Multi-agent development environment (MADE) and its applications

Li, Dongtao

2007

Li, D. (2007). Multi-agent development environment (MADE) and its applications. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/35732>

<https://doi.org/10.32657/10356/35732>

Multi-Agent Development Environment (MADE) And Its Applications

Li Dongtao

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Master of Engineering

2007

Abstract

Agent technology represents a new software paradigm. The autonomous, intelligent and goal-oriented characteristics make agent technology a promising solution for the next generation of software products. However, despite of enormous research efforts on agent modelling, there is still a lack of widespread deployment of agent systems. The major reason is that the research narrowing the gap between agent mental state design and agent implementation is rare. This thesis presents a comprehensive agent development environment to facilitate the agent system design and development.

The proposed environment, namely Multi-Agent Development Environment (*MADE*), is based on Goal Net Model and Goal Net Agent Development Methodology. *MADE* is able to simplify agent development work by providing comprehensive guidance to developers from agent mental state design to agent implementation. The environment consists of two major modules: Goal Net Designer and Agent Creator. Goal Net Designer provides a graphical user interface for designing Goal Net, the agent mental model. According to the Goal Net design, Agent Creator is able to create appropriate agents which are minded with the designed Goal Net. The low level details, such as how an agent manages its Goal Net and how an agent executes different tasks to pursue its goals, become transparent to developers and are automatically managed by *MADE*. Hence, agent developers can focus their work on how to design their agent system instead of how to convert their design into implementation.

This thesis addresses the key issues in agent system design and development. The first important issue is Goal Net representation and management. Although Goal Net model has clearly defined each component of it, a proper data structure is still necessary in order to manipulate the model within a lower level context. The second

important issue is agent architecture, which will directly affect the ability and performance of agents. A hybrid agent architecture is designed to make the agent suitable for manipulate Goal Net and also extensible for other agent functions.

To evaluate the performance of MADE, besides a systemic comparison with other agent development environments, several applications have been developed. An agent-based e-learning system and an interactive learning world are developed using MADE, and their development procedures are illustrated in the later chapters of this thesis. These studies show that MADE is able to effectively facilitate the design and implementation work in agent system development.

Acknowledgements

I would like to express my deepest thanks to my supervisor, Dr. Miao Chun Yan, for bringing me into this research field and for her continuous guidance, encouragement and suggestions throughout the course of my research. Her kind help has contributed enormously to my research work and the production of this thesis.

I would like to offer my special thanks to Dr. Shen Zhi Qi, for his invaluable advices on my research work. He is always approachable when I need his advices. Discussions with him have given me a lot of inspirations in my research.

I would like to express sincere thanks to Mr. Weng Jian Shu, an enthusiastic and kind friend. His selfless share of his research experience and enthusiastic help have benefited me a lot in my research work.

I would like to thank Mr. Cai Yun Dong, Mr. Chathapuram V Satish, Mr. Choo Ye Liang, Ms. Ramakrishnan Sowmya, Mr. Surajit Goswami and Ms. Peng Xiao Feng, for their kind help and support for the implementation of case study applications in my research.

Last but not least, my thanks also goes to Emerging Research Lab of School of Computer Engineering, for providing various facilities support to carry out my research.

Author's Publication

Journal Articles

- (i) Zhiqi Shen, Chunyan Miao, Robert Gay and Dongtao Li ***Goal-Oriented Methodology for Agent System Development***, *IEICE Transactions on Information and Systems*, 2006.E89-D(4):p.1413-1420.

Conference Paper

- (i) Zhiqi Shen, Dongtao Li, Chunyan Miao and Robert K.L. Gay ***Goal-oriented Methodology for Agent System Development***, *IAT 2005*, 95-101.
- (ii) Dongtao Li, Zhiqi Shen, Chunyan Miao and Robert K.L. Gay ***Goal Oriented E-Learning Agent System***, *KES(1) 2005*, 664-670.
- (iii) Jianshu Weng, Chunyan Miao, Angela Goh and Dongtao Li ***Trust-based Collaborative Filtering***, *CIKM 2005*, 299-300.

Contents

Abstract	i
Acknowledgements	iii
Author's Publication	iv
List of Figures	viii
List of Tables	x
1 Introduction	1
1.1 Motivation	1
1.2 Research Objectives	3
1.3 Contributions	4
1.4 Thesis Organization	5
2 Theoretical Background and Related Works	6
2.1 Agent Basics	6
2.1.1 Agent Definitions	6
2.1.2 Goal-oriented Agent	9
2.1.3 Agent Architectures	10
2.2 Goal Net	12
2.2.1 Goal Net Model	12
2.2.2 Goal Net Agent Development Methodology	14
2.3 Agent Development Frameworks and Tools	17
2.3.1 JADE	17
2.3.2 JACK	19
2.3.3 ZEUS	20
2.3.4 MADKIT	22

2.3.5	NetLogo, RePast and Swarm-based Agent Toolkits	23
2.4	Summary	24
3	Multi-Agent Development Environment (MADE)	25
3.1	System Overview	25
3.2	Goal Net Management	27
3.2.1	Goal Net Description Language	27
3.2.2	Goal Net Manipulation and Management	33
3.3	Goal-oriented Agent	35
3.3.1	Agent Architecture	35
3.3.2	Goal Net Processing	37
3.3.3	Perception and Reactions	45
3.4	MADE Work Flow	48
3.4.1	Goal Net Designer	48
3.4.2	Agent Creator	52
3.5	System Evaluation	53
3.6	Summary	57
4	Developing an E-learning System with MADE	58
4.1	Introduction	58
4.2	Motivation Scenario and Challenges	59
4.3	A Learning Goal Net	60
4.4	Agent-Mediated Learning System	65
4.5	Summary	67
5	An Agent-augmented Virtual World for Interactive Learning	69
5.1	Introduction and Motivations	69
5.1.1	Agents for Interactive Learning	69
5.1.2	Virtual Environment for Interactive Learning	70
5.2	Learning Scenarios Design	72
5.3	Learning Goal Net Design	75
5.4	Learning World Construction	79

5.5	System Development	80
5.6	Summary	81
6	Conclusion and Future Work	83
6.1	Conclusion	83
6.2	Future Work	85
	References	87

List of Figures

2.1	Agent typology by Nwana [1]	9
2.2	Goal Net sample	13
2.3	Goal Net Methodology overview	15
2.4	JADE platform overview [2]	18
3.1	System overview of MADE	26
3.2	Direct description of Goal Net model	27
3.3	Key components of Goal Net Description Language	28
3.4	Illustration of loosely coupled Goal Net description	30
3.5	Goal Net Description Language sample : state description	31
3.6	Goal Net Description Language sample : Goal Net description	32
3.7	Goal Net Description Language sample : transition description	32
3.8	Agent architecture [3]	36
3.9	Overview of Goal Net process	38
3.10	Composite State Process Flow	40
3.11	Concurrent Transition Process Flow	42
3.12	A sample Goal Net	44
3.13	Status of stacks and queues in Goal Net processing	44
3.14	Agent's perception and reaction	46
3.15	Goal Net Designer	48
3.16	Goal Net Designer	50
3.17	Overview of Agent Creator	52
4.1	High level states of the learning Goal Net	62
4.2	Pre-Assessment Goal Net	62

4.3 General rules in Goal Net construction 64

4.4 Deriving agents from the learning Goal Net 66

4.5 Structure of the learning system 67

5.1 Active Worlds snapshot 71

5.2 Overview of Goal Net for interactive learning 75

5.3 State decomposition of Information Collection 76

5.4 Snapshots of River City 79

5.5 System architecture of River City 80

List of Tables

3.1	Class GoalNet	34
3.2	Class State	34
3.3	Class Transition	34
3.4	Class Task	35
3.5	Comparison of MADE, JADE and JACK	54
5.1	Transitions & Tasks for Information Collection	77

Chapter 1

Introduction

1.1 Motivation

Agent technology represents a new paradigm in software engineering [4]. It provides new means of analyzing, designing and building complex software systems [5]. It is predicted to have a major impact on future generation of software in various application domains including e-learning, e-business, grid computing, intelligent manufacturing etc [6, 7].

The concept “Agent” was originally used in the domain of *Artificial Intelligence (AI)* in 1970s, in which it represents an entity with intelligence. The use of agent became widespread in 1980s, especially in Expert System and Distributed Computing. Since 1994, agent has become one of the major research areas in computer science [5]. Integrating with other technologies, such as high performance computer and network, agent systems have been used in many research fields. It has been forecasted that in ten years time most software developments will have agent systems involved in [8].

Unfortunately, despite of the research effort in academics, there is still a lack of widespread deployment of agent systems for industrial use [9]. Most of the research efforts

CHAPTER 1. INTRODUCTION

are focusing on the agent theories, such as agent mental state models, classifications of agent's features and characteristics and possible agent applications in some domains. These researches provide the theoretical foundations of the agent system; however, they are not practical enough and detailed enough to guide agent developers directly on how to develop an agent system step by step. It is a formidable challenge for agent developers to translate the theoretical models to implementations as the lack of complete agent development methodologies and complementary tools.

To give some directions in agent development, a number of methodologies have been proposed in recent years, such as *Gaia* [10], *Tropos* [11, 12], *MaSE* [13, 14], *Goal Net* [3], and *Extending-UML* [15] etc. These methodologies have provided theoretical foundations for agent system development, and they are proposed based on certain specific agent models (such as *Believe-Desire-Intention (BDI)* [16, 17] model and social agent model) and from different perspectives. For example, *Gaia* focuses on the social role modeling of each agent and it only gives a conceptual guide for agent system development; others, like *Tropos* and *MaSE*, concern more about the internal of each agent, like architecture design and mental modeling, and they provides comprehensive guidance from requirements analysis to system implementation. In fact, most of the agent development methodologies can be classified into two types: those extending from *Object Oriented (OO)* development methodology (like *MaSE*) and those extending from *Knowledge Engineering (KE)* (like *Tropos*) [18]. The former is easier to be understood and accepted by object oriented software engineers, but it loses the power on agent mental state design. On the other hand, the latter, as it borrows concepts from knowledge engineering, is convenient to model agent mental state, but it requires tedious work for programmers to implement a system. Currently there is no methodology that has the perfect mixture of these two classes. The trade-off between the ability on agent mental state modeling and the degree of difficulty on

CHAPTER 1. INTRODUCTION

agent system implementation, has limited the capability of these methodologies to bridge the agent modeling and agent implementation gap. To solve this problem, one rational way is to use the methodology that extends from knowledge engineering to model agent's mental state, and use some supporting tools providing guidance and code generation functions to simplify agent programmers' work.

There are some agent development tools emerging as more research efforts have been put in. The two established and well recognized agent development environments are: *JADE (Java Agent Development Framework)* [2] and *JACK* [19]. JADE provides a general agent running environment but it does not directly support agent mental state development. JACK, comparing with JADE, is rather comprehensive and provides full cycle support of agent development; at the same time it is much more complicate and very difficult to use, which has, to some extent, limited the simplification of developers' work. The details of these two agent development tools, and other related tools, will be discussed in the next chapter.

Thus, to effectively bridge the gap between agent model design and agent system implementation, a new agent development environment, which is able to provide comprehensive guidance to agent developers, is highly needed. In this thesis, a new agent development tool based on Goal Net methodology, Multi-Agent Development Environment (MADE) is proposed to simplify the work on agent system design and development.

1.2 Research Objectives

There are many challenges involved in designing an effective agent development environment. This research aims to address some of the major challenging issues, and the objectives of this research are listed as follows:

CHAPTER 1. INTRODUCTION

- To design a comprehensive agent development environment as a complementary tool for Goal Net agent development methodology, to assist and simplify the work in goal-oriented agents development. In order to bridge the gap between agent mental state design and implementation, the desired environment should provide both Goal Net design tool and use suitable agent architecture to process the designed Goal Net.
- To propose a Goal Net description language that is able to accurately describe Goal Net and represent it in software agents. This language must be highly flexible in order to maintain the integration and decomposition of Goal Net.
- To evaluate the proposed agent development environment by comparing it against some well established agent development platforms according to a list of standard criteria.
- To demonstrate how the desired agent development environment can be used to design and develop agent systems to solve problems in different application domains.

1.3 Contributions

The main contributions of this research include:

- A Goal Net description language. This flexible language is able to describe Goal Net and enable agents to easily understand and process Goal Net. It provides a simple and effective means to represent goal net in the low level context, like software programs.

CHAPTER 1. INTRODUCTION

- A comprehensive agent development environment. This environment, including Goal Net Designer and Agent Creator, covers both the design and implementation phases in agent system development.
- An evaluation of the proposed MADE through comparison with other agent development platforms.
- Several case studies to explore and demonstrate the realization of the real world agent applications using Goal Net and the proposed MADE in different domains.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 presents the theoretical background and related works. Chapter 3 illustrates the details of MADE design, especially the Goal Net description language, agent architecture, agent management tools, and the work flow of MADE. It also provides a systematic evaluation of MADE by comparing with other agent development platforms in different dimensions. Chapter 4 & chapter 5 present two application cases of Goal Net and MADE: an agent-based e-learning system and an agent-augmented interactive learning world. Finally chapter 6 discusses the future work and concludes this thesis.

Chapter 2

Theoretical Background and Related Works

This chapter consists of four sections. Section 2.1 explains the basic concepts of agents and agent systems. Section 2.2 illustrates the theoretical foundation of the proposed MADE: Goal Net model. Section 2.3 discusses the related works in agent development toolkits and several well established agent development frameworks. Finally research challenges are summarized in section 2.4.

2.1 Agent Basics

2.1.1 Agent Definitions

The word *Agent* has been used frequently in many research fields recently. You may have heard about *learning agent*, *intelligent agent* [20], *personal agent* [21], *mobile agent* [7], and *driving agent* [22] and so on. However, the definitions of these agents may not be exactly same; or even worse, they are defined far different from each other. Most of the time, people who are not from agent research area just have a common expectation on agents, which is that agents can do something for human being.

