Agent simulation of vessels traveling in sea

Zhang, Huiliang

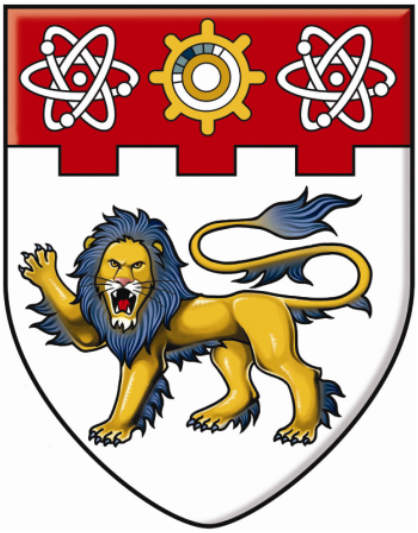
2008

Zhang, H. (2008). Agent simulation of vessels traveling in sea. Doctoral thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/2605

https://doi.org/10.32657/10356/2605

NANYANG TECHNOLOGICAL UNIVERSITY



Agent Simulation of Vessels Traveling in Sea

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

Zhang Huiliang
Supervisor: Huang Shell Ying

Division of Computer Science
School of Computer Engineering
Nanyang Technological University
Singapore

Dec 2006

ACKNOWLEDGEMENTS

Greatest thanks to my supervisor, Dr. Huang Shell Ying, who provided invaluable help and indispensable guide for the research. She often provided new and useful papers in the area to me. And she was always patient in discussing research with me. From her wise suggestion, I learned much for the research. I am also imbued with her meticulous attitude for science.

I would like to thank all my friends for their help and support. Without them, the living in school would have been very boring. At the same time, I also learned much from them. My lab, PDCC, provided me the biggest convenience for research. The center director Dr. Stephen Turner, technician Irene Goh, Lau Lijun and Ek Ming Hong were very helpful in solving the problems that I met in the lab. Thanks to all staff who helped me.

Without financial support, this research could not have started. I would like to thank Nanyang Technological University for providing me scholarship, which was the main financial source to support my study and living during my PhD candidate period. I would also thank my supervisor and professor Hsu Wen Jing for providing me a part-time job opportunity. Not only I earned some livings for the late period without scholarship, but also it provided a working experience.

Finally, I want to say thanks to my parents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The objective of our project is to design an agent architecture to simulate the intelligence and behaviour of a vessel captain in navigation. An agent representing a vessel captain should be able to perceive the environment, make decisions and act simultaneously. The agent should be able to prioritize its activities according to their importance and urgency. The agent should be able to reconsider its goals and intentions and adapt in the changing environment. The agent should be able to improve its performance with the accumulation of experience. Different vessel captains and thus different vessel agents should behave differently based on different personalities and past experience. It is also the objective of our research that work done here is general enough for building agents in other contexts like a robot looking after a patient or old people.

Many agent architectures have been proposed based on various processing philosophies, including deliberative architectures, reactive architectures and hybrid architectures. The deliberative agents have powerful reasoning ability compared to the reactive agents, but the slow processing speed due to the theorem proving based on complex symbol systems of the world makes them unsuitable for some dynamic environments. The agents based on the reactive architecture do not need deliberating and have quick processing ability. However, it is hard to design and maintain such agents, especially for complex agent systems. More importantly, the reactive agents lack learning ability which is essential for a truly automatic and evolutionary agent. The hybrid architecture combines deliberative and reactive architectures. The emergencies can be processed by the reactive layers while the deliberative layers process other decisions. Currently, the behaviours of all the existing agents are organized in a sequential way: detect-think-act. When an agent is thinking, it cannot detect the environment and may be in the danger of overlooking emergencies.

In this thesis, a general framework for real time performance in the Belief-Desire-Intention (BDI) model is proposed. It is an improvement for the BDI agent model. The agent consists of three parallel components: belief manager, intention generator and intention executor. The communication among them is realized by interrupts. The current running actions in the intention generator or intention executor can be suspended if the new incoming data has a higher priority. It supports the following agent abilities at the architecture level: (1) the ability to respond to emergencies timely; (2) the ability to reconsider and modify goals, intentions and actions in reaction to unexpected or new information; (3) the ability to perform multiple actions at once; (4) the ability to perceive, deliberate and act simultaneously; (5) the ability to prioritize the deliberations and intention executions. The architecture provides a possibility for the deliberative agents to be applied in complex and dynamic environments. A comparison experiment among the parallel agent and the sequential ones is made by simulating the processing of incoming events. The results show that the parallel agent has a powerful processing ability. The issue of how much parallelism and how to configure a parallel agent based on the general framework are studied by experiments with different configurations of the parallel agent.

Furthermore the vessel agent is personalized by its past experience and personality. We incorporate Experience Function library into the basic BDI model. As an example for accumulating experience, we apply the reinforcement learning algorithm to improve the agent's skills of obstacle avoidance. The algorithm is incorporated into the vessel agent as an Experience Function. The agent accumulates the experience during its navigation and the different past experiences will make the agent behave differently.

Then we propose a Priority Control extension to the BDI agent. The priorities of the deliberations/intentions in the agent can be controlled by proper Priority Control Functions. This provides a way to schedule the deliberations/intentions. A *reminding-forgetting* Priority Control Function is designed by simulating human behaviours when dealing with several things at the same time. Such function can be used when designing human-like agents. The agent with different settings for the Priority Control Function behaves differently.

Finally, a software agent system of vessel captain traveling at sea is developed based on the parallel BDI agent framework with the Experience Function library and Priority Control components. The structure for realizing the software agent is designed. The experiments show that the agent is able to respond according to expectations.

At the end of the thesis, we conclude on the contributions made in this research. Possible future research and applications are also discussed. The work presented in this thesis was done in simulation. We expect that it can be applied in real robots some day.

ABBREVIATIONS

AFSM	Augmented Finite State Machines
ART	Average Response Time of all events
ART_e	the estimated ART
ART_p	the ART of the events with priority *p*
ART_w	the weighted ART by the priorities of the events
AVG_d	the average PG time
AVG_e	the average PE time
AWTD	Average Waiting Time for Deliberations
AWTE	Average Waiting Time for Execution of intentions
BCM	Beam-Curvature Method
BDI	Belief-Desire-Intention
BDIE	Belief-Desire-Intention-Experience
BG	Belief Generator
CPEF	Continuous Planning and Execution Framework
CVM	Curvature-Velocity Method
DAI	Distributed Artificial Intelligence
DG	Desire Generator
DS	Desire Scheduler
DWA	Dynamic Window Approach
EEC	Emotion Eliciting Condition
EM	Environment Monitor
HTN	Hierarchical Task-Network
IM	Intention Manager
IS	Intention Scheduler
LCM	Lane-Curvature Method
LORA	Logic of Rational Agents
LSA	Logic-Based Subsumption Architecture

NPC	Non-Player Character
PCF	Priority Control Function
PE	Plan Executor
PG	Plan Generator
PRS	Procedural Reasoning System
RBF	Radial Basis Function
VFF	Virtual Force Field
VFH	Vector Field Histogram

CHAPTER

1

INTRODUCTION

1.1 Background

The Singapore Strait is used by vessels entering and leaving the Port of Singapore as well as by transiting vessels. Vessels enter and leave the Port of Singapore via various navigational approaches. The types of vessels using the Singapore Strait range from very large container carriers to small crafts such as passenger ferries. During peak periods, like morning or evening, or public holidays, a larger number of ferries will appear. The heavy marine traffic makes certain sea areas very crowed and accidents do happen occasionally.

In such an environment, vessels exhibit the following behaviours:

- Moving towards the destination. This means that a vessel has to reach its target, instead of navigating aimless. For example, a vessel moves to the Port of Singapore. With a given destination and the map of the sea, each vessel plans its own route to arrive at the destination.
- Avoiding stationary objects and other moving vessels. For example, for a transiting vessel, it should avoid collision with islands, reefs and lighthouses spotted in the Singapore Strait. At the same time, when it meets other navigating vessels, it should also avoid these moving vessels. Thus methods of avoiding dynamic obstacles are necessary.

In the real world, vessels are under the commands of their own captains. As people have different personalities, different captains demonstrate different vessel navigation arts. So human factors have very important effects on vessel navigation. The human factors affecting navigation can be seen from the following facts:

- Captains have different behaviours in moving towards the destinations. Some people tend to move fast, some take it easy. Some are behind schedule, therefore have to rush. Some are before schedule, so need to slow down.
- A bold and a meticulous captain may have different styles of command when dealing with the same situations when other vessels are nearby. A meticulous captain always adopts the safest strategies earlier than his bold peers. Different types of vessels also have different velocities, sizes and capacities.
- Experienced and green-horn captains have different reactions to the same events.

A simulation system of vessels traveling at sea is very useful for risk analysis and channel capacity estimation in the Singapore Strait or any other waters. The risk analysis will be carried out for the interaction between each type of vessel and each of the other types of vessels, the time of the day and the different areas. The simulation system will be able to indicate what type of vessels, what kind of captains or behaviour, which area and what weather condition are high risk factors. Then remedial or precautionary actions may be taken. This simulation can also be used to find how many vessels can safely use the Singapore Strait at the same time. The channel capacity is defined as the number of vessels that can safely use the channel. Ferry schedule determines the frequency and the size of the ferries. Given a fixed demand of passenger capacity, increasing the number of high seating capacity ferries will reduce the ferry frequency. However, reducing ferry frequency may cause costumer unhappiness and drive out business. The channel capacity estimation can be used to find an optimal balance between them. In order to have meaningful conclusions from the simulation system, different vessel behaviours must be simulated realistically.

1.2 Objectives of This Research

In this research, we will try to design an agent architecture for the agent that replicates the behaviours of vessel captains traveling in sea. The captain agent has the ability to navigate from starting point to target using different navigation methods. Each vessel plans its global path first using a global path-planning algorithm. When the vessel moves along its path, it may detect some unknown obstacles. Then it uses some local obstacle avoidance methods to avoid collision with them while still trying to move to its destination. More importantly, the vessel agent should obtain: (1) the ability to respond to emergencies timely; (2) the ability to reconsider and modify goals, intentions and actions in reaction to unexpected or new information; (3) the ability to perform multiple actions at once; (4) the ability to perceive, deliberate and act simultaneously; (5) the ability to prioritize the deliberations and intention executions.

It is also the objective of this research that the work done here is general enough for building agents in other contexts.

1.3 Main problems and Technical Issues

The agent will be realized based on the famous BDI agent architecture. Currently, the BDI agent architecture is not suitable to simulate a real-time vessel captain in two aspects: 1. the reactivity of the agent cannot be assured; 2. the characters of the captain cannot be easily realized. We design a parallel BDI agent architecture to solve the first problem. As we said earlier, a vessel is under the commands of its captain, a human being. The captain's own personality and experience will affect the vessel's navigation. So we integrate some more components into the parallel BDI agent architecture to realize the agent characters. In the future, multi-agent simulation of the vessels based on the parallel BDI agent architecture can be used for the risk analysis and channel capacity estimation as discussed in Section 1.1.

