004.

9. **Zhiqi Shen** and Robert Gay, **Goal-based Modeling for Intelligent Business Forecasting Agents,** *Proceedings of the First International Conference on Fuzzy Systems and Knowledge Discovery (FSKD2002)*, Singapore, November 18-22, 2002.

10. **Zhiqi Shen** and Robert Gay, **Agent Oriented Business Forecasting,** *Proceedings of the Pacific Asian Conference on Intelligent Systems (PAIS2001)*, Seoul, Korea, 16-17 November, 2001.

11. **Zhiqi Shen**, Xiang Li, Robert Gay, Yuan Miao, Zhonghua Yang, **An Intelligent Agent Architecture for Business Forecasting,** *International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000)*, *International Congress on Intelligent Systems And Applications (ISA'2000)*, Wollongong, Australia, 11-15 Dec. 2000.

12. **Zhiqi Shen**, Robert Gay, Yuan Miao, **Intelligent Forecasting Agent,** *The 20th International Symposium on Forecasting,* Lisbon, Portugal, 14 Jun. 2000.