Actually even in the agent research field itself, there is no universally accepted de-

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

definition for agents [20]. Different views on agents have emerged and existed in the research collection. Agent has been defined as “persistent software entity dedicated to a specific purpose” [23], “computer programs that simulate a human relationship by doing something that another person could do for you” [24], “active object” [25] and “smart object” [26] etc.

Due to the diversity of agent usage, it is very difficult to give a unified definition for agent [20]. Instead of giving a precise description on *what agents are*, it is more reasonable to characterize and categorize agents along certain dimensions [27] and measure their properties. One of the most famous and foundational classification of agent’s characteristics is given by Wooldridge and Jennings [20]. They distinguished agent’s characteristics into *a weak notion* and *a strong notion* according to the usage of agents. The weak notion of agency is identified by four properties: *autonomy*, *social ability*, *reactivity* and *pro-activeness*.

Autonomy An agent can control its actions and internal states, and it can operate without the direct interactions of human being. This is one of the fundamental differences between an agent and an object [26]. An agent is able to have control of itself for both the actions it is executing and the goals it is pursuing. The autonomy property allows an agent to be a dynamic entity and be able to perform differently under different situations without supervision.

Reactivity An agent can perceive its living environment, and it can respond to the environment changes. Theoretically, agent’s living environment can be anything that is sensible and measurable such as the physical world, internet, computers, and printers and so on. Instead of dealing with the high level agent knowledge and goals, most of the time the reactivity can be achieved simply by an appropriate collection of stimulus-response, input-output or conditional rules [28].

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

Pro-activeness Besides responding to the environment changes, an agent can exhibit some goal-oriented behaviors by taking initiatives. Pro-activeness is at the opposite side of the reactivity of an agent; it focuses more on how an agent can actively plan and execute the appropriate actions in order to achieve its goal irrespective of the environment changes.

Social Ability Agents can interact with each other via some kinds of *Agent Communication Language* [29, 30]. This property unleashes the power of cooperation and allows a group of agents to work towards a common goal. Moreover, a social role can be assigned to an agent and organization can be built on top of a group of agents to make agents work more efficiently [31, 32].

In Wooldridge and Jennings’s work, the weak notion identifies the most fundamental characteristics of an agent, and it has been well recognized as the essence of *agenthood* [33] by agent research community. Besides the weak notion, they also provided a strong notion of agency [20]. The strong notion is at the higher level comparing with the weak notion. In the strong notion, an agent should have the properties listed in the weak notion; at the same time, it should also implement the concepts that are more usually applied to human being. These concepts include knowledge, belief [17], intention, and emotion and so on. The strong notion gives a more complex picture of agent and requires an agent to behave more like a human being. This notion is more preferred in artificial intelligence research. In this research the weak notion of agency is used.

Besides Wooldridge and Jennings’s work, some other agent characteristics have also been defined for agents in different usage domains. For example, an agent with mobility may travel through a network [7]; an agent with learning ability can gain experience and improve its skills during its work [34]; a group of collaborative agents can work

Figure 2.1: Agent typology by Nwana [1]

together as a team. Nwana has defined an agent typology along three agent properties: *Autonomy*, *Cooperation* and *Learning* [1]. Several different types of agents are identified by the combination of these three properties, as shown in Figure 2.1.

In general, the additional properties other than those listed in *the Weak Notion* are not always necessary for all the agents; they should be implemented based on the application requirements.

2.1.2 Goal-oriented Agent

Among different agent’s characteristics, “goal-oriented” has been increasingly emphasized as one of the most important features that distinguish agents from objects [35]. In fact, many people have argued that goal-oriented agents are the agents that can achieve real agent autonomy, which is defined as one of the primary characteristics in the weak notion of agency. There are two levels of agent autonomy: *behavior autonomy* and *goal autonomy* [36].

Behavior autonomy is the basic level of agent autonomy. A behavior autonomous agent can dynamically select and execute an appropriate action to respond to the environment changes or pursue its goal. The agent can control what it is doing;

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

however, it may have no idea about what it is doing for. It is very possible that an agent with merely behavior autonomy irrationally pursues unrealistic goals by keeping repeating actions [3]. Thus behavior autonomy is not the only autonomy that an agent needs [37].

Goal autonomy is at the higher level of agent autonomy. An agent with goal autonomy is able to select an appropriate goal to pursue based on its living environment and its own conditions. Such an agent will not pursue any unreachable goal. Further more, a goal autonomous agent can work more efficiently by selecting appropriate goals [3, 38].

A goal-oriented agent is an agent that is not only behavior autonomous, but also goal autonomous. The agent is designed to be goal-oriented, and it always tries to find an appropriate next goal and execute the proper actions to achieve its goal. In our desired MADE, agents created by the environment shall be goal-oriented agents.

2.1.3 Agent Architectures

Agent architecture defines how to build or construct a software entity that has agent properties. It specifies “how the agent can be decomposed into construction of a set of component modules and how these modules should be made to interact” [39]. Different applicable agent architectures have been developed, and these agent architectures, in essence, can be categorized into the following three types [20].

Deliberative Agent Architecture Deliberative agent architecture is a classical agent architecture. Jennings defined a deliberative agent as the one “that contains an explicitly represented, symbolic model of the world, in which decisions are made via logical or pseudo-logical reasoning, based on pattern matching or symbolic manipulation” [20]. An agent with deliberative architecture has a mental state that reflects

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

its living world, and it is also able to make some decisions to control its actions. Comparing to agents with reactive architectures, which will be explained later, deliberative agents are more active and autonomous as the decision making are based on internal reasoning rather than event triggering from outside environment. A number of deliberative agent development attempts have been made and some deliberative architectures have been built, such as *Intelligent Resource-Bounded Machine Architecture (IRMA)* [40], *HOMER* [41], *GRATE* [42] etc. Most of them have implemented the well recognized *Beliefs, Desires, and Intentions (BDI)* agent model [16].

Reactive Agent Architecture A reactive architecture is an alternative architecture of deliberative agent architecture. As its name indicates, this type of architecture emphasizes the interaction between an agent and its environment. Instead of focusing on the complex internal reasoning and symbolic manipulation, a reactive agent acts using stimulus-response type of behavior responding to the present state of its living environment [43]. This does not mean that a reactive agent is not an intelligent agent. Brooks has argued that the “real intelligence is situated in the world and intelligent behavior arises as the result of an agent’s interaction with its environment” [44, 45]. Agre also pointed out that “the most everyday activity is routine in the sense it requires little (if any) abstract reasoning” [46]. Many systems have been successfully developed using different reactive architectures such as Brooks’s *Subsumption Architecture* [44], Agre’s *Routine Architecture* [46], Mase’s *Agent Network Architecture (ANA)* [39] etc.

Hybrid Agent Architecture Many researchers have suggested that neither a completely deliberative architecture nor a completely reactive architecture is suitable for building agents [20]. Hybrid agent architecture is the one that integrates deliberative architecture and reactive architecture.

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

Hybrid architecture takes the advantages of deliberative and reactive architecture. In general, there are two essential components inside an agent with hybrid architecture: a planning or reasoning component to control the agent's long term goals and decisions, and a reaction component to quickly react to the environment changes. These two components actually can be considered as the representations of deliberative architecture and reactive architecture respectively. Hybrid architecture has been widely used in many agent systems today, and a number of such architectures have been designed such as *Procedural Reasoning System (PRS)* [47], *Touring Machines* [48], and *COSY* [49] and so on.

In this research, agents created from the proposed MADE implement hybrid agent architecture.

2.2 Goal Net

Goal Net Agent Development Methodology is based on *Goal Net Model* proposed by Zhiqi Shen [3]. It is a complete methodology for goal-oriented agent development which covers the system development cycle from initial requirement analysis to implementation. Goal Net is used as the mental model for agents in the proposed agent development environment.

2.2.1 Goal Net Model

Goal Net is a composite goal model which is composed of *states* and *transitions*. State, represented by a circle, is used to represent the state that an agent needs to go through in order to achieve its final goal. Transition, represented by an arc and a vertical bar, connects one state to another specifying the relationship between states it joins. Each transition must have at least one input state and one output state. It is associated with a task list which defines the possible tasks that the agent needs to

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

Figure 2.2: Goal Net sample

perform in order to transit from the input state to the output state. A simple Goal Net example is shown in Figure 2.2.

There are two types of states in Goal Net model, *atomic state* and *composite state*. An atomic state, represented by a blank circle, accommodates a single state which could not be split anymore; a composite state, represented by a shadowed circle, may be split into states (either composite or atomic) connected via transitions. In such a manner, a complex problem, which is supposed to be solved by the root goal, can be recursively decomposed into sub-goals and sub-goal-nets. Thus the system can be easily modeled and simplified.

In Goal Net, there are four types of temporal relations of goals represented by different transitions connecting the input states and the output states: *sequence*, *concurrency*, *choice* and *synchronization*. Sequence represents a direct sequential relationship between one input state and one output state. Concurrency has one input state but more than one output states, and all its output states can be achieved simultaneously. Choice specifies a selective connection from one input state to more than one possible output states, and only one output state can be selected and achieved by the

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

agent. Synchronization specifies a synchronization point from different input states to a single output state, and the output state can only be achieved when all its input states are synchronized. With different combinations of the basic temporal relationships, Goal Net supports a wide range of complicated relationships among goals. This is one of the major differences between Goal Net and other goal modeling method.

Goal Net is an agent model that supports not only behavior autonomy but also goal autonomy. In the short term, an agent is able to select and execute an appropriate action in order to achieve its next goal; in the long term, the agent can do a proper reasoning to select the best goal path to achieve its final goal. More than that, Goal Net does not specify a particular reasoning mechanism for agent's action selection and goal selection; instead, it gives the flexibility to Goal Net designers so that they can choose any suitable reasoning mechanism based on their application and system requirements.

2.2.2 Goal Net Agent Development Methodology

Goal Net Agent Development Methodology provides a complete guide for designing and developing agent systems using Goal Net model. It covers the full development cycle for agent systems, as shown in Figure 2.3.

Initial Analysis Initial Analysis is the first phase in Goal Net methodology. The objective of this phase is to derive a preliminary Goal Net by identifying goals (*what*), possible tasks for achieving the goals (*how*) and the environment that may affect how goals are pursued (*situation*). In this phase, Goal Net serves as a problem modeling and analysis tool. The given problem is modeled in the manner that what goals need to be achieved, what are the possible ways to achieve these goals and what are the relations among different goals. Following a knowledge-based top-down process, a

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

Figure 2.3: Goal Net Methodology overview

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

goal, which represents a complex problem, can be decomposed into a set of sub-goals, each of which represents a small problem. Such decomposition continues until all the goals can be easily achieved. After identifying the goal hierarchy, transitions and the task list associated with each transitions, can be added in to complete the Goal Net.

Architecture Design Architecture Design is to convert the Goal Net model into a multi-agent system. In another word, this is to identify different agents to achieve different goals according to some agent-goal binding policies, such as load balancing, organizational role, modularity etc. A Goal Net can be divided into a set of sub-goal-nets by splitting transitions, and these sub-goal-nets can be used as the goal model for a group of agents. These agents are organized in a hierarchy structure, called *Agent Hierarchy*. The agent, derived from the high level goals, becomes the coordinator of the agents derived from the low level goals. Each coordinator agent pursues its goal by coordinating the work of its coordinated agents. In such a manner, the root goal in the preliminary Goal Net becomes the common goal of the derived multi-agent system.

Details Design In the previous two phases, the skeleton of the agent system has been developed. In this phase, the system design goes into details, which includes the goal properties, task details, and agent communication protocols. The output of this phase is able to guide the actual implementation of the agent system which will happen in the next phase.

System Implementation In the implementation phase, all the detailed design and specifications from the previous phases, are mapped to the platform dependent concepts. This includes task development, communication handling, perception and reaction development etc. Goal Net does not specify any particular agent platform. Developers can choose any agent platform based on the system requirements.

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

Goal Net is a straightforward methodology for goal-oriented agent development. Goal Net model gives the convenience for agent developers to easily model the mental states of agents, which are represented by different goals. Transitions in Goal Net are used to describe the relations between different goals; task list associated with each transition, can be considered as the execution plan to pursue the desired goals. All these concepts have greatly simply the modeling process for agent design. However, similar to other agent development methodologies, Goal Net does not provide strong guide support on how to efficiently implement the design. Many issues in the implementation phase, such as how to make agents understand goals and how to make agents process according to the designed goal net, are not directly related to system logic, but they cost much time and effort for developers to program. Fortunately, different from other agent mental models, Goal Net manages agent's goals into a highly structural hierarchy. This gives the possibility that the mental state design can be automatically translated into agent implementations by some tools. That is the main reason that Goal Net is selected as the theoretical model for our proposed agent development environment.

2.3 Agent Development Frameworks and Tools

An agent development environment should provide an agent running environment, at the same time it should also provide a comprehensive guidance and assistance in agent design and implementation. In this section, several well established agent development environments are discussed.

2.3.1 JADE

Java Agent Development Framework (*JADE*) [2] is an open source agent development framework which is created by Telecom Italia Lab. It is fully developed in Java [50],

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

Figure 2.4: JADE platform overview [2]

and consists of an FIPA [51, 52] compliant agent platform and a package of tools.

JADE provides a configurable agent running platform. The platform serves as a middleware that hides the heterogeneity of different machines and allows JADE agents running across a computer network. The platform can be represented as Figure 2.4.

Within JADE platform, there are an *Agent Management System (AMS)* and a *Directory Facilitator (DF)*. AMS has the supervisory control over the platform and is responsible for maintaining a directory of *agent identifier (AID)* and agent state. Each agent must register with AMS when it is created. DF provides the yellow page service in the platform, so that an agent is able to find other agents in the same environment. The underline of JADE platform is the *Agent Message Transport System*, which provides channels for agents' communication. When JADE starts running, AMS and DF will start immediately in the main agent container.