To validate the performance and applicability of the agent architecture we proposed, the research methodology adopted is computer simulation. In the simulation, we assume that

the agent can receive the world information in the form of beliefs and the actions are carried out immediately. We will judge whether the system runs according to our expectation by examining the behaviour records of the agent.

1.4 Thesis Organization

The following chapters are organized as following:

- Chapter 2 LITERATURE REVIEW. This chapter introduces related research, including agent architectures, agent character, agent learning, action selection and agent systems. The techniques related to robot navigation are also reviewed here.
- Chapter 3 PARALLEL BDI AGENT ARCHITECTURE. In this chapter, the design of the general parallel BDI agent framework is introduced. Simulation experiments of the parallel BDI agent and several sequential BDI agents are performed. The experiment results have demonstrated the advantages of the parallel BDI agent. The parallelism is analyzed using experiment simulations.
- Chapter 4 AGENT CHARACTER. The basic agent character is analyzed in this chapter. The Experience Function library is incorporated into the agent to support combining the reinforcement learning algorithm. Then an experiment simulating the agent experience is made.
- Chapter 5 PRIORITY CONTROL. A component to control priority change in an agent is proposed. As an example, a personalized priority control schema for action scheduling is shown. The schema is designed by simulating human behaviours when dealing with several things together.
- Chapter 6 A VESSEL CAPTAIN AGENT. A software vessel captain agent is realized based on the general parallel BDI framework with the two character components: experience function library and priority control schema. The architecture to realize the software parallel BDI agent is shown. The captain agent is realized by using multi-threads programming techniques. The agent shows the applicability of the parallel BDI model.
- Chapter 7 CONCLUSIONS AND FUTURE WORK. We conclude our researches and make some proposals about future research.

CHAPTER

2

LITERATURE REVIEW

In The Merriam Webster Dictionary, agent is defined as: "agent *n* 1 : one that acts 2 : MEANS, INSTRUMENT 3 : a person acting or doing business for another" [3]. From the viewpoint of the semantic meaning, we may regard the computer agent as an instrument that acts. However, a single definition of 'computer agent' is not accepted unanimously since it first appeared in the 1970s due to the diversification of the computer agents' attributes, roles, architectures, and other features.

In 1977, Hewitt introduced the concept of agent as 'actor' in the research of Distributed Artificial Intelligence (DAI). In the model, an actor "is a computational agent, which has a mail address and a behaviour. Actors communicate by message-passing and carry out their actions concurrently" [53]. This is the original model of an agent. From then on, the research on agents has been carried out in various areas and applications. In an overview of software agent, Nwana describes software agents as a broad range of computational entities [100]. From his viewpoint, agents in software area can be reusable software components that provide controlled access to (shared) services and resources or the basic building blocks for applications organized as networks of collaborating agents. For a real physical system, like a Robot World cup team, each robot also can be seen as an agent,

which is a computational unit [73]. The whole team is considered as a multi-agent system, in which each agent can finish some tasks individually and cooperate with other teammates. Both kinds of computer agents will embody some characteristics of humans. In fact, the ultimate objective of agent research is to make agents act as real intelligent human agents. In our research, the final objective is to design an agent which is able to behaviour as a real vessel captain in navigation.

As pointed out in [100], the three common attributes of agents are: autonomy, cooperation and learning ability. Such attributes make the agents different from conventional programs. Firstly, programs are sequences of clear and detailed instructions provided by their designers to be followed exactly. But agents are autonomous and act on behalf of a user. Agents act according to their own desires and interests without getting detailed instructions from the user. Secondly, agents often need to cooperate and coordinate with others. They have social ability. This human-like ability is lacking in conventional programs. Agents often communicate with the user, the system, and other agents. Through communication, agents can obtain knowledge about the user's reaction, environment and others' intention. Agents can then decide and act more effectively. Agents can also cooperate with other agents to carry out more complex tasks than what they can handle themselves. This cooperation can be seen from a Robot World Cup team. The agents must cooperate with each other efficiently to gain victory, like a human soccer team. In a multi-agent system for distributed computing, agents often obtain the ability to access remote resources. Thus, the efficiency of the whole system can be increased [57]. Thirdly, some agents have learning ability, which shows that they have some kinds of intelligence. If agents do not learn, they are not suitable for dynamic environments where situations cannot all be foreseen. Agents learn from actual actions and/or training. Thus they can choose the best actions based on the experiences gained from past actions. When under training, sometimes agents must be proactive. For example, an agent may test some actions to gain the rewards from these actions. Different to this, most programs always choose their actions using the same approach according to the way it is programmed. In recent years, emotions also become a very important

attribute for the agents in some applications. This shows another distinction from traditional programs: efficiency is not the only goal to pursue.

Research about agents has been illustrated, summarized and concluded in many books or papers, such as [107, 65, 100, 135, 115, 137]. In these publications, the agent research is divided into many areas:

- Agent architectures
- Agent language and programming
- Agent characters
- Automatic learning in agents
- Agent systems and applications
- Multi-agent systems
- Agent coordination and negotiation
- …

Nowadays the areas of agent research have become so large and wide that it is very difficult to include all the areas even in a book. Here, our simple summaries will be focused on the areas which are mostly related to our research, including agent architectures, agent character and action scheduling. Agent learning and some agent systems are also introduced. At the end, some researches of robot navigation are examined, which are related to our vessel agent simulation.

2.1 Agent Architectures

Wooldridge and Jennings group agent architectures into three types: deliberative architecture, reactive architecture and hybrid architecture [135]. A deliberative architecture is based on the hypothesis of physical symbol system, which is said to be capable of general intelligent action. Different kinds of deliberative agents emerged from it, for example, planning agents, Belief-Desire-Intention (BDI) agents, Homer and so on. However, many unsolved problems existing in symbolic AI restricted its developments. A reactive architecture is designed without symbolic world model and complex symbolic reasoning. A hybrid agent architecture is built by a combination of these two agent

philosophies. The hybrid agents try to maximize the strengths and minimize the deficiencies of the most relevant technique for a particular purpose [100]. This kind of agent can inherit the advantages from other agents and avoid their shortcomings. It is becoming more popular in agent design.

In the following, we will review these three kinds of agent architectures. We choose the BDI architecture as representative for deliberative architecture and the subsumption architecture for reactive architecture because of their popularity among the peers.

2.1.1 BDI architecture

BDI architecture is the deliberative architecture that is researched mostly. It provides a folk psychological way by simulating human deliberation. The mental attitudes of belief, desire, and intention represent the information, motivational, and deliberative states of the agent respectively [18, 111]. It may seem useless for simple agents, like a thermometer or an alarm clock, but it is helpful when developing agents that work in complex environments.

Rao and Georgeff provided a BDI model in software engineering area [111, 110]. They defined the BDI components and explained their significance to agents. The actions or procedures that achieve the various objectives are dependent on the state of the environment and are independent of the internal state of the system. So it is necessary that there is some component of system state which represents the information on the state of the environment and which is updated appropriately after each sensing action. Such a component is called the system's beliefs. It is also necessary that the system has information about the objectives to be accomplished. This component is called the system's desire. In order to limit the frequency of reconsideration and thus achieve an appropriate balance between too much reconsideration and not enough, it is necessary to include a component of system state to represent the currently chosen course of actions. This additional state component is named as the system's intention. This BDI agent architecture is used in an air-traffic management application [111]. Brazier et al. present

an extended task hierarchy for a BDI-agent in [19]. The agent process control consists of the belief, desire, intention and commitment determinations.

Procedural reasoning system (PRS) [63] is a famous implementation of the BDI model. The deliberative process runs in iterations. At the beginning of each iteration step, new goals and new facts are obtained through input. Then several plans in the KA (knowledge area) library are triggered by the new belief and one or more of the applicable plans are selected to be sent to the intention structure. At the end of each iteration step, the intentions are executed. This kind of idea of implementing the BDI agent is adopted in many BDI systems [109, 4, 56]. In UM-PRS [77], an extension of the PRS system, the hierarchy of the plans is kept for monitoring plan execution and replanning. The formal specification of the PRS can be found in dMARS system [28]. JAM is a BDI agent architecture developed by Huber in 1999 [59]. It combines the advantages of the previous BDI agent researches. With the JAM toolkit, users can create and run their own agents by designing beliefs, plans, and primitive functions following the defined grammar. The basic structure is similar to PRS system.

AgentSpeak(L) is a popular BDI programming language proposed by Rao in 1996 [112]. It defines a set of basic beliefs and a set of plans. The plans are searched for the triggering events (new beliefs). Then applicable plans are inserted into the intention stack for execution. A more formal description of AgentSpeak(L) can be found in [29]. This language has been combined into other agent language, for example 3APL [54]. SIM_AGENT is one application based on the AgentSpeak(L) [83].

The reasoning of the BDI agents can also be performed by automatic theorem provers. A set of logic is defined in such agents. For example, Wooldridge introduced a BDI logic called Logic of Rational Agents (LORA) in his book [136]. LORA contains a temporal component as an addition to the traditional first-order logic. The theorem provers are used to produce some outputs.

The BDI agents have been applied in many applications. Rana et al. have applied Conflict Management Strategies in BDI Agents for Resource Management in Computational Grids [109]. A rational agent executes a plan from a pre-defined plan library (belief) to achieve local goals (desire), and can try alternate plans (intention) if a goal cannot be achieved by a chosen plan. Ambroszkiewicz and Komar use the BDI model in a Game-Theoretic Framework [4]. In a game, the agent's belief is identified with the knowledge about the game and about other agents together. The desire is represented as agent's goal to achieve a maximum level of its utility. A reasoning process based on the agent's rational behaviour is proposed. This process determines the agent's intention. Rational behaviour may be used to construct such reasoning process. The process of reasoning is defined as a transformation that conveys the knowledge from higher types into lower types and finally into the ground type. This final ground knowledge is the basis for determining the final intention.

Recently, a flexible BDI agent system is proposed in [106]. This paper identifies two drawbacks of the sequential BDI agents. One is that concrete layout of the cycle will determine the nature of the agent, for example, the caution level and reconsideration rate. Another drawback is that the agent architecture is not easy to be extended with additional facilities because the processing is step by step and very restrictive. The authors propose a more flexible way of mapping the original BDI model to a system based on agenda scheme in order to allow easier extension of the agent. The steps are transformed to meta-actions. A main interpreter will decide which meta-action will be selected to execute from the agenda queue. The execution of the meta-action may update the status and insert new meta-actions into the agenda. The extension of new agent abilities can be easily done by designing the meta-actions. However, the outside messages are inserted into the agenda directly as external actions. This may indicate that the agent does not detect outside environment automatically. If the detection action is modeled as regular meta-actions, the concrete layout problems still exist. The belief cannot be updated in real-time because the detection is performed in predefined intervals. Then caution level and reconsideration rate cannot be improved.

As seen from above, the BDI model can be used to design rational agents. In the design of a vessel agent, we will develop an agent architecture based on the BDI model. The environment information can be seen as the vessel agent's belief. The desire is to navigate to target safely. The vessel' admissible actions can be seen as intentions. We design a parallel BDI agent architecture to solve the problems caused by slow deliberating in traditional BDI agents. This is shown in Chapter 3. The BDI model represents the general attributes of vessel agents. However, the BDI model is not sufficient to represent realistically the vessel agents. There is no a proper representation of the vessel captain's characters in the model. In a real world the character of a captain will affect the decision he makes to control his vessel. Even in identical environments, two captains may have different navigation decisions simply because one is more conservative and cautious than the other. The personality and character of the two captains are making the difference even with the same beliefs and desires. There is no component in the BDI model to represent variations among agent characters. Thus, we plan to incorporate a new factor into the BDI model. We call it 'character', which represents agent's personality. The agent is expected to behave human-likely as demonstrated in Chapter 4 and 5.

2.1.2 Subsumption architecture

In deliberative systems, the world is represented by symbols and the reasoning is performed through theorem provers. Since the speed and efficiency of the provers cannot be ensured, this makes this architecture unsuitable in a dynamic environment. A reactive architecture takes a different approach than symbolic AI. It does not include any kind of central symbolic world model and does not use complex symbolic reasoning. Among reactive agent applications, Brooks' subsumption architecture is the most celebrated one [22, 21]. The architecture consists of a set of modules, each of which is described in a subsumption language based on augmented finite state machines (AFSM). An AFSM is triggered into action if its input signal exceeds some threshold, though this is also dependent on the values of suppression and inhibition signals into the AFSM. The modules are grouped and placed in layers, which work asynchronously, such that

modules in a more complex level can inhibit those in lower layers. In [84], a learning algorithm is proposed to improve the scheduling of the behaviours/layers based on the feedbacks when to activate the behaviours.

This architecture is often used for robots. Each layer has a hard-wired purpose or behaviour, e.g. in a robot a layer is to avoid obstacles and other layer is to enable/control wandering [100]. The different layers represent different behaviours of a robot. Then the behaviours from a more complex layer suppress the behaviours from the low layers. For example, in the MIT AI laboratory Mobots [23], three kinds of behaviours are controlled in three layers. The lowest-level layer implements a behaviour, which makes the robot avoid being hit by objects. The next layer makes the robot wander about when not busy avoiding objects. The third layer makes the robot try to explore. There is no central control in the robot. Each layer is driven by the messages it receives. Though the process of deciding actions is similar to neural network, Brooks claims that this architecture has no relation with neural network because there is no biological significance existing in the architecture.

The architecture is simple and efficient in terms of the amount of computation required. But the limitations are also obvious [137, 78]. One problem is that the arbitration technique only allows a single behaviour to be active at one time. The architecture chooses one action at each deciding cycle and other actions are suppressed. Though the deciding speed is promising, the single action decision will affect agent's performance. For example, if a robot's action is to avoid obstacle, the robot may deviate far away from the target. Several researchers have proposed to incorporate fuzzy logic technology with this architecture [78]. The decision process is fuzzied. In the process, behaviours from different layers are composed together and decomposed to get a final decision result.

Another problem exists in that the agent makes decisions based on local information. Thus, global information is omitted when making decisions. This means that the agent always has a 'short-term' view. The architecture is not suitable for making global plans. At the same time, the information from the local environment needs to be sufficient,

otherwise the agent will not be able to determine its actions because the agent does not store models of environments. An improvement to deal with this weakness can be seen in an architecture for persistent reactive behaviour [26]. Long-term conceptual memory, long-term skill memory and short-term memories are incorporated in the agent. The knowledge encoded in the memories can be utilized when the agent updates its beliefs, selects and executes skills. A persistence factor is used to control the agent's bias to select the skills picked on the previous time step. Thus, the agent can take into account the global environment and its previous behaviours.

It is also difficult to implement agents' learning ability using this architecture in a hard-wired implementation. Purely reactive agents can hardly be designed to learn and improve performance over time. Besides, agents with many behaviours are very hard to build. The dynamics of the interactions between behaviours are very complex to understand. This implies that the subsumption architecture cannot be applied in a dynamic and complex environment. For our vessel agent, the deliberative architecture can better simulate the human behaviours with psychological significance.

It is worth noticing that the layers run asynchronously in a subsumption architecture. In [21, 66], the layers are run concurrently. The speed of reaction of the agent is increased by parallelism. In a recent research, the deliberation ability is realized based on the machinery of the subsumption architecture. In Logic-Based Subsumption Architecture (LSA), the layers are realized as the theorem provers. Thus the reasoning ability of the deliberative agents is combined into the reactive agents. At the same time, the empirical results from a robot implementation show that the provers can be used without sacrificing much reactivity [5, 6]. Each control-loop cycle is shown to take 0.1-0.3 seconds, which is acceptable for the robot in their experiment.

Another recent research about the subsumption system is a dynamic subsumption system [91]. The layers consist of several cells, which contain possible partial descriptions of certain functions of the agent. If the environment changed, only the related cells are affected.

2.1.3 Hybrid architecture

A hybrid architecture is an architecture that combines the above two kinds of architectures together. This represents the new and popular trend in designing robots, because this architecture can inherit the advantages of the two architectures. For example, a reactive architecture is suitable for real-time environments, but the reactive architecture produces behaviours not goal-oriented at times. A deliberative architecture can handle that, but it sometimes cannot react timely. A hybrid architecture incorporating two architectures can solve the problems. In fact, human can be seen as a hybrid system. In the human reasoning system, we do not really think and spend time deliberating what to do in face of an emergency. For example, the kitchen is on fire, we just get water and put out the fire without spending time deliberating.

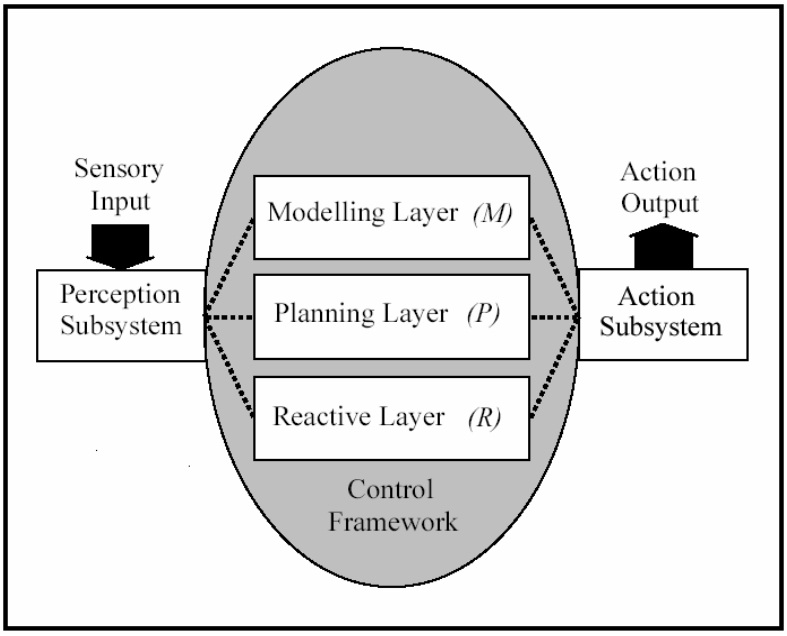


Figure 2-1 The TouringMachines agent control architecture (from [35]).

An example for this architecture is Ferguson's TouringMachines hybrid agent architecture [35, 36]. We can see from Figure 2-1, the three layers work together to control the agent. The reactive layer is in the style of the subsumption architecture. The

planning layer is designed as a deliberative architecture, so the agent can have an overall planning ability and also can deal with emergencies. The modeling layer is used to model other agents in the environment. The three layers are embedded in a control framework, which deals with conflicting proposals from different layers by using control rules. The control rules will decide which action in the action buffer is chosen to be sent to the agent's effectors. The rules will ensure that only one action will be activated at a time slice.

As we can see, the hybrid architecture is good at representing both the meditated behaviour and the reactive behaviour. But as Ferguson points out that there were still many problems existing in the TouringMachine, a hybrid architecture as TouringMachine is not really perfect for agents which operate in dynamic and unpredictable multi-agent environments. For example, there is not a learning component in the architecture, which may improve the agent's adaptation in new environment. Also the computations in the three layers are restricted strictly by the pre-determined time resources. This can be seen as the concrete layout problem of the cycle in [106]. Thus the TouringMachine may fail to make the best use of the time resources.

This hybrid architecture has been used successfully in designing robot agents. In most cases, the deliberative layer is used for global path planning and the reactive layer for obstacle avoidance, subgoal decision and so on. In [11], three layers are used for a 3T robot architecture: deliberation, sequencing and reactive skills. The sequencer is used to activate and deactivate the skills. In the hybrid mobile robot [82], deliberative and reactive models are used for long-term and real-time decision respectively. The planning module will provide targets for the target reaching module. The commands from the three layers in reactive models are combined to make the final motor control signal.

Other kinds of hybrid architecture without the explicit deliberative and reactive layers also exist. Sometimes, it is a combination of agent abilities. For example, an architecture for Non Player Character (NPC) is proposed in [92]. The NPC agent architecture consists of 4 layers: behaviour system, social system, goal based planner and schedule. The

selection is made by selecting from the outputs. Another example is a homogenous agent architecture for robot navigation [62]. In the designation, an agent can be created by combing the functions of several other agents, such as goal-seeking agent, vacancy-pursuing agent and obstacle avoidance agent.

2.2 Agent Character

Normally, agents are built to make rational and best-fit decisions. Thus, in the same situation, different agents will all make the same decisions and they will have the same behaviour. However, in some areas, this is not always desired and the agent should demonstrate its own character. For example, in multi-agent simulation of human society [61], agent character is essential for simulating various human beings. The agents will not always work in an ideal way. Their characters will affect their decisions. As Sloman points out when answering the question of what sort of architecture is required for a human-like agent, "designing human like agent is part of the more general problem of understanding design space, niche space and their interrelations, for, in the abstract, there is no one optimal design, as biological diversity on earth shows" [120]. The discussion of a similar question is seen in [121].

The agent characters separate one agent from another. An agent's character can be found in three aspects: the physical characters, experience and the emotions. The physical characters include the agent's basic attributes of physical resources, for example, a vessel agent's length, maximum acceleration, maximum loads and so on. The experience can be expected to be realized by reinforcement learning algorithms. Compared to the physical characters and experience, most researches working on the human-like agents concentrate on simulating human emotions. Mainly, there are two objectives of this research: creating human-like agents and promoting agent performance.

2.2.1 Creating human-like agents

The most direct application of the emotions should be human-like agents, which are supposed to show some human-like behaviours [117]. Humans show various behaviour modes according to different emotions. In [122], it is said that though we do not understand how human emotions work, by trying to model emotions, it is possible to learn more about the emotions, and it is possible to create more realistic agents.

In the game area, the Non-Player Character (NPC) agents must demonstrate different emotions for a vivid scene. In [96, 99], different kinds of Quake players are created based on the BDI model. The players with different interactive characters show different behaviours when executing the plan 'win'. The characters are created by making some probes. Then the characters are created based on the answers of the agent. In [97], Norling argues that the BDI model is incapable of representing several human characteristics. In a psychological way, the characteristics include decision making, expertise, emotion, timing, and so on. She proposes to make a folk psychological extension to the BDI model to represent these characteristics. Some special modes representing the characteristics are incorporated into the BDI model to show the character. An example of incorporating the recognition-primed decision model with the BDI model to make human-like decision making is given. In [95, 98], COJACK architecture is proposed to support psychologically plausible human variability. In this architecture, that agent's reasoning and actions are moderated via a set of parameters. Some external and internal moderator will also affect the agent's decision.