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

Besides the platform, JADE also provides some tools for agent system implementation. A graphical user interface is provided for configuring the system running and controlling each agent's life cycle. Other tools, such as *Log Manager Agent*, *Socket-ProxyAgent* [53] etc., are also available to provide some utility services.

As agent systems are applied to not only computer but also mobile devices, *Lightweight Extensible Agent Platform (LEAP)* has been developed on top of JADE [54]. LEAP is an add-on to JADE, which allows an agent to run in mobile devices; Due to the computing power of mobile devices and the limitation of J2ME [55], the LEAP agents are made light-weight. However, LEAP provides the same development interfaces as JADE.

JADE and LEAP provide a good agent running platform. However, they do not provide much assistance and help in agent design except a collection of classes. Moreover, JADE provides very little support for the mental state design of agents. This makes JADE able to accommodate different agent models; on the other side, designing and coding work will be heavier for agent developers as the lack of direct support from the platform.

2.3.2 JACK

JACK is a commercialized agent platform developed by Agent Oriented Software Group. The name represents both the platform and the programming language used on the platform. The JACK language extends Java programming language [19], and the platform uses a component-based approach for building and running agents. The platform consists of some architecture-independent facilities, and it also provides a set of plugins that address the requirements of specific agent architectures.

As an agent platform, JACK provides an agent running environment, an agent development environment and an agent debugging environment. These give a complete

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

guide to developers for developing, running and maintaining their agent systems. Moreover, JACK also provides many useful plugins to simplify the development work. For example, JACK has BDI agent model plugin to support the BDI agent model, which can assist the BDI agent development. JACK also provides a SimpleTeam model [19] plugin, which can help developers to develop and manage a team of agents.

JACK can also be considered as a customized agent programming language, which extends from Java with a set of agent-oriented concepts, such as agent, plan, capabilities, resources etc. For the convenience of programmers with AI experience, it also supports logical variables. JACK compiler, which serves as a converter, is responsible for compiling JACK language code into pure Java class. This is very useful as developers can directly manipulate the high level concepts using these special syntaxes without worrying about how to implement them.

JACK is a comprehensive agent platform with customized agent programming language. However, it requires the developers to be properly trained. The introduction of agent oriented concepts into the language itself requires not only the agent designers but also the system programmers to have strong understanding of agent. Programmers have to be familiar with high level agent system design; at the same time, they also need sound knowledge of concurrent Java programming. Hence the platform loses some degree of user-friendliness because of its complexity and the heavy requirements on programmers.

2.3.3 ZEUS

ZEUS [56] is an open source agent development toolkit, which is developed by British Telecom Laboratory as part of the Midas and Agentcities research projects.

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

ZEUS is developed in Java programming language. It consists of three functional groups: an *agent component library*, a set of *agent building tools* and a suite of *utility agents*. The agent component library serves as an application interface to developers. It provides a collection of classes for agent construction, such as tools of knowledge manipulation, communication and planning and scheduling. The agent building tools have a set of graphical user interfaces to provide agent developers an agent construction environment. The agent building tools also provide a code generator, which is able to greatly simplify programmers' work by automatically generating individual agent program according to their specifications. The utility agents in ZEUS provides general utility services to the platform, and they provide functionalities to allow users to monitor and manage the agent system at runtime. These agents include name server, facilitator and visualizer.

The principle underlying the ZEUS toolkit is that "application-specific agents can be constructed by configuring the generic ZEUS agent, and equipping it with the necessary application functionality" [57, 58]. According to ZEUS documentation [56], there are several key steps in developing agents using ZEUS. The first step is *Domain Study*, which is to analyze the problem scenarios under the application domain and identify the candidate agents. Based on the domain study, each candidate agent should be defined with its planning capability including the tasks each agent may perform and the resources it can access. This step is called *Agent Definition*. The following step, namely *Task Specification*, is to work out the details of tasks defined before. The next two steps are *Agent Organization* and *Agent Coordination* respectively, which are to define some organizational relationship and interactions between agents. Once these steps are done successfully, agent building tools can be used to generate Java code automatically. Developers can also implement special abilities for their agents through the interface defined in ZEUS.

ZEUS provides a complete integration of different stages in software agent development, from analysis to deployment. However, as another complex software building tool, ZEUS is long and difficult to be mastered by software developers [59].

2.3.4 MADKIT

MADKIT stands for *Multi-Agent Development Kit*, and it is developed by a group of researchers led by Olivier Gutknecht and Jacques Ferber. MADKIT uses an organizational model to manage agents, which includes *agent*, *group* and *role* [60, 61]. Under such a model, an agent belongs to a group and it plays a certain role in this group. Each agent group may have multiple roles. MADKIT is developed in Java; however, MADKIT declares that developers are able to use other programming languages to develop agents on MADKIT, like Scheme and Jess.

MADKIT is a scalable and generic platform for agent development. It is designed to support and manage *agent heterogeneity*, *language heterogeneity* and *application heterogeneity* [62]. This is achieved by building an *agent micro-kernel* in MADKIT platform. The micro-kernel has three key functions: control of local groups and roles, agent life cycle management and local message passing. On top of this kernel, developers are allowed to develop different application agents by extending the MADKIT agent class. The agent class in MADKIT defined only the necessary agent life cycle management methods; hence developers have the choice to implement different agent models. Besides the application agents, MADKIT also use agents to provide system management functionalities like distributed message passing, migration control and dynamic securities.

MADKIT provides a highly flexible platform as supporting heterogeneity is one of the key characteristics; however, on the other side, this characteristic has made MADKIT

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

lack of direct agent development methodology support. It is not easy to develop a complex agent system without a comprehensive development methodology. Moreover, some researchers have pointed out that MADKIT is mostly a multi-agent runtime engine rather than a complete agent development platform as there is very limited support during system analysis and design phases [59]. Furthermore, developers still need to do a lot of coding in order to develop a complete system.

2.3.5 NetLogo, RePast and Swarm-based Agent Toolkits

Agent technology has been used for research on mass social behaviors, and there are several agents toolkits that focus on the such type of agent development. These toolkits facilitate the simulation of hundreds or thousands of independent agents, which is called swarm, to monitor and explore the mass behaviors in social network. Two of most famous such agent toolkits are NetLogo [63] and Repast [64].

NetLogo is created by Uri Wilensky in 1999. It has both an agent programming language and an agent modeling environment. The toolkit provides a large pre-defined agent model library so that users, even without programming knowledge, can easily use the toolkit for simulation. The user interface of NetLogo allows users to adjust the model parameters and view the agent activities based on the model. Netlogo is easy setup and use; however, the pre-defined model may not perfectly fit the user's application, and creation of the models requires good knowledge of programming.

Repast is another toolkits for agent-based simulation, and it is developed by Social Science Research Computing in University of Chicago. Repast was initially designed for swarm simulation. It provides a set of Java classes as the library for developing and running agents. The toolkit is very developer-friendly as the agent models can be easily extended and modified and it has been well supported by Repast community.

CHAPTER 2. THEORETICAL BACKGROUND AND RELATED WORKS

Both NetLogo and Repast are mainly used for agent-based simulation, and they focus more on the behaviors of a large group of agents instead of each individual agent. They emphasize the interactions and communications of agents; however, they do not provide strong support for the higher level of agent characteristics like goals and plans.

There are also some other agent development tools which may focus on specific types of agent development. For example, Microsoft Agent Platform [65] is mainly for user interface agent development. In this research focus on a generic agent development environment.

2.4 Summary

This chapter provides a brief discussion of the existing research in developing agent development environments. It is found that there is a gap between the current agent development methodologies and the current agent platforms and frameworks. Implementation of a real agent system requires heavy and tedious work for developers, and there is a real need for a comprehensive agent development environment to guide and simplify the agent development work.

Goal Net is a highly structured yet flexible model for goal-oriented agent system design. It is able to achieve both behavior-autonomy and goal-autonomy, and has provided an easy way to model a real system. Multi-Agent Development Environment (MADE), is proposed based this model. The proposed environment is able to provide complete environment for both designing Goal Net and creating and running agents based on the designed Goal Net. The detailed design of MADE is illustrated in the next chapter.

Chapter 3

Multi-Agent Development Environment (MADE)

Goal Net Methodology provides a theoretical guide of goal management in agent development. However, to develop a complete agent system based on Goal Net, there are still practical issues and problems which need to be studied. This chapter illustrates the details of how the concepts in Goal Net is realized and manipulated in the proposed MADE, and how MADE is able to facilitate and simplify the work in agent system development using Goal Net. The several key design concepts and development issues, such like Goal Net management mechanism and agent architecture, are discussed in details. An Evaluation of MADE, by comparing MADE with two well established agent frameworks according to several well-known criteria, is provided at the end of this chapter.

3.1 System Overview

MADE, which is developed in JAVA, is designed to be a comprehensive agent development environment. Since it uses Goal Net model as agent mental state design model, MADE can be considered as a complementary toolkit for Goal Net agent development methodology.

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

Figure 3.1: System overview of MADE

MADE focus on solving these two issues: how to assist agent developers to design a Goal Net easily and how to translate their design into system implementation automatically. The overall system architecture is illustrated in Figure 3.1.

In this architecture, there are three major modules: *Goal Net Designer*, *Goal Net Storage* and *Agent Creator*. Goal Net Designer provides a user-friendly graphical interface that allows designers to easily construct and edit Goal Net; Goal Net Storage, which connects with Goal Net Designer and Agent Creator, is a supporting database that is used to store and manage the designed Goal Net persistently; Agent Creator is able to create suitable agents and load the Goal Net into the agents for running.

Following Goal Net agent development methodology, once developers have worked out the system requirements, they can use Goal Net Designer to construct the desired Goal Net. After the Goal Net is constructed, it can be saved into the predefined database, Goal Net Storage. When creating agents, developers only need to tell Agent Creators the Goal Net that the agents need to process; Goal Net Creator will load these Goal Net from Goal Net Storage and create suitable agents to run the loaded Goal Net.

In designing the three modules, there are two key challenging issues: An effective Goal

Figure 3.2: Direct description of Goal Net model

Net description and management mechanism for Goal Net design and storage, and a suitable agent architecture for Goal Net processing. In the following two sections, these two issues are discussed in details respectively.

3.2 Goal Net Management

3.2.1 Goal Net Description Language

Goal Net is a rather structured and neat model, and it consists of a small number of components yet achieves high flexibility in designing agent mental state. To manage a Goal Net effectively, a suitable data structure is needed. This data structure should be in the form of description language or logical variables so that it can be understood and manipulated by agents, which are essentially computer programs.

There are two key components in a Goal Net, states and transitions; Hence, the most direct way to describe a Goal Net is to specify what states and transitions it has and how these states and transitions are related and connected. Figure 3.2 shows a sample Goal Net description in the above mentioned manner.

However, such kind of Goal Net description is neither sufficient nor efficient. First, such a descriptor is rigid and makes Goal Net lose its flexibility of decomposition and

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

```

Goal Net :
<GoalNet>
  <ID> identification of a particular goal net </ID>
  <root_State> ID of the root state</root_State>
  <start_State> ID of the start state</start_State>
  <end_State> ID of the root state</end_State>
  <properties> custom properties </properties>
</GoalNet>

State :
<State>
  <ID> identificaiton of a particular state </ID>
  <composite> whether a composite or a atomic state </composite>
  <goal_net_rooted> the goal net having this state as root </goal_net_rooted>
  <input_Transition> ID of an input transition </input_Transition>
  <output_Transition> ID of an output transition </output_Transition>
  <properties> custom properties </properties>
</State>

Transition :
<Transition>
  <ID> identification of a transition</ID>
  <type> transition type </type>
  <input_State> ID of an input state </input_State>
  <output_State> ID of an output state </output_State>
  <properties> custom properties </properties>
</Transition>

```

Figure 3.3: Key components of Goal Net Description Language

integration. As a Goal Net may contain several levels of decomposition and hence sub-goal-nets, such kind of description actually leads to information duplication and waste of resources, and this will further lead to the difficulty to decompose a goal or integrate simple Goal Net into a complex goal net. Second, even worse, it is inconvenient to describe complicate temporal relations, which may be achieved by a combination of synchronization transitions and choice transitions in Goal Net.

To overcome the above limitations, a Goal Net Description Language is proposed. This language is developed to facilitate the Goal Net Management in MADE with the following objectives:

- **Interpret Goal Net design** The language should be able to interpret the goal

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

net design, which is drawn in graphics, to logical format so that the contents of the design, instead of the drawing itself, can be easily understood by computer programs.

- **Store Goal Net design** Once a Goal Net is interpreted to logical format, it can be actually stored in any database or file. However, the language itself provides the ease of storing the design, the Goal Net management can be more effective and efficient.
- **Maintain Goal Net feature** Goal Net has many features such like reusing of goals and decomposing states. This has provided Goal Net some advantages than other agent models. The language itself should effectively maintain these important features so that it is not a complicated to interpret / store the Goal Net design when users apply these features.

With the above objectives, Goal Net Description Language has been designed in the form of Extensible Markup Language (XML) [66], which is used to store and manage data and can be easily processed. Figure 3.3 shows the key components of the proposed Goal Net Description Language.

There are three key tags in the language, and they are used to describe the three key components in Goal Net model: *Goal Net*, *State* and *Transition*. Each component has been assigned with an identification number (ID), which is automatically generated by MADE. Different properties of Goal Net components are described following the component ID, as shown in Figure 3.3. User can also provide custom properties for each Goal Net component, and the use of these custom properties are to be specifically designed by user in the later stage of agent details development.

The proposed Goal Net Description Language has the following features:

Figure 3.4: Illustration of loosely coupled Goal Net description

Each component in Goal Net is mapped to an XML entity which is described separately. This means that the details of states and transitions will not be given when describing a Goal Net; instead, only their identifications are given and the identifications can link to the details.

A goal net will be described in a loosely coupled manner. In other words, the Goal Net descriptor is only responsible for the description of the first level of decomposition, and all the sub-goal-nets in the goal net are described by other descriptors. Each component is self-described. Instead of giving all the information of states, transitions and their relations, a Goal Net descriptor only gives the root state, the start state and the end state. In the description of each state, the input transition and output transition are identified; similarly, the input states and output states are identified in the description of each transition.

In Goal Net, as Task does not affect the overall structure of Goal net and it is attached to a transition, in Goal Net Description Language, it is simply considered as a property of a transition.

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

```

<State>
  <ID> s1 </ID>
  <composite> yes </composite>
  <goal_net_rooted> g1 </goal_net_rooted>
  <input_Transition> null </input_Transition>
  <output_Transition> null </output_Transition>
</State>
<State>
  <ID> s2 </ID>
  <composite> no </composite>
  <goal_net_rooted> null </goal_net_rooted>
  <input_Transition> null </input_Transition>
  <output_Transition> t1 </output_Transition>
</State>
<State>
  <ID> s3 </ID>
  <composite> yes </composite>
  <goal_net_rooted> g2 </goal_net_rooted>
  <input_Transition> t1 </input_Transition>
  <output_Transition> t2 </output_Transition>
</State>
<State>
  <ID> s4 </ID>
  <composite> no </composite>
  <goal_net_rooted> g1 </goal_net_rooted>
  <input_Transition> t2 </input_Transition>
  <output_Transition> null </output_Transition>
</State>
<State>
  <ID> s5 </ID>
  <composite> no </composite>
  <goal_net_rooted> null </goal_net_rooted>
  <input_Transition> null </input_Transition>
  <output_Transition> t3 </output_Transition>
</State>
<State>
  <ID> s6 </ID>
  <composite> no </composite>
  <goal_net_rooted> null </goal_net_rooted>
  <input_Transition> t3 </input_Transition>
  <output_Transition> null </output_Transition>
</State>

```

Figure 3.5: Goal Net Description Language sample : state description

```

<GoalNet>
  <ID> g1 </ID>
  <root_State> s1 </root_State>
  <start_State> s2 </start_State>
  <end_State> s4 </end_State>
</GoalNet>
<GoalNet>
  <ID> g2 </ID>
  <root_State> s3 </root_State>
  <start_State> s5 </start_State>
  <end_State> s6 </end_State>
</GoalNet>

```

Figure 3.6: Goal Net Description Language sample : Goal Net description

```

<Transition>
  <ID> t1 </ID>
  <type> sequential </type>
  <input_State> s2 </input_State>
  <outputState> s3 </output_State>
</Transition>
<Transition>
  <ID> t2 </ID>
  <type> sequential </type>
  <input_State> s3 </input_State>
  <outputState> s4 </output_State>
</Transition>
<Transition>
  <ID> t3 </ID>
  <type> sequential </type>
  <input_State> s5 </input_State>
  <outputState> s6 </output_State>
</Transition>

```

Figure 3.7: Goal Net Description Language sample : transition description

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

Figure 3.4 shows a simple goal net which has two levels of decomposition. This Goal Net can be described as in Figure 3.5, Figure 3.6 and Figure 3.7.

As illustrated above, the independent description of each Goal Net component and the loose connection among states and transitions, allow easy maintenance and modification of a Goal Net without major change. The loosely coupled description of composite states allows easy decomposition and integration of Goal Net.

3.2.2 Goal Net Manipulation and Management

With a Goal Net description language, it is easy to describe and store Goal Net. However, to make this language understood by agents, which are essentially computer programs, this language still need to be processed and converted to certain computer programming language. In MADE, different Java classes are used to represent each component of Goal Net. This is used for agents to understand and manage the Goal Net structure.

Following the Goal Net Description Language, there are four classes to represent and process Goal Net : *GoalNet*, *State*, *Transition* and *Task*. Each class is mapped to a corresponding component in Goal Net.

GoalNet class Table 3.1 lists the properties of GoalNet Class. Similar to Goal Net Description Language, this class gives the meta information of a Goal Net. The *custom_properties* is an array used to store user-defined properties as the application needs.

State class Table 3.2 lists the basic properties of the abstract class State. As there are two types of states, two inherited classes, *AtomicState* and *CompositeState* are

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

Column Name	Data	Description
Goal_Net_ID	Number	unique identification of a Goal Net
Goal_Net_Name	Text	name of a Goal Net
Goal_Net_Description	Text	short description of a Goal Net
Start_State	Number	identification of the start state
End_State	Number	identification of the end state
Root_State	Number	identification of the root state
Custom_Properties	Array	array to store user defined properties

Table 3.1: Class GoalNet

Column Name	Data	Description
State_ID	Number	unique identification of a state
State_Name	Text	name of a state
State_Description	Text	short description of a state
Input_Transitions	Array	array to store input transitions
Output_Transitions	Array	array to store output transitions
Custom_Properties	Array	array to store user defined properties

Table 3.2: Class State

designed to represent atomic state and composite state respectively. In class CompositeState, a new property *Goal_Net_Rooted* is used to link to the Goal Net which has the current state as its root.

Transition class Class Transition is the internal presentation of transition in agent program. Similar to class State, Transition is an abstract class and has four inherited classes to manipulate the four different types of transitions. Figure 3.3 shows the properties of Transition class, and properties like *Input_States* and *Output_States* are implemented in each inherited classes.

Column Name	Data	Description
Transition_ID	Number	unique identification of a transition
Transition_Name	Text	name of a transition
Transition_Description	Text	short description of a transition
Tasks	Array	array to store tasks
Custom_Properties	Array	array to store user defined properties

Table 3.3: Class Transition

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

Column Name	Data	Description
Task_ID	Number	unique identification of a task
Task_Name	Text	name of a task
Task_Description	Text	short description of a task
Execution_Function	Text	the class name of a task
Custom_Properties	Array	array to store user defined properties

Table 3.4: Class Task

Task class Class Task is not representing the real tasks that are to be executed for firing transitions; instead, it functions as a symbol to represent task in Goal Net. As listed in Table 3.4, property *Execution_Function* links to the real function that is to be executed for firing transition. As an agent runs, it will call the execution function when it needs to fire the corresponding transition.

When constructing Goal Net model, appropriate objects are created from these four classes. The ID property of each object is the key identification and it must be given when creating new objects. Other properties, such as name and descriptions, can be set after object creation.

3.3 Goal-oriented Agent

3.3.1 Agent Architecture

A solid agent architecture is important to design and implement an agent system. In this research, we need to consider both how the agent architecture is able to realize the definition of agents (the *Weak Notion* in [20]) and how to make the architecture suitable to process Goal Net. In MADE, a hybrid agent [67] architecture is defined, and the overall architecture is illustrated in Figure 3.8.

This architecture has the following key components:

- *Data Unit*: This is a physical connection between an agent and the Goal Net data source and other persistent data source. It is responsible for loading Goal

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

Figure 3.8: Agent architecture [3]

Net from Goal Net storage before the agent starts running. The data source can be either in the form of Goal Net description language or a pre-defined database.

- *Process Unit:* Process Unit has two major functions. First, it works as a converter to understand the Goal Net description from Data Unit and convert different goal net components into the appropriate objects; Second, it serves as a runtime Goal Net storage when agent is running.
- *Knowledge Unit:* This unit interacts with a knowledge base and processes the knowledge level information such as agent's learning. This unit is not actually implemented in the current MADE.
- *Compute Unit:* This unit is the computing center to conduct the computation work for the agent. In processing Goal Net, Compute Unit is responsible for the reasoning for goal selection and action selection.
- *Perception Unit:* This unit is monitoring and perceiving the change of the agent's living environment.

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

- *Control Unit*: This unit is the brain of the agent. It processes Goal Net to achieve the final goal, coordinates other units and manages the overall work of the agent.
- *Action Unit*: This unit is used for task execution. It also reacts to environment change if necessary.
- *Communication Unit*: This unit handles the interaction between the agent and other agents. It controls the message sending and receiving.

In the implementation of this architecture, there are five threads running simultaneously. The first thread is taken by Perception Unit, which keeps monitoring the environment change and reports the critical events. The second thread is taken by Communication Unit, which listens to the incoming messages. The third thread is taken by Control Unit, the most important unit of an agent. The forth thread is used in Action Unit, in which tasks are executed. The last thread is used in Compute Unit. As some calculations in Compute Unit may take long time and occupy most of the computing resource, a separate thread for it will protect the agent from no-response situation when the computation load is heavy. Process Unit and Data Unit are passive units, which work only at Control Unit's request. So they will not take separate thread. Details of how this architecture works are illustrated in the following sections.

3.3.2 Goal Net Processing

When an agent starts running, it first loads a Goal Net from the database through Data Unit and Process Unit. Data Unit is a physical connection between an agent and a database. It queries the database or data source as the agent requests and returns the query results to the agent. Data Unit does not care what kind of data it is

Figure 3.9: Overview of Goal Net process

retrieving; instead, it only knows how to transform the agent's request into a proper data source query and send it to the data source driver.

Process Unit is a program responsible for reconstructing the goal net. As discussed earlier, Process Unit retrieves the information of different Goal Net components in the form of Goal Net description language and links them up to recover the Goal Net. These components are mapped to different objects, and the overall directory of states, transitions and tasks is maintained by Process Unit.

When an agent starts processing a Goal Net, there are mainly four units involved: Process Unit, Control Unit, Compute Unit and Action Unit. Figure 3.9 gives an overview of Goal Net processing.

The process can be summarized in the following steps:

- (i) Control Unit requires Process Unit for the next goals (there are more than one next goals available when there is a choice transition).
- (ii) Process Unit sends Control Unit all the possible next goals.

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

- (iii) Control Unit sends these goals to Compute Unit for goal selection.
- (iv) Compute Unit does a computation on these goals according to the user-designed goal selection algorithm, and sends the selected goal back to Control Unit.
- (v) Based on the selected goal, Control Unit asks Process Unit for the transition that can reach the selected goal.
- (vi) Process Unit sends Control Unit the corresponding transition.
- (vii) Control Unit sends the transition to Compute Unit for action selection.
- (viii) Compute Unit finds the best action to execute based on user-defined action selection criteria, and sends this action back to Control Unit.
- (ix) Control Unit sends the selected task to Action Unit for execution.
- (x) Action Unit executes the given task by dynamically invoking the task function, and sends acknowledgement to Control Unit once the task is done.
- (xi) Control Unit transits from its current goal to the goal it has just reached, and asks Process Unit for the next goals.

The above steps are enough for the process of sequential transitions and atomic states. When processing composite states and concurrent transitions, some extra steps are needed. There are two stacks and one queue in Control Unit, to handle composite states and concurrent transitions.

When the Control Unit encounters a composite state, the Control Unit will process the atomic states of the sub-goal-net whose root is the current composite state, as illustrated in Figure 3.10. The major steps are summarized as follows:

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

*Stack End & Stack Composite: process in a first-in-last-out manner

Figure 3.10: Composite State Process Flow

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

- (i) Control Unit gets a composite state S from Process Unit.
- (ii) Control Unit keeps the composite state S and asks Process Unit for the start state and the end state of state S .
- (iii) Process Unit sends back the start state $S1$ and the end state $S2$.
- (iv) Control Unit puts the composite state S to stack *Stack_Composite*, and put the end state $S2$ to stack *Stack_End*.
- (v) Control Unit processes the start state $S1$.
- (vi) Following the normal Goal Net processing steps, Control Unit moves on the processing until it reaches state $S2$. If Control Unit encounters another composite state before reaching state $S2$, step (1) to this step will be repeated.
- (vii) When Control Unit reaches state $S2$, $S2$ will be removed from *Stack_End*. This means the composite state S has been accomplished. Control Unit will remove state S from stack *Stack_Composite*.
- (viii) Control Unit asks Process Unit for the next states of state S .

When Control Unit encounters a concurrent transition, instead of creating more threads to process each routine in parallel, Control Unit will process each routine in a round-robin manner, as illustrated in Figure 3.11. This is to avoid overloading of the computing resources. For simplicity, we assume that the concurrent transition has three output states. The increase of the output state number will not affect the process mechanism. The process steps can be summarized as follows:

- (i) Control Unit fires an concurrent transition T , and reaches its output states $\{S1, S2, S3\}$

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

Figure 3.11: Concurrent Transition Process Flow

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

- (ii) Control Unit puts all the output states $\{S1, S2, S3\}$ into a queue *Queue_Concurrent*.
- (iii) Control Unit gets the first element of *Queue_Concurrent*, $S1$.
- (iv) Following the normal Goal Net processing steps, Control Unit processes state $S1$ and reaches the next state of $S1$ from Process Unit, $S4$.
- (v) Control Unit puts state $S4$ into *Queue_Concurrent*.
- (vi) If state $S4$ can be reached through an non-synchronization transition, following the normal Goal Net processing step, the transition is fired to reach the state $S4$. Once it is reached, state $S1$ will be removed from *Queue_Concurrent*.
- (vii) If state $S4$ can only be reached through a synchronization transition, Control Unit will check whether all the input states of this synchronization transition are available in *Queue_Concurrent*. If available, the transition will be fired and state $S4$ is reached, and all the input states will be removed from *Queue_Concurrent*; if not available, state $S1$ will be moved to the end of *Queue_Concurrent*.
- (viii) Control Unit repeats step (3) to step (7) until *Queue_Concurrent* only has one element, which means the all the concurrent routines have been synchronized to a single routine.

Figure 3.12 shows a simple Goal Net, which has two composite states and one concurrent transition. Following the steps illustrated earlier, Control Unit uses stacks and queues to control the composite state and concurrent routines during Goal Net processing. Figure 3.13 gives the status of the two stacks and one queues at different stage of goal net processing.