A five-factor (extraversion, agreeableness, conscientiousness, neuroticism, and openness) model of emotion is borrowed from the psychological research. It is used to affect the learning strategies of the agent [50] and the information behaviours [51]. Patrick Gebhard makes his research to connect the five-factor model of personality to the agent behaviours. A set of appraisal and dialogue act tags is designed. The tags are mapped onto the emotion eliciting condition (EEC) variables to control the emotion changing. Then the emotion will affect the behaviours of the agent [41]. This EEC system is used in multi-character conversations [42]. Based on this, a layered model of affect is designed.

Emotion, mood and personality are differenced in terms of short, medium and long periods. The personality is realized as using the five-factor model. The five values are defined by user at start. PAD (Pleasure, Arousal, and Dominance) system is used to simulate mood. Then the 24 kinds of emotions are defined by the PAD values with different weights. The agent's behaviours are modulated based on the PAD values according to the predefined rules [43]. A similar emotion model based on the five-factor model is shown in [52]. The emotion factors are incorporated to make the decision-making process of complex agents less predictable and more realistic. A kind of emotions architecture is implemented with a three-dimensional personality space (Arousal, Pain, and Confusion). The different status of the agent can be mapped to different emotions. Then the emotions are used to make decisions together with the external and internal inputs.

Another method to show the agent emotions is by designing the agent's distinct behaviours directly. The behaviours of the agents can be expected to be different because of the personalities. The audiences can conclude the agents' personalities based on the behaviours they saw. An example of SceneMaker can be seen in [40]. The roles can make plans based on the pre-scripted scene. In [87], a method of decision making for social agents is proposed as the PsychSim system. From the theory of mind, the action of the agent must exhibit consistently, self-interest, speaker self-interest, trust and support. A quantitative value of each possible action is defined according to their beliefs and goals. Such values can be modulated after the interaction with the environment, so the agent will show different behaviours. This method is used to create characters based on the story scripts as in the Thespian system [118]. Thespian is a tool to create agents with personalities which are consistent with the behavious defined by the story path. The characters' goal weights can be modulated with the equation defined in PsychSim. The character can be reused in different scenes. The agent will try to select an action based on the reward of applying the actions to the current state.

2.2.2 Promoting agent performance

Besides the applications for human-like agents, emotions also help to promote the adaptability and autonomy of the agents. Though emotions were thought useless for agent reasoning for long time, in [102], three benefits are identified. First, the emotion itself is an important source of information which is highly centered on the individual. The emotions will affect the agent's behaviours as we show above. Secondly, the emotional mechanisms are useful to filter relevant data from multiple, distributed and highly noisy sources. An example can be seen in [86]. Here, emotions are used to change beliefs of the characters in a scenario from the mission rehearsal exercise. With the same conversations among the characters, the beliefs of mother, sergeant and soldier are changed according to their own emotions. Thus their behaviours are consistent with their roles. Finally, the emotions also provide a global management over other cognitive capabilities and processes. In [132], the author states that the action decision of the agent should be affected by both the sensory input and the desires of the agent. Agents with different emotions may produce different desires. An example for human-like decision-making can be found in [93, 94]. A recognition-primed decision making approach is integrated with the BDI model as an emotion feature. The agent can learn from past experiences.

2.3 Agent Learning

An agent can better adapt in a dynamic environment if it has some learning abilities. Then the agent can change its strategies to cope with the new situations automatically. Automatic learning is that an agent saves historical actions and scenes in order to use when meeting the same scene again. Learning is divided into supervised learning and unsupervised learning [125]. Under supervised learning, a manager exists and is responsible for providing training samples to agents. Such samples can be chosen from special examples. This may fill agent with experience quickly [45]. In an unsupervised learning situation, there is no such a special manager. The agent will learn from all random events happening in environment, and the agent can become more robust for various environments.

Reinforcement Learning is viewed as an on-line variation of dynamic programming, which is defined as a discrete-time system with the state transitions and costs/reward functions [124]. Using reinforcement learning algorithm, an agent can choose an action based on its current and past status. The algorithm will use a reward function to choose the maximum reward value for several future steps. Different kinds of reward functions have been defined in various situations. According to the book [80], reinforcement learning may be computationally implemented depending on (1) whether some heuristics are employed, (2) whether a model of the problem domain or a utility function for action selection is available, or (3) whether the learning always converges. For example, Q-learning does not utilize any domain model, but tries to iteratively derive an action-weighting function.

Learning is very useful to improve the competition and coordination strategies in multi-agent system. For example, Stone, Riley and Veloso used the learning method to train their robot soccer team [104]. The robot can gain experience through examples and choose better actions in a real game. Learning is also used for data collection problem. Caragea, Silvescu and Honavar designed a multi-agent decision tree learning from distributed autonomous data source [24]. Goldman and Rosenschein have made an application by incorporating mutual supervised learning into multi-agent systems [45]. They test the teaching technique in a scenario of a crosswalk with two traffic signal agents. Each agent controls the traffic light for its direction. Each agent is the other's teacher and also receives samples from the other.

With learning algorithms, a single agent can improve its behaviours according to the past experience. For example, in OBELIX, an automatic robot, the RL algorithm is adopted to solve the problem of pushing-box tasks. Experimental result showed that after the initial learning phase an agent will outperform a hard coded agent that does not learn [85]. In a more complex robot, the robot soccer team for Robot World Cup, Q-learning is used to help the robots finding the best actions [72, 58]. The percent of successful actions is greatly promoted after using Q-Learning for the robots. The learning algorithm for obstacle avoidance in navigation is another typical application. An example can be found

in [47]. The variables related to the positions of the robot and obstacles are used as the status. The parameters about control the robot motions are modulated through training.

2.4 Action Scheduling

When there are several actions/intentions waiting for execution, the agent should have a mechanism to decide the execution order. For example, in AgentSpeak(L) [112], the selection function S_I selects an intention to execute from the intention set *I*. However, the detailed selection criteria are not specified in the paper. Some scheduling mechanisms can be seen in other researches. Normally, there are two kinds of scheduling schema.

One is by a single attribute of the actions, for example, the priority. This is normally adopted in the systems where the actions are independent. Imaging that several dependent actions can be integrated to a single mega action, this kind of method can be seen as a general method for scheduling actions. An example is shown in the JAM agent architecture [59]. The intention selection is done based on the utility value of the plan. The intention with higher utility will be executed first. Recently, another work of intention scheduling is reported in [79]. The researchers take several properties into consideration when scheduling the intentions, such as the importance of the plan, the estimated running time, the deadline utility function, the degree of completeness and FairFacter.

The other scheduling method is by the relationship among the actions. This is suitable for complex job circumstances, for example, where the execution of an action depends on the results from the execution of other actions. For example, in AgentSpeak(XL), an extension version of AgentSpeak(L) [12], a task scheduler is incorporated into the interpreter to decide how to select intentions. The set of intentions in AgentSpeak(L) is converted into a corresponding TÆMS task structure. Then the selection is done based on the analyses of the relationship among the plans in the TÆMS task structure. The 'enables' and 'hinders' relationships indicate which plan may be executed first. A method to identify the potential common subgoal is provided in [127]. At first, the positive

common subgoals are identified. Then the potential common subgoals are figured out by maintaining summaries of definite and potential effects of goals and plans.

Human-like action scheduling schema has seldom been researched. In [74], a priority control mechanism for behavioural animation is shown. The priority is set at minimal value immediately after the agent displays certain behaviour like drinking. Then this priority is increased with time. The increased priority will induce the agent to drink again. A more formal description of this human behaviour system can be found in [75]. However, expecting the priorities of all intentions to change in the same manner is not realistic. Different intentions should be allowed to change their priorities in various suitable ways. Some intentions may also change priorities with the arrival of new beliefs. This problem will be discussed more in Chapter 5.

2.5 Agent Systems and Applications

Many kinds of agent and multi-agent systems are designed for real applications besides the robot agents. For example, the BDI agents were designed to manage the air-traffic [111]. In a resource management system, mobile agents are capable of finding computing resources in network, completing the goals, and returning the results [108]. The agents are also designed for providing an interface. In a hosting system, the agents interact with a visitor to design visiting schedule based on the visitor's areas of interest, name and organization [126]. As an example in the economic area, an agent is designed to perform Market-Based Supply-Chain Management [67].

Multi-agent systems are useful for solving problems which are composed of subproblems. As shown in [108], each mobile agent can complete a subgoal. Then all the results are brought together for making the final result. It also can be seen that a decentralized multi-agent system is more robust than the centralized systems. The container port system described by Thurston and Hu is another example for distributed multi-agent system [128]. The system is used to manage the container handling process in a port. Four types of agents are designed for the management tasks. The agents cooperate

with each other to accomplish the job. An agent failing will not halt the whole system. A method to transfer the centralized policies to the decentralized policies in the multi-agent system can be seen in [138].

Here, our focus will be put on the agent and multi-agent simulation systems.

2.5.1 Agent simulation system

Several human-like agent simulation systems have been developed in the agent character sector. Besides those, creature simulation is also an important topic in agent area. For example, a simulation system of a highland terrier is shown in [64]. This paper describes a kind of brain architecture for synthetic creatures. The brain consists of sensory system, perception system, working memory, action system, navigation system, motor system and blackboard. Action tuples are designed. If the TriggerContext is satisfied, the action will be executed.

2.5.2 Multi-agent simulation system

Multi-agent simulation provides a tool for simulating various societies. Simulation is widely used to enhance knowledge in real worlds and enables us to make artificial worlds for measuring the influence of different multi-agent coordination strategies in an unpredictable environment. For example, Horling, Kesser and Vicent have designed a simulation system that can be used for testing in an actual system [55]. This simulation system enables users to directly control the baseline-simulated environment and permit the addition of 'deterministically random' events that can affect the environment throughout the run. In an agent-based interaction analysis of consumer behaviours, Customer BEhaviior Simulator model is designed to simulate consumer behaviours when selecting a new brand [116]. The agent evolution is simulated using the GA algorithm.

As a hot research topic, multi-agent simulation systems of traffic were researched for predicating traffic information and finding ways to relieve traffic jam. In [103], the unorganized traffic is simulated. With different parameter settings, the drivers are

modeled to be cautious, normal and aggressive. The agent's action is calculated by physical motion laws. Then the average speed, the numbers of overtakes and accidents are counted. A fully agent-based simulation of the traffic in Switzerland is shown in [7]. In the simulations, it is important for the driver agents to think human-likely. A method to simulate human-like thinking is provided in [114]. Based on the psychological studies on human drivers, Rigolli and Brady propose that the agent translate the objective world into its own subjective world. With different parameters for perception, the agents will have different views of world. By simulating 330 agents, some macroscopic performances are gotten, including zone density and lane occupancy.