Figure 3.12: A sample Goal Net

Current Processing State : S7

S1	S4	S7
S3	S10	S6
Stack_Composite	Stack_End	Queue_Concurrent

Current Processing State : S9

S1	S4	S9
S3	S10	
Stack_Composite	Stack_End	Queue_Concurrent

Figure 3.13: Status of stacks and queues in Goal Net processing

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

In the process, Control Unit works as a middleman on other units. The centralized coordination allows developers to easily control an agent's running state. For example, the developers can easily suspend an agent by informing Control Unit to pause, and wake up the agent by resuming Control Unit. Once the ending state of the goal net is achieved, it is considered that the final goal is achieved. The agent will inform the user and enter a dummy state.

3.3.3 Perception and Reactions

Perception is the way that an agent feels its living environment and gets information from external world. Perception Unit is playing such a role in the proposed agent architecture. It maintains a list of targets, which are defined by agent developers, and keeps monitoring these targets. Some event handling mechanisms are also defined so that agents can handle specific events. There are two types of events: events that may affect the Goal Net processing and events that do not affect Goal Net processing.

For the events affecting Goal Net processing, they should be modeled as some specific states or user-defined properties of states so that agent can make proper decisions when it reaches these states. When the Goal Net reaches a certain state which need information from external environment, Control Unit will request for the information from Perception Unit.

For events that do not affect Goal Net processing, agent does not need complicate reasoning to handle it; instead, it can simply react to these events. In this proposed agent architecture, this is achieved by Perception Unit and Action Unit through a *Reaction Map*, as illustrated in Figure 3.14.

Reaction Map is a table that maintains a list of *situation-action* pairs. Each record in the table defines that under a given situation what action should be done to react to

Figure 3.14: Agent's perception and reaction

the situation. Generally all the situations defined in this map are emergent matters which need to be handled immediately. Reaction Map is maintained by Control Unit. When the information from Perception Unit has matched one of the defined situations in Reaction Map, Control Unit will send the corresponding action to Action Unit to execute immediately.

3.3.3.1 Reasoning Issues

As discussed earlier, Compute Unit is the decision unit for goal selection and action selection. Hence it is the essential key to achieve goal-autonomy and behavior-autonomy in the system implementation phase. In the above mentioned agent architecture, Compute Unit itself does not defines any reasoning mechanism for goal selection or action selection. Instead, it is an open interface which only defines what should be the optimal result under a given condition. The main reason to do so is that there are many kinds of reasoning mechanisms such as *case based reasoning* [68], *rule based reasoning* etc. Different applications have different requirements on reasoning. Hence developer should implement their own reasoning mechanism, or integrate the existing reasoning system to the agent, through Compute Unit. A goal selection algorithm from [38] has been implemented in MADE as default.

3.3.3.2 Communication

Communication is the way that different agents can interact with each other. In MADE, a central *Message Board* is used for message passing among agents. Message

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

Board is shared by all the agents. When an agent needs to send a message to the destination agent, it will post the message into Message Board. Similarly, an agent regularly checks Message Board for incoming message. If there is a message to it, it will take the message and delete it from Message Board.

Within the proposed agent architecture, messages are handled by Communication Unit. Communication Unit has three major components: *Message Writer*, *Message Receiver* and *Message Handler*. Message Writer is used to format the sending message properly, and post it on Message Board. Message Receiver is responsible for checking the incoming messages from Message Board. Message Handler, which is a interface in a dummy agent, is to make proper action and decision on how to handle a certain incoming message.

The communication mechanism is still at a prototyping stage, which can only be used for simple string message passing. Many factors such as security are not considered yet. Integration with a mature agent platform, like JADE, may provide a more reliable and complete communication for MADE agents.

3.3.3.3 Agent Coordination

More or less, agents are expected to work as a group or team. In Goal Net model, because of the hierarchy structure of states and goal-agent binding, an agent may possibly become the coordinator of other agents, and these agents are working for a common goal as a team. Hence, it is necessary to provide the basic agent coordination management functions for developers. For example, a coordinator agent may need one of its participant agents to start work; when a participant agent has achieved its goal, it may need to notify or report to its coordinator. The current MADE does not implement these team management functions yet. The detailed management mechanism can be explored in the future work.

3.4 MADE Work Flow

3.4.1 Goal Net Designer

To design a Goal Net, a user-friendly graphical interface allowing designers to directly draw out their design is highly needed. Goal Net Designer, a design tool in MADE, is to serve the purpose of easy Goal Net design.

Figure 3.15: Goal Net Designer

Figure 3.15 is a snapshot of Goal Net Designer. It is a comprehensive and user-friendly

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

tool for constructing goal net. As shown in the figure there are three windows and one toolbox in Goal Net Designer.

The main window in Goal Net Designer provides a canvas which allows designers to draw out their desired Goal Net. By selecting different tools in the toolbox, designers can easily create states and transitions and manipulate their relations through simple connections. In Goal Net Designer, the Goal Net drawn on the canvas is described using Goal Net Description Language. When designers create a new Goal Net component, either a state or a transition, the component ID is automatically generated; designers can edit other properties of the component and add different custom properties based on their application requirements.

The toolbox, locating above the main canvas, provides different components and tools for users to edit their design. There are three types of components in the toolbox: state, transition and arc, and each of them is used for designers to manipulate the corresponding Goal Net components in their design. Arc is not necessary for describing a Goal Net in program, but it is easier for users to edit the connection between a transition and a state.

When a user needs to add in a state in their design, he/she may click on the button “Add State”. A circle, which represents a state in Goal Net, is generated on the main canvas. The designer can drag the circle to any location of the canvas. If the designer wants to edit his/her design, for instance, change the type of a certain state from “atomic state” to “composite state”, he/she can simply select the target state on the canvas and click on the desirable radio box in the toolbox panel. The system will ask the designer to enter the start state and end state of this newly generated composite state, as shown in Figure 3.16. Once the designer has set the start state and end state

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

properly, the system will change the target state as required and the canvas will be refreshed to reflect the change.

The toolbox functions are also available in the popup menu of the main canvas window. When the mouse is right clicked in the canvas window, the popup menu will be shown and the available toolbox functions are listed in the menu. If the mouse is pointed to a certain component when it is right clicked, only the functions applicable to that component are shown.

Figure 3.16: Goal Net Designer

The upper-right window gives another view of the components in the canvas design. There are three trees in this window, which list all the states, transitions and arcs

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

respectively. This window is consistent with the design canvas. This means that whenever there is a change, by either adding a new state or deleting an existing transition, the tree will reflect the change synchronously. And also, when a component is selected in the tree, the corresponding component in the canvas design will also be selected. In such a way, this window gives designers the convenience to quickly identify a Goal Net component when they are designing a complex Goal Net.

The lower-right window is the property edition panel of Goal Net components. Designers can view and modify the property details through this panel. When a state or transition is selected in the design canvas or in the component trees, the properties of the selected state or transitions will be displayed in this panel. As you may notice in Figure 3.15, some properties of the component are pre-fixed by the system and are not allowed to be changed. For example, the component ID is generated by the system, and it should not be modified by designers. This is to prevent some mistakes from the incaution of designers.

When the Goal Net design is saved, there are two different copies of saving files. The first file saves the information of Goal Net itself, which is stored using Goal Net Description Language. This saved data is used for creating agent system, hence it does not include the information of graphical drawings of Goal Net. The second file saves the information of graphical drawings, which records the positions of each Goal Net components, so that Goal Net design can be retrieved exactly same as the one before saving when the saved file is reopened. The connections between different states or transitions, which represent the relations between these Goal Net components, are dynamically reconstructed according to their relations described in the first file.

As a design tool, Goal Net Designer also provides simple Goal Net design validation functions. For example, the system does not allow designers to use an “arc” to directly

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

connect two states or two transitions as this has broken the definitions in Goal Net model. A more systematic work on Goal Net model validation can be added into the Goal Net Designer in future study.

3.4.2 Agent Creator

Agent Creator is a function unit to create agents to process the designed Goal Net. The separation of agents and Goal Net models allows the Goal Net to act as the independent knowledge of an agent, so that an agent is able to equip different mental mind and function differently when different Goal Nets are loaded into it. Figure 3.17 gives an overview of Agent Creator.

Figure 3.17: Overview of Agent Creator

Agent Factory, as illustrated as the bottom layer of Agent Creator, is working as an agent manager and provides an agent running environment. Whenever developers want to create a new agent, it should request from Agent Factory instead of creating it manually. The agent running platform has two major functions: providing a communication platform for all agents, and providing basic agent management mechanism. All agents, which are created from Agent Factory, will be able to use the communication platform and are under management of Agent Factory.

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

In Agent Creator, most of the low level details, like Goal Net process and task executions, are automated and they are transparent to developers. When creating agents, developers only need to specify which Goal Net they want the agent to process. Then Agent Factory will create a dummy agent, register the agent and tell the agent where to retrieve its Goal Net. The agent will load the complete Goal Net and start running according to its Goal Net.

3.5 System Evaluation

MADE is a comprehensive agent development environment, as it provides both the mental state design and agent system implementation. To evaluate the its functionality and performance, a set of evaluation criteria are used to analyze it and compare it against other agent development platforms.

Four key criteria to evaluate agent development platform have been proposed in [59]: *Completeness*, *Applicability*, *Complexity* and *Reusability*. These four criteria are able to measure the major performance of agent development platform. However, they are rather general and can be used to evaluate any software development platform. To make the evaluation more specific to agent development platform, besides the above four, more criteria are used to make the evaluation more comprehensive and complete.

The following criteria are used in the evaluation:

- **Completeness** The degree of life cycle coverage for agent system development provided by the platform.
- **Applicability** The range of application domains offered by the development platform.

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

Criteria	MADE	JADE	JACK
Completeness	design & development	development	design & development
Applicability	goal-oriented agents	general	general
Complexity	low	medium	high
Reusability	flexible	medium	medium
Models Support	Goal Net	not specific	BDI...
Team Management	agents coordination*	not specified	<i>SimpleTeam</i>
Mobility Support	no	LEAP	no
Facility Services	no	directory service	general

* : future work

Table 3.5: Comparison of MADE, JADE and JACK

- **Complexity** The degree of easiness that a developer can use the given platform to develop and deploy agent systems.
- **Reusability** The possibility of reusing previous works.
- **Models Support** The mental models that the platform can directly support.
- **Team Management** The mechanism and degree of agent team work management provided by the platform.
- **Mobility Support** The degree of support for lightweight mobile agents provided by the agent platform.
- **Facility Services** The service that is used to facilitate agent systems.

To evaluate MADE, the above criteria are used to compare MADE with the well established agent development platforms: JADE and JACK. Table 3.5 shows the preliminary evaluation results.

From the comparison results, it is found that:

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

- MADE provides both Goal Net design and goal-oriented agent implementation support through the two core tools: Goal Net Designer and Agent Creator. JACK is a very comprehensive environment that provides a set of tools and interfaces in system design and development. JADE provides merely development support as it only provides a set of library classes for developer to use but no system design tools.
- MADE is particular used for developing goal-oriented agent systems, as Goal Net is the back end agent model. JADE does not have any explicit restriction on the application domains as it does not directly bind with any agent models; JACK is comprehensive and it can support both BDI agents and non-BDI agents.
- MADE is very user-friendly to developers as it simplifies developers' work by automatically translating their design into implementations. JADE is also easy to use but developers still need to do lots of coding works as no code generation function are provided. JACK requires developers to be properly trained and to have extensive knowledge of agents [69]; but once developers have mastered JACK, they will benefit from the great productivity of JACK.
- The reusability in MADE is achieved by the decomposition and integration of Goal Net model, which is highly flexible. For example, a complex Goal Net can be decomposed into several sub-goal-nets, and a sub-goal-net can be integrated and used in different composite Goal Net. Further more, the same tasks can also be used with different transitions in different Goal Nets. This decomposition and integration is easy to achieve in MADE as no extra coding is required. JADE and JACK provide moderate reusability during the agent development process through their modular design and through the object oriented programming.

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

- MADE uses Goal Net to model and create goal-oriented agents. JADE does not directly bind with any agent models. This gives JADE the flexibility to support different agent models; however, on the other side, developers need a lot of design and coding works to develop their agents as the lack of direct support from JADE. JACK is declared to develop different kinds of agents, and they have direct plugin support for BDI agents.
- MADE is designed to provide agents coordination mechanism. The coordination relationship is derived from the relationship between composite states and their sub-goal-nets. The general organizational relationship among agents, like agent teams, can also be explored in MADE. JADE does not provide any team management for agents except the ability to put agents in different agent containers. All the agent team management work has to be done by agent developers. JACK provides a very powerful agent team management system, named *SimpleTeam*. This is a distributed reasoning mechanism for agent team cooperations.
- The current version of MADE does not provide specific function support in mobile agent development. This can be extended in the future work. JACK currently does not provide the support for mobile agents either. JADE provides a great support for mobile agents by add-on work LEAP. The interface of LEAP is compatible with that of JADE, which provides great convenience for JADE developers to get familiar with LEAP.
- Currently MADE does not have much work on facility service for agents. JADE provides some general services like directory service. Distributed message passing is also an advantage of JADE. JACK provides several kinds of general services in agent running.

CHAPTER 3. MULTI-AGENT DEVELOPMENT ENVIRONMENT (MADE)

From the comparison, it can be concluded that MADE is competitive with these two widely recognized agent development platforms in several key aspects. On some aspects, like user-friendliness and reusability, MADE is even more advantageous comparing with JADE and JACK. On the other hand, MADE is focusing on goal-oriented agent system development, and it also can be improved and extended to provide more support and functionalities. This will be discussed in the last chapter.

3.6 Summary

In this chapter, the proposed agent development environment, MADE, is illustrated. MADE is comprehensive as it assists agent developers in both Goal Net design and system implementation for goal-oriented agents. Goal Net Designer provides an efficient and easy way to design Goal Net; while Agent Creator provides code generation function and handles most of the low level details for developers in system implementation. The Goal Net Description Language and Goal Net processing mechanism in the proposed agent architecture, provide translation from Goal Net design to agent implementation, which has effectively bridged the gap between agent design and system implementation. Hence, MADE releases agent developers from the tedious goal management and state process work, and allows them to focus on agent's mental state design and the agent task development.