Better traffic control is important to relieve traffic jam. In simulation, the coordination can be done in two ways, centralized and decentralized. In the centralized way, a manage agent will collect all the traffic information and provide optional solutions. Traffic lights are used for this objective [101]. The light coordination is made using distributed constraint optimization. For a single intersection, a reservation-based mechanism is proposed in [31]. An improvement of this system is shown in [32]. The driver agents are assigned more abilities, for example, turning and accelerating in the intersection. In [27], the traffic signal controller agents are divided into three layers: intersection, zone, and region. The results from the lower layers are summarized at the higher layer. An example of decentralized control is shown in [139]. Each driver agent will send and receive the traffic information through a route information server. Then each agent will re-calculate its own shortest path based on the newest information.

2.6 Robot Navigation

Basically, there are two issues in navigation: path planning and obstacle avoidance. In this part, we will summarize the existing global path planning algorithms and obstacle avoidance algorithms respectively.

2.6.1 Global path planning

Path planning is a fundamental problem in navigation. A robot will usually do its path planning at the beginning of navigation. According to known map information, several intermediate targets will be put on the path line to the final target. Path planning methods assume that the environment does not change while a robot is moving. Latombe summarized a larger number of robot motion planning algorithms in his book [76]. These methods are based on a few general approaches: roadmap, cell decomposition and potential fields. The first two methods convert the planning problem into searching a graph by analyzing the connectivity of the whole free space. In these methods, an effective searching algorithm is involved. However, the potential field method is usually defined with a limited range of influence. It can be applied while the robot is moving. So the potential field method is often seen as a local method. We will introduce the potential field method as a local obstacle avoidance method in Section 2.6.2.1.

The general idea of the roadmap is to construct a network of one-dimensional curves. Then the roadmap is used as a set of standardized paths. The path planning is reduced to search a path between the initial and goal points. Based on this idea, various methods are proposed. The visibility graph method is one of the earliest path planning methods. In this method, a roadmap consists of line segments connecting two nodes that do not intersect the interior of an obstacle region. Then a path can be obtained through searching this roadmap.

Cell decomposition decomposes the robot's space into simple regions, called cells, such that a path between any two configurations in a cell can be easily generated. This method can be broken into exact and approximate methods. The exact method divides the space by drawing vertical rays from obstacles' vertices. The approximation method keeps decomposing a rectangle space into identical rectangles till the interior of the rectangle is completely free or the predefined resolution is achieved.

We can see that these methods all require complete and accurate information about obstacles' configurations and locations. After the path is decided, it will not be changed

during navigation. Thus the path planning methods alone cannot react to dynamic environment. We will review various local obstacle avoidance methods in the following section.

2.6.2 Local obstacle avoidance

The robot should have the ability to cope with obstacles detected by sensors in navigation. Usually two objectives of obstacle avoidance should be fulfilled. One is to make the robot to go around obstacles to avoid collision with them. The other is to make robot move toward its target. The second objective will be pursued together with the first one. This will make the robot move to its target safely and quickly.

Many obstacle avoidance algorithms have been invented and applied in real robot navigation, for example, the wall-following method [8] and the edge detection [16]. In some cases, the wall-following method works as an alterative function when the robot is trapped in a local minima situation [13]. The drawback of these methods is that robot needs to know exactly the configuration of the obstacles before deciding the next step. Thus this will consume much time when measuring the obstacles. Because of the limitations of these two methods, they are seldom adopted in current robot systems.

The two main approaches of the methods are Potential Field Method and Steer Angle Field Methods. For obstacle avoidance in a dynamic environment, the robot needs mechanisms different from the static methods. Methods of dynamic obstacle avoidance are included at the end. In the following, we will introduce the methods and their application respectively.

2.6.2.1 Potential Field Methods

In 1985, Khatib published his paper about Potential Field Method [68]. Potential Field Methods solve the problem by assuming that obstacles and target have influence on robot, like magnetism. The influence is materialized as a force. Obstacles will produce repulsive forces on robot. At the same time, the target produces attractive force. In the

field of force, the robot is pushed by these forces. Having properly defined potential functions of repulsive forces from the obstacles and attractive force from the target, the robot will move away from obstacles and toward target automatically. The potential functions can be modified in fluid dynamics and magnetic field forms.

Around the same time, Moravec and Elfes pioneered the concept of certainty grids, a widely popular map representation that is well suited for sensor data accumulation and sensor fusion [89]. By integrating the concepts of potential field and certainty grid, Borenstein and Koren developed the Virtual Force Field (VFF) method [13, 14, 15]. This method is a direct expression of the original potential field method. The robot's motion is decided by the resultant force factor of the repulsive and attractive forces. Though the robot can achieve a maximum travel speed of 0.78m/sec, several limitations exists in this method. As identified in [71, 39], the robot may be trapped to local minima, oscillate between obstacles and narrow passages, and cannot reach the goals with obstacles nearby.

In order to overcome these drawbacks, Borenstein and Koren introduced the Vector Field Histogram (VFH) method in 1991 [17]. Polar obstacle density is designed to calculate the repulsive forces from the obstacles. The VFH+ method is an improved version of the VFH method [129]. It explicitly takes into account the robot dimensions and the trajectory of the mobile robot. The VFH* method employs a four-stage data reduction process in order to compute the new direction of motion. This method is combined with the A* search algorithm to find the optimal path [130].

2.6.2.2 Steer Angle Field Methods

Though the potential field method is good at computing an obstacle-free motion direction, it often fails in controlling the speed of a robot. It is mainly because the potential field method does not include the robot's velocity as a factor for computing collision free path. Different from the potential field method, the steer angular field method will compute the collision free path based on both the obstacles positions and the

velocity of the robot. The main steer angle field methods include the Dynamic Window Approach (DWA) [38] and Curvature-Velocity Method (CVM) [119].

Fox, Burgard and Thrun invented DWA in 1997 [38]. This method starts from searching the velocity space of a robot. Among all possible velocities for the next step, an objective function is used to evaluate them and the velocity with the highest evaluation value is chosen. Brock and Khatib modified the Dynamic Window Approach to holonomic dynamic window approach for holonomic robots, which can instantaneously accelerate in all directions. And this holonomic DWA was integrated with a global planning method to result in the global dynamic window approach [20]. Stachniss and Burgard integrated path-planning techniques with DWA to produce an integrated approach [123]. In this approach, the robot first computes a path. Then the path is used to determine the search space to be explored in the next step. Through the path planning, the path to the target is divided into several parts. For each part, the robot will use the dynamic window approach to achieve the sub-goals, the terminals of each part.

CVM is another steer angle field method, which is introduced by Simmons [119]. Different from DWA, this method has different function for the three factors. It concentrates more on how to get the distance from obstacles. Ko and Simmons invented Lane-Curvature Method (LCM) on the base of CVM [69]. The lanes are constructed by determining the maximum collision-free distance to obstacles along the desired goal targeting. The longest trajectory to each lane is chosen for the objective function. Because CVM may ignore some opening and LCM may neglect some radical openings, Benayas, Fernάndez, Sanz and Diéguez introduced their Beam-Curvature Method (BCM) approach [10]. This method works by integrating beam method with CVM approach. Beams are used as searching intervals instead of lanes in LCM. In the experiments, BCM has the best performance in comparison with CVM and LCM.

2.6.2.3 Dynamic Obstacle Avoidance

A robot moving in a dynamic environment must have the ability of avoiding moving obstacles. In many cases, the moving obstacles will move unpredictably. The obstacle

avoidance methods reviewed so far are all for relatively static environments. In such environments, obstacles are stationary objects or slow moving persons. Though the obstacle avoidance algorithms may divert the robot from the moving obstacles past enough to avoid collisions, they may fail if the speeds of obstacles are high.

Fiorini and Shriller proposed the Velocity Obstacle (VO) concept for the dynamic obstacle avoidance problem [37]. VO consists of velocities that will potentially cause the robot to collide with moving obstacles. Castro, Nunes and Ruano have integrated this VO concept with the dynamic window approach to produce a reactive local navigation method for dynamic environment [25]. The velocity space for the dynamic window approach is the reachable avoidance velocities obtained by using the VO approach. Then a velocity is chosen for the next step from the RAV. This enriches the dynamic window approach's ability in a dynamic environment.

In the VO method, the rotational velocity of the moving obstacle is not considered. As the authors pointed out, several optimal solutions may be omitted because each possible velocity consisting of the searching tree tries to avoid all obstacles. So we have suggested the dynamic map idea for dynamic obstacle avoidance [140]. But our method is based on the assumption that the vessel can accurately predict the moving obstacle's motion. Thus, if the vessel is far away from the obstacle, a small estimation error may cause the obstacle avoidance to fail. However, this possible failure can be compensated when the robot moves near the obstacle, because the computation time based on dynamic map may be short enough and the estimation of the obstacle may be more precise after a longer observation of the obstacle.

CHAPTER

3

PARALLEL BDI AGENT ARCHITECTURE

The traditional BDI agent has 3 basic computational components: generate beliefs, generate intentions and execute intentions. They run in a sequential and cyclic manner. This may introduce several problems. Among them, the inability to watch the environment continuously in dynamic environments may be disastrous. One possible solution is by using parallelism. We propose a parallel BDI model with three parallel running components which are the belief manager, the intention generator and the intention executor. The coordination between the parallel components is done by interrupts of different priorities. The agent built with this architecture has the ability of performing several actions at once. The agent also has the ability to prioritize the deliberations and intention executions so it is able to respond quickly to circumstance changes and all the thinking and acting are done at appropriate times.

In order to evaluate the parallel BDI model, we compare the parallel agent against five versions of sequential agents where the 3 components of the BDI agent are controlled and managed in different ways and different time resources are allocated to the 3 components. Experiments are designed to simulate the operation of the three components in the agents. The ability of the agents to respond to the same sequences of external events of various

priorities is assessed. The comparison results show that the parallel BDI agent has quicker response, react to emergencies immediately and its behaviour is more rational.

This chapter is structured as follows. In the first section, an introduction of background and motivation is given. In Section 3.2, we present the general framework for parallel BDI agents that need to perform in real time. The functions of the processing units in the framework are identified, their operations are defined and how these functional units interact and cooperate is specified. In Section 3.3, simulation experiments are presented to compare the performance of the parallel agent and five versions of sequential agents. A theoretical analysis about the performance of the parallel agent is presented in Section 3.4. In Section 3.5, the issue of how much parallelism and how to configure a parallel agent based on the general framework with a limited number of CPUs are studied by experiments with different configurations of the parallel agent. We describe some advantages and a limitation of the parallel BDI architecture in Section 3.6. A short conclusion is given in the last section.

3.1 Introduction

Hayes-Roth [48] defined the primary objective of an intelligent agent that needs to perform in real time as "to maintain the value of its own behaviour within an acceptable range over time". Among the requirements for an intelligent agent, two related are flexibility (the agent should react to important unexpected events) and timeliness (the agent should meet various real-time constraints). Many agent architectures or frameworks have been developed for building an intelligent agent. As identified in the survey of agent architectures [135], three kinds of agent architectures, deliberative architecture [18, 111, 63, 110], reactive architecture [22, 23] and hybrid architecture [35, 36, 81], are proposed according to the processing mechanism of the agents. The BDI model is well understood for designing deliberative architectures because it combines a respectable philosophical model of human practical reasoning [44]. The reactive architecture, most noticeably, the subsumption architecture, is a different paradigm [21, 22, 23]. The hybrid architecture is

proposed to combine the deliberative and reactive architectures to inherit the advantages of both.

With the different agent models proposed, it is said [44] that the basic components of an agent designed for a dynamic, uncertain world should include some representation of Beliefs, Desires, Intentions and Plans – the BDI model. There are three main operations in this model: detecting, thinking and executing. In normal implementation of the deliberative agent, the three operations run sequentially. For example, in PRS [63], the deliberative process runs in iterations. At the beginning of each iteration step, new goals and new facts are obtained through input. Then several plans in the KA (knowledge area) library are triggered by the new belief and one or more of the applicable plans are selected to be sent to the intention structure. At the end of each iteration step, the intentions are executed. In PRS, the agent will not proceed to the next step until the current step is finished. In a complex and dynamic environment, the agent needs more time to search for proper intentions or one action may need more time to execute. Then more time is needed in each iteration step and the agent is not able to detect new events before the current iteration step is finished. As a consequence of this, the agent may not be able to start processing the emergencies immediately. So the reactivity of the PRS agent cannot be assured in such circumstance.

A possible solution to promote the reactivity of the agents appears in the TouringMachine [35], a well known hybrid agent architecture. The architecture consists of three sequential components: perception subsystem, control framework and action subsystem. The control framework will output actions to the action subsystem based on the sensory input from the perception subsystem. It consists of three layers, a reactive layer, a planning layer and a modeling layer. The outputs from the three layers are summarized by using some context-activated control rules. A clock is used to control the execution time of the control framework. For each time cycle, the control framework has fixed time resource to use. And the primitive schema (action) structure is designed with a ‘cost’ property, which indicates how much time it costs to execute the action. If in a cycle, the remaining time resource is not sufficient for an action to execute, another action needing less time cost

will be executed instead or the remaining time is wasted if no suitable action is available. This method insures that the agent can sense the environment at fixed time intervals. The probability of overlooking emergencies is low if the time spent in detection and processing is balanced well. It is usually required that the detection should not consume much time. If the TouringMachine puts much time on the control framework in a cycle, the problem of poor reactivity still exists.

In agents based on BDI logics, such as AgentSpeak(L) [112] and LORA [136], the problems may appear as reactivity and intention reconsideration issues. The reasoning is done by theorem provers, which usually need much executing time. In LORA, the basic agent control loop of the BDI interpreter consists of perception, updating belief, generating desires, choosing intention and executing actions. Desires, intentions, and actions are generated based on belief. The original circumstance/belief may have changed during these processing. The intentions may become impossible under the new circumstance. The agent should not commit to infeasible intentions. An improvement is made by updating beliefs and reconsidering intentions after executing each action. An experimental result of intention reconsideration by Kinny and Georgeff is provided in the book [136]. The result shows that the more frequently the intentions are reconsidered, the lower the effectiveness of the agent is. Thus, the reactivity of the agent cannot be ensured.

Pokahr [106] suggested that in the sequential BDI agents the concrete layout of the processing cycle will determine the nature of the agent, for example, the caution level and reconsideration rate. And the agent architecture is not easy to be extended with additional facilities because the processing is step by step and very restrictive. The authors proposed a more flexible way of mapping the BDI model to allow easy extension of the agents. In the architecture, the steps are transformed to meta-actions. A main interpreter will decide which meta-action will be selected to execute from the agenda queue. The execution of the meta-action may update the status and insert new meta-actions into the agenda. The extension of new agent abilities can be easily done by designing meta-actions. However, it is noticed that the outside messages are inserted into the agenda directly as external

actions. If perceiving environment is modeled as a regular meta-action, there is no guarantee at the architecture level that the environment is monitored appropriately closely. At the same time, the problems of low caution level and reconsideration rate also remain.

Pokahr [105] commented that the current BDI model does not support any mechanism for handling goal relationships at the architecture level. They proposed a deliberation strategy for agent developers to specify relationships between goals such that there is a maximum number of goals that an agent may pursue at once and the activation of one goal may inhibit another goal. However, an important factor that is not considered is the importance and urgency of a goal that influences which goal should have the attention of the agent first.

Hayes-Roth [48] pointed out that parallel subsystems with buffered communications to provide asynchronous perception, cognition and action will allow an agent to perform in real time. We propose what is required of an agent for real time performance: (1) ability to respond to emergencies timely; (2) ability to modify goals, intentions and actions in reaction to unexpected or new information; (3) ability to perform multiple actions at once (e.g. talking while walking); (4) ability to perceive, deliberate and act simultaneously (e.g. thinking while walking).

In [73], a multi-threaded approach is used to simulate soccer agents for the RoboCup competition. The sensing, thinking and acting behaviours are executed in parallel. Thus the soccer agent does not need to wait for I/O operations (sensor and act) with the world and gains more time for thinking. The experiments show that the agent with a parallel architecture has obvious advantages in lessening the impact of I/O operations in the simulation of an intelligent agent like a human being.

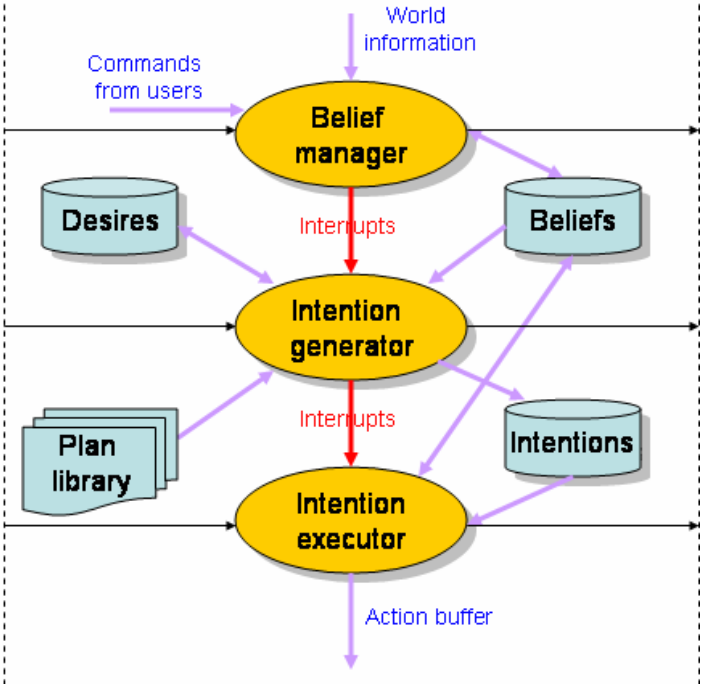


Figure 3-1 Parallel BDI agent model.

In this chapter, we propose a parallel BDI agent framework for real time performance based on the BDI model. The general idea is that such a framework consists of three main components, the belief manager, the intention generator and the intention executor which are running in parallel as shown in Figure 3-1. The horizontal dark thin lines show the control flow in the agent. The three components each consists of a number of smaller processing units and they run in parallel. The coordination between them is done by messages and interrupts of different priorities. The data flows are shown by the purple and red line between the components. The message flows and interrupts are shown by arrows. The belief manager generates beliefs from world information perceived by the agent and human commands given to the agent. The intention generator generates desires, then schedules and reschedules the generation of intention plans for the desires. The intention executor schedules and reschedules the execution of intention plans and executes them. Some parallelism can be achieved by simply running these three components as parallel threads. However, in such set-up there is no pre-emption of less important and urgent desires/intentions by more important and urgent ones so the agent is not able to respond quickly to emergencies. Furthermore, the degree of parallelism is

limited because there is no parallelism in multiple intention generations and multiple intention executions. The general framework proposed offers much better functionalities. Under this general framework, parallel BDI agents with different configurations based on the best way to share the available computational resources may be built. These agents have a number of advantages over the sequential one: 1. they have the 4 abilities required of an agent as discussed earlier; 2. support is provided at the architecture level for reconsideration of desires and intentions and consideration of goal relationships when a new belief/desire is generated.

The idea of parallel operation can be seen in some other agents. However, it is realized in different applications or to deal with different problems from ours. For example, in the designation of subsumption systems [21, 66], the layers of control are run concurrently. In LSA, the layers are realized as the theorem provers. So the reasoning ability of the deliberative agents is combined into the reactive agents. The empirical results from a robot implementation show that the provers can be used without sacrificing much reactivity [5, 6]. This kind of parallel deliberative architecture is different from ours. In LSA, the layers are presumed to work independently. Each parallel layer will perform the actions of detection, reasoning and output sequentially. Output from one layer can be input for another layer. It can be regarded as several deliberative agents each with its own sub-goals running in parallel in the subsumption architecture.

Another example of parallel operation can be seen in JAM [60]. The JAM agent can execute some *action_sequences* in a plan simultaneously. This means that some actions in an intention can be performed concurrently. This parallel execution of some actions is also different from our work. In our work, the three basic behaviours of the agent, detection, deliberation and execution, are parallelized. The agent can be watching, thinking and acting at the same time. Together with the interrupt mechanism in the agent, this parallel BDI agent architecture can solve the problem of concrete layer in traditional sequential agents. The reactivity of the agent can be improved to real-time level.

3.2 The General Framework

We propose a general framework for parallel BDI agents based on the parallel BDI agent model shown in Figure 3-1. The framework is shown in Figure 3-2. The arrow lines in the figure show the control flow among the processing elements of the agent. The framework can be useful when designing a robot agent. Each device is a processing element which can be run on a processor. It can also be used for agent-based simulation of a physical system that is capable of parallel actions. An example of software agent representing a vessel captain who can watch, think and act simultaneously is presented in Chapter 6.