Chapter 4

Developing an E-learning System with MADE

In this chapter, we discuss an agent based e-learning system, which is developed using Goal Net Methodology with the assistance of MADE. Here the main focus is to demonstrate how to use goal net and MADE in a real application instead of designing perfect e-learning system.

4.1 Introduction

With the fast growth of computer and communication technology, the traditional learning is replaced by the convenient e-learning. However, most of the existing e-learning systems are using the new technology to wrap the traditional learning materials and teaching styles, and they do not take the full advantages offered by the current technologies [70].

One of the major limitations is that many current e-learning systems are system-centric instead of learner-centric. The system adopts the same learning materials and strategies to different learners without considering their different profiles. The system is not able to take care of the diversity of learner populations. As a consequence, it is also not able to provide effective teaching services to part of the learner group. On

CHAPTER 4. DEVELOPING AN E-LEARNING SYSTEM WITH MADE

the learner side, learners may expect the teaching system to be flexible and the course to be customized to closely fit their background and needs. Thus there is a mismatch between the rigid services provided by the system and the flexible goals required by learners.

To fix this problem, a personalized e-learning system based on agent technology is proposed. The agent system is designed in a goal-oriented manner, which is modeled using Goal Net, to provide personalized e-learning service that can effectively match learners' goals. The system first analyzes a learner's current profile. After that it will generate a suitable learning path for the learner by choosing some specific courses, to adapt the learner's ability and preference. Finally the system will provide the assessment to test the learner and evaluate the learning process.

4.2 Motivation Scenario and Challenges

To make the illustration of the system development clearer, a typical learning scenario that motivates this research is used.

Now IT industry is placed at a state of fast changing due to the knowledge explosion. In order to stay competitive and deal with the dynamic market demands, companies need to frequently train their employees to acquire new knowledge.

Suppose a software company forms a project team for the newly tendered software project, and the project team members need to be trained through an e-learning grid service, to acquire the necessary knowledge and skills for the project. These team members have different background and will take different roles in the new project; thus the new skills they need to acquire and the technical levels of the skills that are required are possibly different. For each learner, a unique learning path needs to be

CHAPTER 4. DEVELOPING AN E-LEARNING SYSTEM WITH MADE

defined based on their learning goals. Further more, different learner may prefer to use different terminals such as laptop or PDA to access the learning materials.

To help the employee having a joyful and efficient learning experience, an agent system to provide a learner friendly e-learning system need to be developed. The system should be able to

- dynamically generate an on-going personalized learning path for employee based on his/her current skills level and the skills required by the project;
- refine the learning path according to the feedback from the learner in order to optimize the acquisition of the needed competencies;
- deliver the selected learning course to the learner site on his/her demand based on the learning path;
- provide self-learning cycle until the assessment of the learning results meet the requirements.

Besides the above explicit requirements, there are also some implicit requirements such as selecting the shortest learning path to reduce the learning cost. All the requirements will be carefully considered in the agent system design.

4.3 A Learning Goal Net

To build a learning system using Goal Net, as Goal Net Methodology suggests, we need to identify what goal need to be achieved, how to achieve these goals, and the situations.

CHAPTER 4. DEVELOPING AN E-LEARNING SYSTEM WITH MADE

For a learning system, the overall goal can be first identified, which is *E-learning Service*. After that, we may follow a system top-down approach, splitting this overall goal into a group of sub-goals, each of which represents a small problem.

- *Profile Analysis*: collect and analyze the profile of a learner, to assist generating an effective learning path for the learner;
- *Path Generation*: based on the learner's profile and his/her learning requirements, dynamically select the best learning courses for the learner. Courses can be represented as different goals such as "course Java programming learned";
- *Course Delivery*: negotiate with service providers in the server and deliver the selected course to the learner; customization is also provided according to learner's preference;
- *Assessment*: conduct assessment to a learner, and analyze the assessment results to provide necessary self-learning cycle for the learner.

The above list shows the first level of the designed Goal Net. These goals are still too abstract to achieve, so they can be further decomposed until that each goal represents a small problem which can be easily solved or achieved. The high level complex goal will automatically be achieved once all its sub-goals are achieved. Figure 4.1 shows a step further decomposition of the learning Goal Net, and Figure 4.2 shows the details of the pre-assessment sub-goal-net.

To explain how to decompose a composite goal, the goal Path Generation is used as an example. Learning Path Generation is a composite goal which is constructed dynamically for each learning application. All the necessary courses or learning objects are identified as goals in the Goal Net, and all the possible connections among the

CHAPTER 4. DEVELOPING AN E-LEARNING SYSTEM WITH MADE

Figure 4.1: High level states of the learning Goal Net

Figure 4.2: Pre-Assessment Goal Net

CHAPTER 4. DEVELOPING AN E-LEARNING SYSTEM WITH MADE

coursers, based on their pre-requirements relationship, are identified as transitions. Properties of a course, such as technical grade and cost, are used as the properties of the goal. The learner's profile is used as environment information for the learning Goal Net. Thus the Goal Selection Algorithm in Goal Net can be used to infer the best path to achieve the final goal. The learning path is dynamically generated during the learning process, which means that the next course is only selected or decided when the learner has finished the current course and he/she is ready for the next course, so that the learner's feedback of the previous course can be considered when selecting the next course. Such an on-going learning path generation can easily adapt the change of the learner's requirements.

After all the goals are identified, the next step is to define the transitions between goals, and work out the tasks to achieve each goal. Each transition may be associated with several tasks so that a suitable task can be selected and executed under different situations. For example, to collect a learner's profile, there are three possible ways: getting the learner's profile from the database of human resource, letting the learner fill up a technical survey, or conducting a test to get the learner's profile. Any of these three tasks can achieve the goal Profile Acquisition. The system may try to collect the learner's profile from human resource if it is available, otherwise it may try to let the learner to fill in the profile or conduct the test at learner's preference. Such kind of behavior-autonomy gives flexible choices to both the agent and the learners.

To complete the learning Goal Net model, the environment should also be modeled. In Goal Net, variable-value pairs are used to describe the environment. Some environment information has already been used in the previous task design. For example, the availability of the learner's profile in human resource can be considered as one environment variable. In this step all the environment information which may affect the goal pursuit or task execution are formally modeled.

CHAPTER 4. DEVELOPING AN E-LEARNING SYSTEM WITH MADE

Figure 4.3: General rules in Goal Net construction

Once all the preliminary goals are identified, MADE can be used to construct the Goal Net. In this design step, MADE has served as a design interface which allows us to easily draw out our design. The designed Goal Net, with all the states, transitions and conceptual tasks (the real task function may be implemented later) are maintained by MADE and can be used for later agent system development.

To construct Goal Net using the Goal Net Designer in MADE, developers can follow the general procedure as shown in Figure 4.3. First, we can add in all the states as

CHAPTER 4. DEVELOPING AN E-LEARNING SYSTEM WITH MADE

atomic states, and set each state's properties at the property edition panel. After this, we can set the composite state by setting their start state and end state. The system will automatically connect the composite state to its start state and end state using dash lines. The corresponding properties of these changed state, like the start state and end state, will also be automatically set accordingly. Then we can add in the transitions, which are represented by rectangles on the canvas, and set their basic properties. Finally, we can use arc to connect the proper transitions and states. We can also do some adjustments of the location of each component so that the drawing is fine to view. This adjustment can be done anytime, and it will not affect the connections from or to the adjusted components. The connections will follow the location change dynamically.

4.4 Agent-Mediated Learning System

The learning Goal Net we designed consists of tens of goals and a number of small tasks. Certainly it is possible that we use one single agent to pursue all the goals from profile collection to assessment analysis. However, it is not efficient to do so. The reason is that the pursuit of different goals requires different types of tasks to be executed; mixture of different levels of goals and different types of tasks makes the system difficult to be implemented and maintained. For example, the Path Generation goal requires heavy computation to generate a suitable learning path, while Course Delivery goal is more involved in communication and file transfer. The agent will be very heavy-weighted and error-prone if we use one single agent to process the entire learning Goal Net. It is reasonable that we use several agents and each of them processes part of the Goal Net.

To derive agents from the designed learning Goal Net, we follow the agent identification policies proposed in Goal Net Methodology. Based on the modularity and load

CHAPTER 4. DEVELOPING AN E-LEARNING SYSTEM WITH MADE

Figure 4.4: Deriving agents from the learning Goal Net

balancing policies, five agents from can be derived the learning Goal Net by splitting the goals in the first level, as shown in Figure 4.4.

As shown in Figure 4.4, E-learning Agent acts as a coordinator of other agents. It controls the general process of the e-learning system. Learner Profile Analyzer is pursuing the Profile Analysis goal by processing all the sub-goals and transitions of Profile Analysis goal. Once it achieves its goal, that is, it gets the profile analysis results, it will inform the coordinator agent which will further request Learning Path Generator to generate the next course or learning object for the learner. Learning Path Generator is used to find the next learning object the learner should take; this is actually to find the best next goal in its own Goal Net. After the learning object is figured out, Course Delivery Agent will start to negotiate and make deal with the service providers, customize the course format and deliver it to the learner. Assessment Agent will be informed after the course delivery. Once the learner has finished learning the course, Assessment Agent will automatically start to assess the learner. If the learner fails in the test, Assessment Agent will analyze the assessment results and provide the self-learning cycle until the learner passes the assessment test. When the learner has successfully learned one course (passed the assessment test), his/her

CHAPTER 4. DEVELOPING AN E-LEARNING SYSTEM WITH MADE

Figure 4.5: Structure of the learning system

profile will be updated and the suitable next course will be prepared and started. Once the Learning Path Generator has achieved its final goal, or there is no more next course, the whole learning process is considered accomplished.

In this step, MADE is able to created agents that load and process the user-defined Goal Net. This has greatly simplified our work in the implementation phase, as developers do not need to worry about the details of how these goals need to be processed and controlled. Only a little amount of work, such as agent coordination and message handling, needs to be coded for the agents created by MADE. In fact, the major work effort is put on task implementation, which can be used in any user preferred programming environments.

4.5 Summary

In this study case, a multi-agent system to provide highly learner-centric and personalized e-learning services has been designed. The system provides learning services to learners by generating a suitable learning path for the learner based on his/her profile. The e-learning system is designed based on Goal Net, which models the general

CHAPTER 4. DEVELOPING AN E-LEARNING SYSTEM WITH MADE

learning process and learner's requirements into different goals, and several agents are developed to pursue these goals. Besides a novel approach to provide learner centric e-learning system, it has also been found that MADE has been serving as an effective and efficient tool in the entire system design and development phase. From the very beginning of how to construct a preliminary learning Goal Net to the detailed system implementation, MADE has provided great assistance in designing Goal Net and constructing agents. Comparing to developing the system from scratch, MADE is serving as a good platform to simplify developers' work so that they are able to concentrate on how to design a learning Goal Net without worrying much about how to realize their design.

Chapter 5

An Agent-augmented Virtual World for Interactive Learning

Interactive learning is an emerging research area in education. Comparing with e-learning, interactive learning is more attractive to learners and it is more learner-centric. To develop such systems, two important problems need to be studied carefully. The first one is how to use an effective technology to provide interactive entities in the system. The second problem is how to manage the interactions between the system and learners to ensure that the desired learning goals are achieved by learners. Considering the natural characteristics of agents, it is believed that agent technology is a promising solution to these problems. This chapter explores why agents can be used for interactive learning and how Goal Net and the proposed MADE can be used to facilitate the interactive learning system development.

5.1 Introduction and Motivations

5.1.1 Agents for Interactive Learning

With the fast growing of information and computing technology, the life styles of human being has been greatly changed. One of the notable area that has been changed dramatically is the emergence of e-learning, which has been applied all around the world.

CHAPTER 5. AN AGENT-AUGMENTED VIRTUAL WORLD FOR INTERACTIVE LEARNING

However, it seems that most of the e-learning applications are deployed in the high education institutes like college and university, and very little work has been done for primary or secondary school students. Most of their course work is still textbook based. One of the possible reasons is that the conventional e-learning is not able to attract the students as they may feel it not more interesting than their textbook.

Interactive learning is a new learning paradigm emerging in recent years. It provides a rich environment and a dramatic learning story embodied with learning contents to educate learners. During the learning process, learners play some roles in the system, to explore the learning environment and interact with different artifacts in the learning environment. Through the interaction between learner and the environments, the learning story implicitly starts and continues, and the learners gain the learning contents.

Comparing with conventional e-learning, interactive learning puts learners into the heart of the learning system. Learners are not pushed by the learning system as they are free to discover the learning environment. Moreover, interactive learning is more attractive to learners because of the plenty of interactions between learners and the system.

Agent system is a good technical approach to design an interactive learning system. The natural characteristics of agent make it suitable to construct the interactive artifacts and model the system. This chapter illustrates how to use agents to build an interactive learning system.

5.1.2 Virtual Environment for Interactive Learning

To build an interactive learning system, a rich virtual environment, which allows learners to explore, is highly needed. This environment should not only provide

CHAPTER 5. AN AGENT-AUGMENTED VIRTUAL WORLD FOR INTERACTIVE LEARNING

the basic interactive interface between the system and the learners, but also provide functionalities to allow the system designers to control or program the interactions between the system and the learners. There are many software providing interactive environment but few is specialized for learning purpose. In this study, Active World is used as the virtual environment.

Active Worlds (AW) is a 3D virtual reality platform that allows multiple users to communicate, interact and explore [71]. Users, who login to the world through client software, are represented by avatars. Users can control their avatars to explore the world and talk with others. The virtual world, designed by graphical developers, has different objects and robots that can be programmed to interact with users. Figure 5.1 is a snapshot of a simple virtual world.

Figure 5.1: Active Worlds snapshot

Besides the general 3D virtual world features, Active Worlds has been customized

for education, which comes to *Active Worlds Educational Universe (AWEDU)*. This version adds in special features to allow teachers to move the whole classroom into the virtual world.