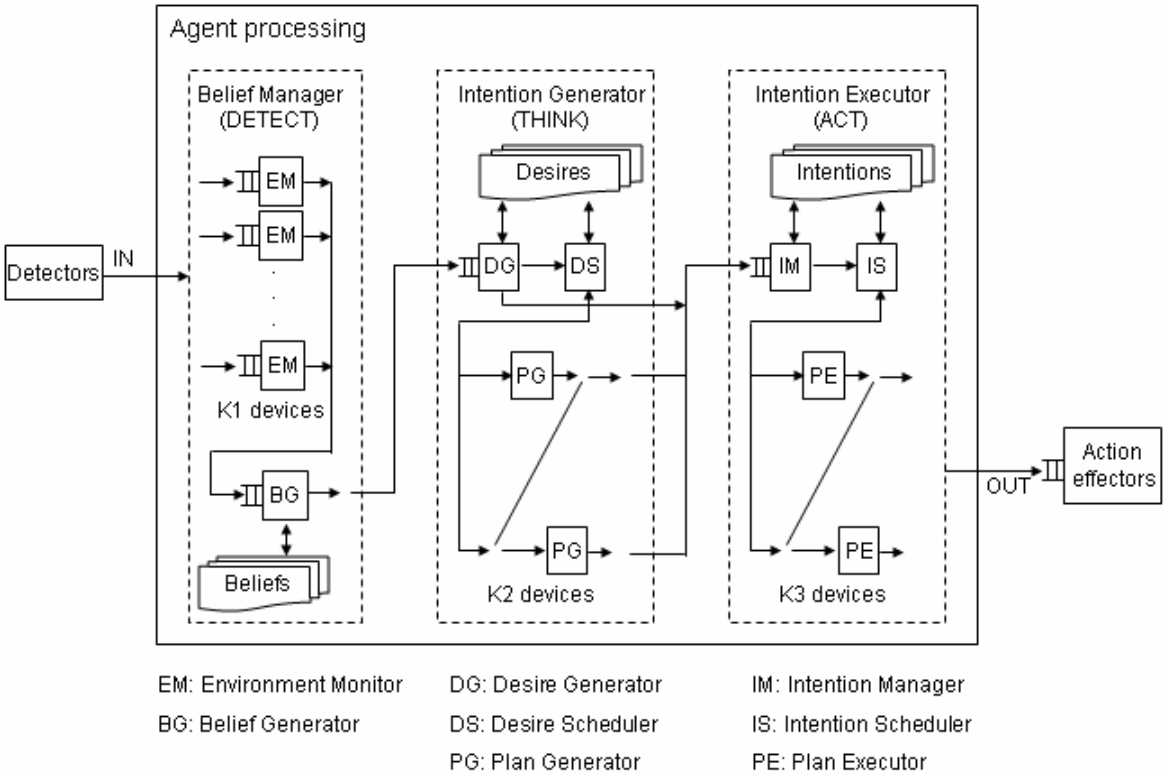


Figure 3-2 The General Framework for Parallel BDI Agents.

The framework consists of three main components: the belief manager, the intention generator and the intention executor. These three components represent the three steps in the deliberation process of an agent: *detect*, *think* and *act* respectively. The three components will retrieve and update data in the three data structures: *beliefs*, *desires*, and

intentions. The *beliefs* contain the agent's view of its environment and of itself. The *desires* will maintain the goals of the agent. The *intentions* store the plans to be executed to achieve the agent's goals. Each of the three components are further divided into smaller processing elements, also shown in Figure 3-2. We define the functionalities of each processing element and the interactions among them.

3.2.1 Belief manager

The belief manager is responsible for detecting the changes in its environment and managing the agent's view of the world and of itself. It is made up with Environment Monitors (EMs) and a Belief Generator (BG). The set of EMs serve as the information collectors from heterogeneous sensors that an agent may have. Each EM monitors world information from one type of sensors or sensory organs like a camera or human eyes and converts the information into abstract representation. Each EM sends the converted information to the BG.

The BG will merge the various information items passed from the EMs into the agent's view of the world. For example, the eyes of a person see and the ears hear a car coming. The visual and audio information will come through two separate EMs and the BG combines the information to form a new belief. This is a cognitive process where the semantics of the sensor information is worked out. Each new belief has a certain degree of urgency. The BG will determine the urgency of the new belief. When a new belief is generated, existing contradictory or obsolete beliefs will be removed. In other words, when b_{new}, a new belief is formed, we have

$beliefs = beliefs \cup \{ b_{new} \}$ and
$\exists b \in beliefs[\text{obsolete}(b_{new}, b)] \rightarrow beliefs = beliefs - \{b\}$

Messages will be produced by the BG to notify the intention generator. Depending on the urgencies of the new beliefs, message of different priority levels will be generated so that the intention generator may process them accordingly. This allows urgent information to be handled immediately and less urgent information be dealt with later. As the belief

manager works simultaneously with the intention generator and the intention executor, this allows the agent to monitor its environment including any emergencies at all times.

3.2.2 Intention generator

The intention generator manages the agent's desires (goals), based on the agent's current beliefs. This includes generation of new goals, removal of obsolete goals and suspension of existing goals. The intention Generator also deliberates on the plans to achieve these goals. It is made up with a Desire Generator (DG), a Desire Scheduler (DS) and a number of Plan Generators (PGs).

The DG may produce new desires after a new belief is added into *beliefs*. The new desire will have a priority level according to these rules:

If $b_1 \wedge b_2 \wedge \ldots \wedge b_n \Rightarrow d$ then $pd = \max(pb_i)$, $i = 1 \ldots n$.

If $b_1 \vee b_2 \vee \ldots \vee b_n \Rightarrow d$ then $pd = \max(pb_i)$,

$i = 1 \ldots n$, and $b_i \in beliefs$

These rules mean that when a number of beliefs b_1 to b_n conjunctively trigger a desire d, the priority of the desire pd, will have the highest priority level of the beliefs that triggered its generation. When a desire d may be triggered by any one of a number of beliefs $b_1 \ldots b_n$, the priority of the desire will have the highest priority level of the belief among $b_1 \ldots b_n$ that the agent currently believes in.

A new belief may make a current desire no longer desirable because it is obsolete or it is not consistent with the new desire. A new belief may also affect the priority of an existing desire. After the generation of a new desire, the new desire may affect an existing desire or be affected by an existing desire with respect to their priorities. The intention generator will handle these cases as described later.

The set of desires of an agent *desires* consists of three subsets:

$desires = pendingD \cup planningD \cup executingD$

where

pendingD: the set of desires waiting for their plans to be worked out or a partial plan for the desire has been decided but more deliberation is needed,

planningD: the set of desires being planned, that is, a plan to achieve the desire is being worked out in a PG,

executingD: the set of desires whose intention plans are waiting to be executed or being executed by the intention executor.

If $d_i \in desires$ then $d_i = (ID_i, g_i, p_i, s_i)$ where ID_i is the identifier of the desire, g_i is the desired goal of the agent, p_i is the priority of g_i representing its urgency and s_i represents the status of deliberation about this goal.

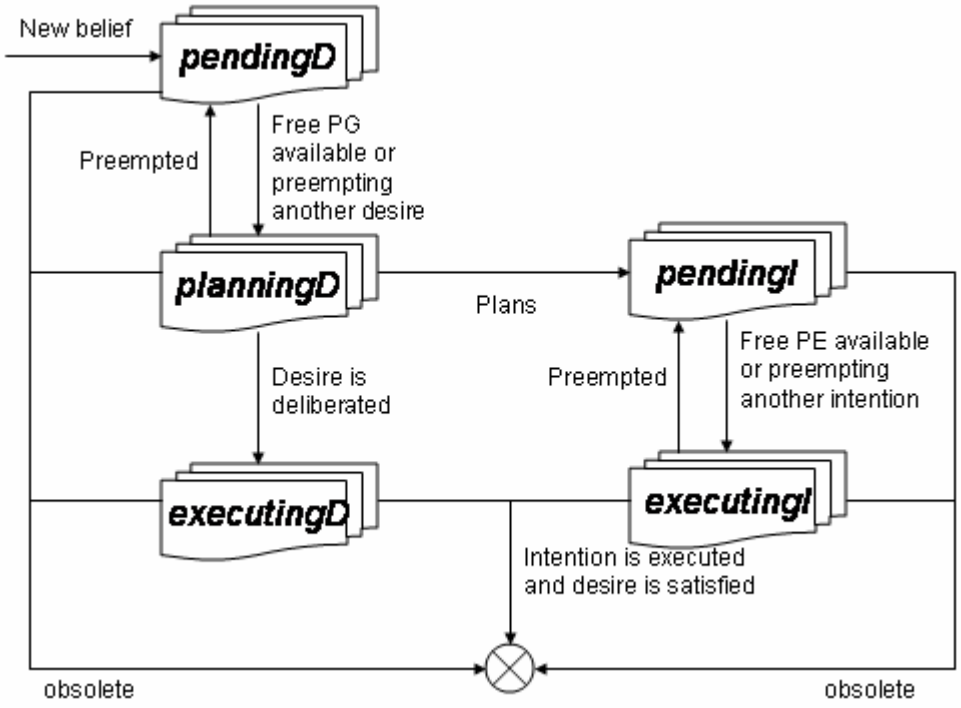


Figure 3-3 States of desires and intentions and their transition.

With the partition of *desires* into *pendingD*, *planningD* and *executingD* as shown in Figure 3-3, we provide a simple way to handle the reconsideration of goals. When a desire is generated by the DG, it is deposited into the *pendingD*. A desire will move from *pendingD* to *planningD* when the Desire Scheduler (DS) decides that this desire should

preempt another desire in *planningD* or there is a free PG to do planning and this desire has the highest priority in *pendingD*. A desire will be moved from *planningD* to *pendingD* by DS when this desire is preempted by another desire in *pendingD* or a new desire in *pendingD* calls for the suspension of this desire in *planningD*. A desire moves from *planningD* to *executingD* when the intention plan for the desire has been generated. A desire will be removed from *pendingD*, *planningD* or *executingD* if a new belief or a new desire makes the agent abandon this desire.

We define the operations of DG in Figure 3-4 and those for DS in Figure 3-5. In Figure 3-4, obsolete(a, d) means belief/desire a makes desire d obsolete. urgencyAffected(a, d) means belief/desire a changes the urgency of desire d or desire d changes the urgency of desire a. This may result in an increase or a decrease of priority for d or a (if a is a desire), or a temporary suspension of d or a. Suspension of a desire happens when an agent decides to shelf the desire temporarily because of the conflicts between a new desire and an existing desire. clash(d_1, d_2) in Figure 3-5 returns true if d_2 was suspended (priority set to suspensionPriority) because of d_1. The function preempt(p_k, p_i) in Figure 3-5 will decide whether d_k should preempt d_i. For example, suppose d_k is the desire with the highest priority in *pendingD*, d_i is the desire with the lowest priority in *planningD* and preempt(p_k, p_i) = $p_k > p_i$. Then at all times, we have

$\forall i \forall j, (ID_i, g_i, p_i, s_i) \in pendingD, (ID_j, g_j, p_j, s_j) \in planningD\ [p_i \leq p_j]$

It is possible to have other more elaborate definitions for preempt(p_k, p_i), for example, p_k > (p_i - some threshold) and (remaining time for d_i > some threshold).

LOOP:
  b_{new} = the belief with maximum priority among the newly generated beliefs in *beliefs;*
  Process the new belief b_{new} and form a new desire d_{new}, if appropriate;
  If $d_{old} \in desires$ and (obsolete(b_{new}, d_{old}))
    If $d_{old} \in pendingD$
      $pendingD = pendingD - \{ d_{old} \}$.
    If $d_{old} \in planningD$
      Set the priority of d_{old} to removalPriority such that DS will remove it from a PG.
    If $d_{old} \in executingD$

Notify IM through interrupts to have the corresponding intention removed.
If $d_{old} \in desires$ and (urgencyAffected(b_{new}, d_{old}))
If $d_{old} \in pendingD \cup planningD$
Change the priority of d_{old} to a higher/lower value or suspensionPriority.
If $d_{old} \in executingD$
Notify IM through interrupts to change the priority of the corresponding intention.
If a new desire, d_{new}, is formed
$pendingD = pendingD \cup \{ d_{new} \}$.
If $d_{old} \in desires$ and (obsolete(d_{new}, d_{old}))
If $d_{old} \in pendingD$
$pendingD = pendingD - \{ d_{old} \}$.
If $d_{old} \in planningD$
Set the priority of d_{old} to removalPriority such that DS will remove it from a PG.
If $d_{old} \in executingD$
Notify IM through interrupts to have the corresponding intention removed.
If $d_{old} \in desires$ and (urgencyAffected(d_{new}, d_{old}))
If $d_{old} \in pendingD \cup planningD$
Change the priority of d_{old} or d_{new} to a higher/lower value or suspensionPriority.
If $d_{old} \in executingD$
Notify IM through interrupts to change the priority of the corresponding intention.
If there is change in membership in *pendingD* or *planningD*
Inform DS by message.