To develop an interactive learning system using AWEDU, there are two major tasks. The first one is to build the graphical virtual world. The platform allows developers to construct a world simply by inserting and editing objects from the graphical object library. The second task is to program the learning scenario through the provided *Standard Development Kits (SDK)*. This includes how to present the learning contents to learners, how to provide necessary guide to learners so that they will not lose in the world and how to assess learners etc.

To explore the possibility of applying agents for interactive learning, AWEDU is used as the learning environment. AWEDU will provide the virtual environment for users to explore; while agents will be responsible for deploying the learning scenario into the world and control the learning flows.

5.2 Learning Scenarios Design

In the selection of learning contents, *Health Science Education* for primary school students is chosen. In particular, dengue fever prevention [72] is used as the learning subject. The objective is to educate students about the symptoms, statistics, treatments, causes and prevention methods of Dengue Fever.

Dengue fever is used as our study topic because it is a prevalent disease in Singapore. This is an acute fever caused by a virus, which is carried by *Aedes* mosquitoes. It occurs in two forms: dengue fever (DF) and dengue hemorrhagic fever (DHF). Dengue fever is marked by the onset of sudden high fever, severe headache and pain behind

CHAPTER 5. AN AGENT-AUGMENTED VIRTUAL WORLD FOR INTERACTIVE LEARNING

the eyes, muscles and joints. Dengue hemorrhagic fever is a more severe form, in which bleeding and sometimes shock occur leading to death. It is most serious in children. Symptoms of bleeding usually occur after 3-5 days of fever.

Recently, there have been numerous outbreaks in Singapore. The government has been emphasizing that more forms of pest control should be directed against the mosquitoes. It is important to take control measures to eliminate the mosquitoes and their breeding places. So the project was designed to convey this dry but crucial information to students in a more interesting and interactive way.

In this scenario, a team of three students will conduct a dengue investigation in a virtual town, where some dengue cases have been found. There are three phases in the scenario: information collection phase, assessment phase and knowledge application phase. The details of each phase are explained as follows.

Information Collection Information collection is the first phase in the learning scenario. In this phase, the student team will collect necessary information about dengue from different locations. There are three key locations defined for major information collection: hospital, health ministry and residential area. In hospital, through the conversation with doctors, nurses and patients, students will get to know the symptoms and treatments of dengue fever. In health ministry, the officers will provide the prevention information of dengue to students, like how dengue is spread and how to eliminate the breeding places of mosquitoes. In the residential area, students will explore and get the feeling of what are the environment in which dengue is easily spreading. Through the visit of these three locations, students will be able to gain the knowledge of dengue fever.

Assessment After information collection, there will be a simple test for students.

CHAPTER 5. AN AGENT-AUGMENTED VIRTUAL WORLD FOR INTERACTIVE LEARNING

This is to ensure that the students have gained the necessary information. As students can explore the town without any restriction, it is possible that they miss some information collection points. The assessment will examine the knowledge students have acquired and ensure that they do not miss any important learning point. If the students are not able pass the test, which means that they have missed some important learning points, the system will recommend the students to go to the appropriate locations to collect the necessary information.

Knowledge Application Assessment is the phase to examine whether the students have collected all the necessary information; knowledge application is a step further, to examine whether the students understand the information and achieve the learning objectives. In this phase, the students are sent to a residential area with high dengue infection rate. They need to apply what they have gained in the first phase, to make proper changes or adopt suitable measures to improve the dengue situation in that area. For example, they may need to remove stagnant water around the houses to eliminate the breeding of mosquitoes, or they may need to kill mosquitoes by spraying insecticide. Whenever a proper precaution is adopted by the students, the system will reward them by one mark. Students will finish this phase when they get six marks.

The above three phases provide a complete learning scenario. As described in the details, the investigation can only be done through different interactions between students and the system robots, and the test can only be passed through the proper manipulation of different system artifacts. The learning is done through the exploration and discovery. In the system development, different agents are used to manage the scenario flows and interact with students.

5.3 Learning Goal Net Design

To develop agent system using Goal Net methodology, the first step is to construct the system Goal Net based on the system requirements. Figure 5.2 shows an overview of the Goal Net for the desired interactive learning system.

Figure 5.2: Overview of Goal Net for interactive learning

This Goal Net is designed according to the three learning phases. Each state in the Goal Net represents one phase in the learning scenario. These three states are abstract and conceptual; hence they should be further decomposed into sub-goal-nets to guide the system development.

Figure 5.3 shows the decomposition of state Information Collection. This sub-goal-net describes states that the system needs to go through in order to achieve the final goal.

When students login the learning world, the system will first greet them to welcome them. After that, the system will briefly explain the learning scenario to the students, to ensure that they have a basic idea of the learning subject. If the students do not understand the scenario, the system will explain the scenario again. Once the students inform the system that they have understood the learning scenario, the system will

CHAPTER 5. AN AGENT-AUGMENTED VIRTUAL WORLD FOR INTERACTIVE LEARNING

Figure 5.3: State decomposition of Information Collection

start to initialize the three scenes for students to explore. When they are successfully initialized, a teleport will be opened to the students so that they can visit the three places. After this step, the system, at this level, will become reactive to wait for the students' coming back. If the system gets informed that the students are ready for assessment, the system will initialize the assessment agent and open the teleport to assessment.

Transitions and tasks are the keys to move the Goal Net from state to state. Table 5.1 shows a summary of the transitions and tasks in Information Collection sub-goal-net. There are some dummy tasks in the table. This means that the system does not need to execute any explicit task; instead, it will wait for the response from students or the transition does not need any task to be executed.

Similar to state Information Collection, the rest two states in the first level of the

CHAPTER 5. AN AGENT-AUGMENTED VIRTUAL WORLD FOR INTERACTIVE LEARNING

Transition	Task Description
5	greet the students when they login the system
6	describe the scenario to the students
7	check whether the students understand the scenario description
8	describe the scenario to the students
9	dummy
10	activate the necessary agents for the hospital scene
11	activate the necessary agents for the health ministry scene
12	activate the necessary agents for the residential are scene
13	dummy
14	open the information collection teleport
15	activate the assessment agent for assessment
16	dummy
17	open the assessment teleport
19	dummy

Table 5.1: Transitions & Tasks for Information Collection

overall Goal Net, Assessment and Knowledge Application, can be further decomposed into different sub-goal-nets. After decomposing all the composite states into atomic states, the Goal Net hierarchy is completed. Several agents have been identified based on the Goal Net design.

- *Instructor Agent.* This agent is the manager of the learning scenario. It monitors and controls the learning flows of the system. An avatar is controlled by this agent in the learning world to give some instruction to students.
- *Assessment Agent.* This agent is responsible for assessing the student team to ensure that they do not miss any important information during their exploration. No avatar in the learning world is representing this agent.
- *Application Test Agent.* This agent is designed to examine the students in the knowledge application phase. No avatar in the learning world is representing this agent.

CHAPTER 5. AN AGENT-AUGMENTED VIRTUAL WORLD FOR INTERACTIVE LEARNING

- *Doctor Agent.* This agent controls an avatar to play the role of a doctor in the hospital. It will directly interact with nurse agent, patient agent and learners. It is also responsible for explaining the symptoms and treatments of dengue to students.
- *Nurse Agent.* Similar to doctor agent, it plays the nurse role in the hospital; however, comparing with doctor agent, it is reactive and will not initialize communication with students.
- *Patient Agent.* This agent plays as a dengue fever patient in the hospital. It will explain its feeling of dengue fever to students when it is asked.
- *Official Agent.* This agent acts as an official in health ministry. It explains to students about how dengue is spread.
- *Villager Agent.* This agent controls an village avatar in the residential area to help students get more information about the dengue spreading situations in that area.

Different agents have different contributions for the learning system. In the above list, first three agents are at management level; they may be not directly visible or interact with students, but they control the learning flows. The rest of the agents in the list are at executive level; they are controlling avatars in the world and directly interact with students. However, as they do not have significant effect on the system, they can be independently modified and changed.

In this step, MADE is used to design and manage the overall Goal Net. In fact, MADE does not have a constraints on the number of Goal Net designed in each canvas; this means that designers can design more than one Goal Net on a single design file. This

allows us to easily decompose our overall Goal Net to sub-goal-nets that can be loaded to different agents.

5.4 Learning World Construction

To play the learning scenario, a learning world is needed. River City, a small virtual town, is built on AWEDU platform. In this town, there are a hospital, a health ministry and a residential area. Figure 5.4 gives two snapshots of River City.

Figure 5.4: Snapshots of River City

To make the world interactive and interesting, many objects are designed manipulable to learners. For example, learners can push the door to open or close it; learners can cover or uncover the mosquito net for a baby bed. Moreover, some places are designed to be environment-aware. For example, when a learner approaches close to a dirty water container, he/she may hear the mosquito sound. This implies that this is a possible place of mosquito breeding.

There are also some static citizens in River City. Comparing with the avatars controlled by agents, the citizens built here are rather static and they are not able to initialize conversations with students. However, they can answer some pre-defined question to give support information of the learning scenario.

5.5 System Development

Once the learning world is built and the Goal Net is designed, the system can be implemented using MADE. The main task in this phase is to create agents to control the learning world and scenario flows. Figure 5.5 shows the overall architecture of the River City learning system.

Figure 5.5: System architecture of River City

In this architecture, agent system is running within MADE but it plays as the brain of the system. As mentioned in the earlier section, agent system monitors the system and controls the learning flows. This has been designed using Goal Net, thus agents will simply run the corresponding Goal Net to achieve the system objectives.

Java Native Interface (JNI) [73], is a mediate interface in the architecture to facilitate the program. MADE agents are running in Java, while Active Worlds SDK is a programming interface in C. Thus JNI is used as a bridging interface to overcome the language gap. Through JNI, agents are able to perceive events from River City and execute their tasks on the system.

Active World SDK, provided with the platform, is a set of application program interfaces to configure and control Active World platform. Comparing with the agent

CHAPTER 5. AN AGENT-AUGMENTED VIRTUAL WORLD FOR INTERACTIVE LEARNING

system, SDK is working at lower level. It is responsible for translating the agent's tasks into execution details on the system platform; it is also responsible for passing system events to agents so that agents can handle their interested events. SDK itself does not make any decision in the aspect of learning flow control.

AWEDU platform provides the running environment of the 3D River City. It executes the commands from the SDK to change and affect the learning world; it also generates different events to reflect users' status and catch users' actions. From the architecture, the system itself is essentially controlled by the agents; the rest of the layers serving as facilitators of the system.

To implement the system, the conceptual tasks designed in the earlier section need to be implemented. This can be done through any user preferred programming environment. Agents will execute the appropriate tasks when processing Goal Net. Once all the tasks are implemented, the next step is to implement the message handling inside each agent, which defines what one agent will respond when it receives a message. The message handling will directly affect the manner of agent cooperation. Once all these work has been done, MADE can be used to create different agents, and load the appropriate Goal Net to each agent. The system will run as each agent pursues its own goal according to the Goal Net given to it.

5.6 Summary

In this chapter, the study of using Goal Net and MADE for interactive learning system development is illustrated. Besides the rich graphical environment, interactions and learning scenario control are the keys for a successful interactive learning system. Agent's natural characteristics have made agent technology suitable for solving these issues and realizing the system. In this case study, different agents have been

CHAPTER 5. AN AGENT-AUGMENTED VIRTUAL WORLD FOR INTERACTIVE LEARNING

developed using MADE to provide interactions and handle learning scenarios. These agents, besides providing individual functionalities, need to work as a team to ensure that the learning goals are achieved. Goal Net and MADE have provided great ease in designing and implementing these agents, in both their individual goals and the whole system goals. The composite Goal Net is able to effectively model the system goals and agent teams, and MADE is able to successfully construct the agents and to ensure their assigned goals are achieved. Many trivial details are handled automatically by MADE, and designers's effort can be focused on the learning scenario design and development.

Chapter 6

Conclusion and Future Work

This chapter concludes this thesis, summarize the contributions of this research and discuss the future work of the MADE system.

6.1 Conclusion

Agent is increasingly recognized as an promising approach for future software development. The autonomous, goal-oriented and intelligent characteristics make agents attractive for solving problems in complex and dynamic software. Large effort has been put in agent research in the past decade. However, majority of the research work has been put in the agent mental models and agent classifications, and very little work has been put in practical agent development. The gap between agent state modeling and agent system implementation has been identified and emphasized by many agent researchers recently.

To bridge the agent modeling and agent implementation gap, in this research we present Multi-Agent Development Environment, MADE. MADE is a comprehensive tool that can effectively connect the agent model design with agent implementation. In MADE, Goal Net is used as the model for agent mental state design. The Goal Net model is described in Goal Net Description Language, and the model design can

CHAPTER 6. CONCLUSION AND FUTURE WORK

be completed using Goal Net Designer in MADE. As a development tool, MADE also provides automatic agent creation, which has greatly simplified developers' work in realizing their design. Through different case studies, MADE has been demonstrated as an efficient and user-friendly agent design and development environment.

The main contributions of this research include:

- A summary of the state-of-art agent modeling and development, especially on the goal-oriented agent modeling and the established agent development tools.
- A comprehensive goal-oriented agent development environment, MADE. MADE provides both agent mental design and agent creation. Based on user's Goal Net design, MADE is able to automatically create agents with the designed Goal Net, hence it effectively bridges the agent mental design and agent implementation. Comparing with the existing work, MADE directly supports a goal-oriented agent development: Goal Net Agent Development Methodology, thus it is much easier to develop an goal-oriented agent system using MADE.
- A Goal Net Description Language. This XML-based language describes a Goal Net in a highly structured format. Different components in Goal Net are described separately, and the relations between transitions and states are described in a loosely coupled manner. In such away, the flexibility and reusability of Goal Net are maintained, and the described Goal Net can be easily decomposed or integrated without major change of the description language.
- An agent-mediated e-learning system. In this case study, we have developed a learner-centric e-learning system. The goal selection and action selection play vital roles in achieving the learning flexibility. The results of the case study proved that MADE is able to simplify the design and implementation work significantly.

CHAPTER 6. CONCLUSION AND FUTURE WORK

- An interactive learning system. This system demonstrates an new approach to develop interactive learning system by applying goal-oriented agents in the system. Comparing with traditional approaches, agents' natural characteristics make the system more efficient in the learning scenario control and the interactions with learners.

6.2 Future Work

Despite the reported contributions stated above, the current work can be extended in at least the following dimensions:

- **Goal Net Model Validation** In the Goal Net design, developers may sometimes mistakenly add in a state but forget to build up the connection between this state and other states or transitions. It is also possible that developers build a deadlock in a Goal Net, which means two or more states form a iteration loop and the processing agent can not transit to other states from these states. All these kinds of design are invalid Goal Net, and they will cause that either some states are never achievable or the agent stops pursuing its next goal. When a simple Goal Net is designed, it is not difficult to figure out, from the agent's perspective, whether the Goal Net is a valid model. However, if the Goal Net is rather complicate, it may need quite some effort to check whether the designed Goal Net can be correctly processed by an agent if the check is done manually. Hence, a Goal Net validation tool is necessary in Goal Net design. This tool can find out the technical errors in the design to avoid the incorrect Goal Net being loaded into agents.
- **Agent Coordination** One of the most of important features of Goal Net is that composite states can be decomposed into sub-goal-nets. When deriving

CHAPTER 6. CONCLUSION AND FUTURE WORK

agents from a Goal Net, many times developers may follow the decompositions in Goal Net. Hence naturally the agents derived from high level composite states become the coordinators of the agents from low level states. Such kind of coordination is simple, but they are frequently used. It is meaningful to add some automated coordination control mechanisms into MADE, so that the agents with the coordination relations can be easily managed by developers during their implementation.

- **Light-weighted Agent Development** Due to the fast growth of mobile computing power, many light-weighted agents have been put into the mobile devices. These agents have been deployed and used in many application domains like personal assistant and directory service. It is meaningful to explore the possibility of using MADE to design and develop light-weighted agents.
- **More MADE Applications** One of the important source on what and how an agent development environment should be improved is the feedback of its applications. In this research, two applications have been developed using MADE. However, the applications are not complicated and many agent features like teamwork and reasoning are not be able to tested practically. Hence, one of the important future work is to develop more and rather complicated applications using MADE, to justify and continue improving the functionalities and effectiveness of the toolkit.

References

- [1] H. S.Nwana, "Software agents: An overview," *Knowledge Engineering Review*, vol. 11, no. 3, pp. 205–244, 1996.
- [2] "JADE." <http://jade.tilab.com/>.
- [3] Z. Shen, C. Miao, X. Tao, and R. K. L. Gay, *Goal Oriented Modeling for Intelligent Software Agents*. PhD thesis, 2004.
- [4] N. R.Jennings and M. Wooldridge, *Agent Technology: Foundations, Applications, and Markets*. Springer, 1997.
- [5] H. S.Nwana and D. T.Ndumu, "A perspective on software agent research," *Knowledge Engineering Review*, vol. 14, no. 2, pp. 125–142, 1999.
- [6] I. T. Foster, N. R. Jennings, and C. Kesselman, "Brain meets brawn: Why grid and agents need each other.," in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 8–15, 2004.
- [7] D. Kotz and R. S. Gray, "Mobile agents and the future of the internet.," *Operating Systems Review*, vol. 33, no. 3, pp. 7–13, 1999.
- [8] N. R. Jennings, "On agent-based software engineering.," *Artificial Intelligence*, vol. 117, no. 2, pp. 277–296, 2000.
- [9] F. Zambonelli, N. R. Jennings, and M. Wooldridge, "Developing multiagent systems: The Gaia methodology.," *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 3, pp. 317–370, 2003.

REFERENCES

- [10] M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia methodology for agent-oriented analysis and design.," vol. 3, pp. 285–312, 2000.
- [11] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini, "Tropos: An agent-oriented software development methodology," Tech. Rep. DIT-02-0015, University of Trento, 2002.
- [12] F. Giunchiglia, J. Mylopoulos, and A. Perini, "The Tropos software development methodology: Processes, models and diagrams.," in *Proceedings of the 3rd International workshop on Agent-Oriented Software Engineering*, pp. 162–173, 2002.
- [13] S. A. DeLoach, "Multiagent systems engineering: a methodology and language for designing agent systems," in *Agent Oriented Information Systems '99*, (Seattle), pp. 45–57, 1999.
- [14] M. F. Wood and S. A. DeLoach, "An overview of the multiagent systems engineering methodology.," in *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*, pp. 207–222, 2000.
- [15] J. Odell, H. V. D. Parunak, and B. Bauer, "Extending UML for agents," in *Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, pp. 3–17, 2000.
- [16] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI-Architecture.," in *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pp. 473–484, 1991.
- [17] A. S. Rao and M. P. Georgeff, "BDI agents: From theory to practice.," in *Proceedings of the 1st International Conference on Multiagent Systems*, pp. 312–319, 1995.
- [18] C. A. Iglesias, M. Garijo, and J. Centeno-González, "A survey of agent-oriented methodologies.," in *Proceedings of the 5th International Workshop on Agent Theories, Architectures, and Languages*, pp. 317–330, 1998.

REFERENCES

- [19] N. Howden, R. Rönquist, A. Hodgson, and A. Lucas, "JACK - summary of agent infrastructure," in *Proceedings of the 5th International Conference on Autonomous Agents*, 2001.
- [20] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowledge Engineering Review*, vol. 10, no. 2, 1995.
- [21] A. D. Stefano and C. Santoro, "Netchaser: Agent support for personal mobility," *IEEE Internet Computing*, vol. 4, no. 2, pp. 74–79, 2000.
- [22] P. Wallis, R. Rönquist, D. Jarvis, and A. Lucas, "The automated wingman - using JACK intelligent agents for unmanned autonomous vehicles," in *Proceedings of the IEEE Aerospace Conference*, vol. 5, pp. 2615–2622, 2002.
- [23] D. C. Smith, A. Cypher, and J. C. Spohrer, "Kidsim: Programming agents without a programming language.," *Communications of the ACM*, vol. 37, no. 7, pp. 54–67, 1994.
- [24] T. Selker, "Coach: A teaching agent that learns.," *Communnications of the ACM*, vol. 37, no. 7, pp. 92–99, 1994.
- [25] Y. Shoham, "Agent-oriented programming.," *Artifical Intelligence*, vol. 60, no. 1, pp. 51–92, 1993.
- [26] J. Odell, "Objects and agents compared.," *Journal of Object Technology*, vol. 1, no. 1, pp. 41–53, 2002.
- [27] M. Luck and M. d'Inverno, "A conceptual framework for agent definition and development.," *Computer Journal*, vol. 44, no. 1, pp. 1–20, 2001.
- [28] R. A. Kowalski and F. Sadri, "Towards a unified agent architecture that combines rationality with reactivity.," in *Proceedings of the international workshop on Logic in Databases*, pp. 137–149, 1996.
- [29] M. R. Genesereth and S. P. Ketchpel, "Software agents.," *Communications of the ACM*, vol. 37, no. 7, pp. 48–53, 147, 1994.

REFERENCES

- [30] "FIPA ACL Message Structure Specification."
<http://www.fipa.org/specs/fipa00061/>.
- [31] J. Odell, M. H. Nodine, and R. Levy, "A metamodel for agents, roles, and groups.," in *Proceedings of the 5th international workshop on Agent-Oriented Software Engineering*, pp. 78–92, 2004.
- [32] L. Gasser, "Perspectives on organizations in multi-agent systems," in *Mutli-agents systems and applications*, pp. 1–16, Springer-Verlag New York, Inc., 2001.
- [33] N. R. Jennings, K. P. Sycara, and M. Wooldridge, "A roadmap of agent research and development.," *Autonomous Agents and Multi-Agent Systems*, vol. 1, no. 1, pp. 7–38, 1998.
- [34] E. Alonso, M. D’Inverno, D. Kudenko, M. Luck, and J. Noble, "Learning in multi-agent systems," *Knowledge Engineering Review*, vol. 16, no. 3, pp. 277–284, 2001.
- [35] M. Kolp, P. Giorgini, and J. Mylopoulos, "A goal-based organizational perspective on multi-agent architectures.," in *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages*, pp. 128–140, 2001.
- [36] M. Bonifacio, P. Bouquet, R. Ferrario, and D. Ponte, "Rationality, autonomy and coordination: The sunk costs perspective.," in *Proceedings of Engineering Societies in the Agent World*, pp. 29–38, 2002.
- [37] C. Cristiano, "Guarantees for autonomy in cognitive agent architecture," in *Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents*, pp. 56–70, 1995.
- [38] Z. Shen, C. Miao, X. Tao, and R. Gay, "Goal oriented modeling for intelligent software agents," in *Proceedings of International Conference on Intelligent Agent Technology*, pp. 540–543, 2004.
- [39] P. Maes, "The agent network architecture (ANA).," *SIGART Bulletin*, vol. 2, no. 4, pp. 115–120, 1991.

REFERENCES

- [40] M. E. Bratman, D. Israel, and M. E. Pollack, "Plans and resource-bounded practical reasoning.," *Computational Intelligence*, vol. 4, pp. 349–355, 1988.
- [41] S. A. Vere and T. W. Bickmore, "A basic agent.," *Computational Intelligence*, vol. 6, pp. 41–60, 1990.
- [42] N. R. Jennings, "Specification and implementation of a belief desire-joint-intention architecture for cooperative problem solving.," *International Journal of Cooperative Information Systems*, vol. 2, no. 3, pp. 289–318, 1993.
- [43] J. Ferber, "Simulating with reactive agents," in *Many Agent Simulation and Artificial Life* (E. Hillebrand and J. Stender, eds.), pp. 8–28, Amsterdam: IOS Press, 1994.
- [44] R. A. Brooks, "Intelligence without reason," in *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pp. 569–595, 1991.
- [45] R. A. Brooks, "Intelligence without representation.," *Artificial Intelligence*, vol. 47, no. 1-3, pp. 139–159, 1991.
- [46] D. Chapman and P. E. Agre, "Abstract reasoning as emergent for concrete activity," in *Workshop on Planning and Reasoning about Action*, pp. 251–266, 1986.
- [47] M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning.," in *Proceedings of the 6th National Conference on Artificial Intelligence*, pp. 677–682, 1987.
- [48] I. A. Ferguson, *Touring Machines: An Architecture for Dynamic, Rational and Mobile Agents*. PhD thesis, University of Cambridge, UK, 1992.
- [49] B. Burmeister and K. Sundermeyer, "Cooperative problem-solving guided by intentions and perception (abstract).," *SIGOIS Bull.*, vol. 13, no. 3, p. 10, 1992.
- [50] "JAVA." <http://www.java.sun.com>.
- [51] "FIPA Agent Management Specification." <http://www.fipa.org/specs/fipa00023/>.

REFERENCES

- [52] “FIPA Abstract Architecture Specification.” <http://www.fipa.org/specs/fipa00001/>.
- [53] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa, “JADE programmer’s guide,” tech. rep., 2005.
- [54] G. Caire, “Leap user guide,” *JADE online tutorial*, 2005.
- [55] “J2ME.” <http://www.java.sun.com/j2me>.
- [56] H. S. Nwana, D. T. Ndumu, L. C. Lee, and J. C. Collis, “Zeus: A toolkit for building distributed multiagent systems,” *Applied Artificial Intelligence*, vol. 13, no. 1-2, pp. 129–185, 1999.
- [57] J. Collis, D. Ndumu, and C. v. Buskirk, “The zeus technical manual,” 2000.
- [58] “ZEUS.” <http://labs.bt.com/projects/agents/zeus/>.
- [59] P.-M. Ricordel and Y. Demazeau, “From analysis to deployment: A multi-agent platform survey,” in *Proceedings of the 1st international workshop on Engineering Societies in the Agent World*, pp. 93–105, 2000.
- [60] O. Gutknecht and J. Ferber, “The MADKIT agent platform architecture,” in *Proceedings of International Workshop on Infrastructure for Multi-Agent Systems*, pp. 48–55, 2000.
- [61] “MADKIT.” <http://www.madkit.org>.
- [62] O. Gutknecht and J. Ferber, “MadKit: a generic multi-agent platform,” in *Proceedings of the fourth international conference on Autonomous agents*, pp. 78–79, 2000.
- [63] S. Tisue and U. Wilensky, “Netlogo: A simple environment for modeling complexity,” 2004.
- [64] N. Collier, “Repast: An extensible framework for agent simulation,” <http://repast.sourceforge.net/projects.html>.
- [65] “Microsoft Agent.” <http://www.microsoft.com/msagent/default.asp>.

REFERENCES

- [66] “Extensible Markup Language (XML).” <http://www.w3.org/TR/REC-xml/>.
- [67] Z. Shen, R. K. L. Gay, Y. Miao, and C. Miao, “Goal autonomous agent architecture,” in *Proceedings of the 28th International Computer Software and Applications Conference*, pp. 45–46, 2004.
- [68] J. Kolodner, *Case-based reasoning*. Morgan Kaufmann Publishers Inc., 1993.
- [69] P. Busetta, R. Rönquist, A. Hodgson, and A. Lucas, “Using Java for artificial intelligence and intelligent agent systems,” Technical Report 99-04, The Agent Oriented Software Group, 1999.
- [70] A. Angehrn, T. Nabeth, and C. Roda, “Towards personalised, socially aware and active e-learning systems,” tech. rep., The Center for Advanced Learning Technologies, 2001.
- [71] “Active Worlds.” <http://www.activeworlds.com>.
- [72] “Dengue Fever.” <http://www.moh.gov.sg/corp/about/faqs/illness.do>.
- [73] “Java Native Interface (JNI).” <http://java.sun.com/j2se/1.3/docs/guide/jni/>.