Figure 3-4 Operations of DG.

LOOP:
1. **Scheduling desires according to the priorities of desires.**
LOOP1:
$p_k = \max_{d_j \in pendingD} (p_j)$;
If P_k equals null or suspensionPriority
break LOOP1.
If there is no free PG
$p_i = \min_{d_j \in planningD} (p_j)$;
If preempt(P_k, P_i)
$pendingD = pendingD \cup \{ d_i \}$;
$planningD = planningD - \{ d_i \}$;
Send an interrupt to the PG to suspend d_i;
else

break LOOP1.
If there is a free PG
pendingD = *pendingD* - { d_k };
planningD = *planningD* ∪ { d_k };
Send an interrupt to the PG to start d_k.

2. **Handling the completion of the planning of a desire by a PG.**
For each desire $d_p \in$ *planningD*
If the planning of d_p is finished in a PG
planningD = *planningD* - { d_p };
executingD = *executingD* ∪ { d_p };
If $d_{old} \in$ *pendingD* and clash(d_p, d_{old}))
Increase the priority of d_{old} such that d_{old} may be considered for plan generation.
Send a message to IM with the information about d_p.

3. **Updating the priorities of desires.**
For each desire $i \in$ *desires*
Update the priority of d_i according to DG instructions or by other factors, like time;
If priority of d_i equals removalPriority
If $d_i \in$ *executingD*
Notify IM through interrupts to have the corresponding intention removed;
else
desires = *desires* - { d_i };
If $d_i \in$ *planningD*
Send an interrupt to the PG to remove d_i.
If priority of d_i equals suspensionPriority
If $d_i \in$ *planningD*
planningD = *planningD* - { d_i };
pendingD = *pendingD* ∪ { d_i };
Send an interrupt to the PG to suspend d_i.
If $d_i \in$ *executingD*
Notify IM through interrupts to have the corresponding intention suspended.

Figure 3-5 Operations of DS.

3.2.3 Intention executor

The intention executor works in a similar way as the intention generator. Intention Manager (IM) will receive the plan information from the PGs in the intention generator. Intention Scheduler (IS) will schedule/suspend/resume the running of intentions in the

Plan Executors (PEs). Finally, the physical actions will be put into the action effectors and the PE will remove the finished desire *d* from *executingD*. The agent's *intentions* consists of three subsets of intentions:

$$intentions = inactiveI \cup pendingI \cup executingI$$

where

inactiveI: the set of intentions that are waiting for the completion of other intentions before they can execute and intentions the agent has to put on hold temporarily,

pendingI: the set of intentions waiting to be executed,

executingI: the set of intentions being executed.

If $i_i \in intentions$ then $i_i = (ID_i, predeccessors_i, peers_i, plan_i, p_i, s_i)$ where ID_i is the identifier, $plan_i$ is the action plan, p_i is the priority, s_i represents the status of the intention, $predeccessors_i$ and $peers_i$ are used to address the dependency issues between different intentions and facilitate synchronization among intentions and is explained in the following paragraphs.

Normally, an intention plan may be a sequence of primitive actions to be carried out by the agent one after another. Or it may be a composite hierarchical task structure where predecessor subtasks need to be completed first and peer subtasks need to be executed simultaneously. If two subtasks are not related as predecessor-successor or as peer, they may be executed in any order or in parallel. Such a composite hierarchical task structure is transformed into several intentions by the PG which generates the intention plan in Intention Generator. The transformed intentions have a *predecessors* attribute to name the predecessor intentions which has to be completed before its own execution. They also have a *peer* attribute to identify what other intentions are to be executed simultaneously with them. An example of the transformation is shown in Figure 3-6.

In the example in Figure 3-6, the original intention plan is transformed into 6 intentions with smaller action plans. Normally, each sequence of actions, like the one presented by Action a, is transformed into a new intention. If some sequences of actions are to be executed one by one, like Actions d and f, it may be possible to combine them into one

intention like Intention 4, provided that both d and f may be executed by the same PE or the PEs are homogeneous. This reduces the cost for scheduling. In the rectangles showing the intentions 1-6, the first pair of bracelets shows the *predecessors* of the intention. For example, intention 6 cannot start before both intentions 4 and 5 are completed. With the transformation, the original intention plan can exploit parallelism supported in the agent's framework. An example is that intention 2 (Action b) and intention 3 (Action c) may be executed at the same time if two PEs are available. This speeds up the execution of intentions and the parallel framework of the agent is used more effectively. Intentions 4 and 5 are peers so they have to be executed in 2 PEs simultaneously. The synchronization issue among peer intentions is discussed in the next section.

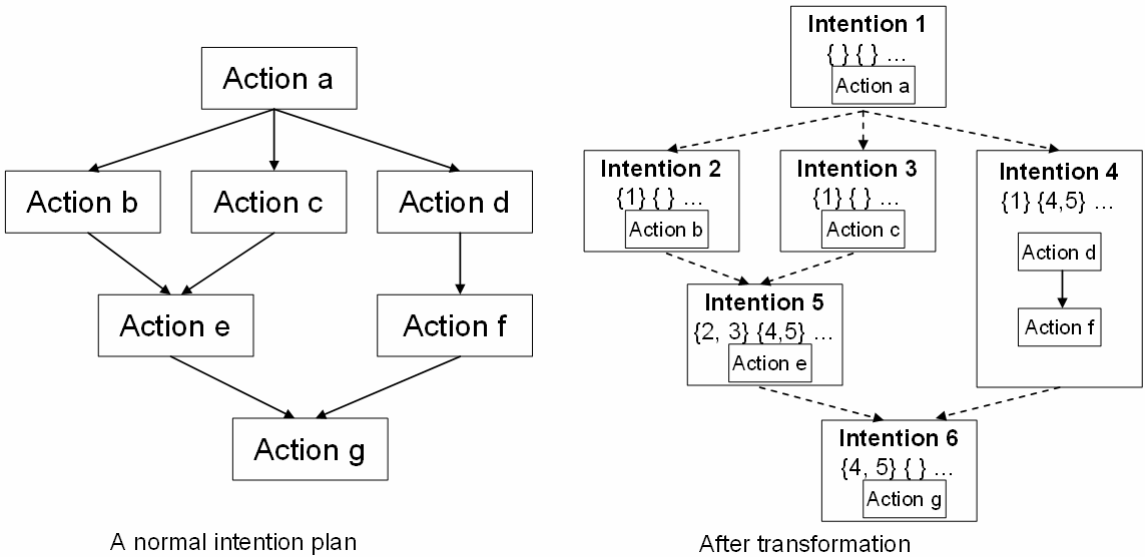


Figure 3-6 Transformation of a normal intention plan.

As shown in Figure 3-3, *intentions* are partitioned into *inactiveI, pendingI* and *executingI*. This supports the scheduling and the reconsideration of intentions. An intention plan in *inactiveI* is one that can only start execution after the completion of its predecessor intentions or it is one that the agent wants to put on hold for the moment. An intention plan in *pendingI* or *executingI* has the same meaning as a desire in *pendingD* and *planningD* respectively. When a new intention plan arrives at IM, it joins *pendingI* if it has no predecessor intentions otherwise it joins the *inactiveI*. For each intention in *inactiveI*, IS checks whether all the predecessor intentions are completed. If yes, it

moves this intention from *inactiveI* to *pendingI*. In this way, scheduling intentions with predecessors are easily managed by updating their *predecessor* attributes and organize them in *inactiveI* and *pendingI*.

The operations of IM and IS are defined in Figure 3-7 and Figure 3-8 respectively. Functions with the same names in Figure 3-7 and Figure 3-8 as those in Figures 3-4 and 3-5 behave in the same way.

With this framework, an agent developer just needs to concentrate on specifying domain specific definitions of obsolete desires, urgent desires and clashing desires. The desires and intentions will be activated, deactivated, suspended or removed automatically.

LOOP:

1. **Handling the message, m_{new}, from DG.**

If $i_{old} \in$ *intentions* and (obsolete(m_{new}, i_{old}))

If $i_{old} \in$ *inactiveI*

$executingD = executingD - \{ d_{old} \}$;

$inactiveI = inactiveI - \{ i_{old} \}$.

If $i_{old} \in$ *pendingI*

$executingD = executingD - \{ d_{old} \}$;

$pendingI = pendingI - \{ i_{old} \}$.

If $i_{old} \in$ *executingI*

Set the priority of i_{old} to removalPriority such that IS will remove it from a PE.

If $i_{old} \in$ *intentions* and (urgenceAffected(m_{new}, i_{old}))

Change the priority of i_{old} to a higher/lower value or suspensionPriority.

2. **Handling the message, m_{new}, from DS.**

Form a new intention, i_{new}, based on m_{new};

If i_{new} and its *peers* intentions have no *predecessors* intentions

$pendingI = pendingI \cup \{ i_{new} \}$;

else

$inactiveI = inactiveI \cup \{ i_{new} \}$.

If $i_{old} \in$ *intentions* and (obsolete(i_{new}, i_{old}))

If $i_{old} \in$ *inactiveI*

$executingD = executingD - \{ d_{old} \}$;

$inactiveI = inactiveI - \{ i_{old} \}$.

If $i_{old} \in$ *pendingI*

$executingD = executingD - \{ d_{old} \}$;

$pendingI = pendingI - \{ i_{old} \}$.

If $i_{old} \in$ *executingI*

Reduce the priority of i_{old} to removalPriority.

If $i_{old} \in$ *intentions* and (urgencyAffected(i_{new}, i_{old}))
  Change the priority of i_{old} or i_{new} to a higher/lower value or suspensionPriority.
Inform IS the priority of an intention has changed.

Figure 3-7 Operations of IM.

LOOP:
1. **Scheduling intentions according to the priorities of intentions.**
LOOP1:
$p_k = \max_{i_j \in pendingI} (p_j)$;
If P_k equals null or suspensionPriority
  break LOOP1.
K = {intention *k* and its *peers* intentions};
If there is no enough free PE(s) for intentions in K
  n is the number of PEs needed if K can get executed;
$p_i = \max_{i_j \in I} (p_j)$
where I = { *n* intentions in *executingI* with the lowest priorities, note that priority of peer intention have the same priority}.
If preempt(P_k, P_i)
  pendingI = *pendingI* $\cup$ I;
  executingI = *executingI* - I;
  Send interrupts to the PEs to suspend the execution of intentions in I.
else
  break LOOP1.
If there is enough free PE(s) for intentions in K
  pendingI = *pendingI* - K;
  executingI = *executingI* $\cup$ K.
Send interrupts to the PEs to start execution of intentions in K.
2. **Handling the completion of the executing of an intention by a PE.**
For each intention $e \in$ *executingI*
  If the executing of i_e is finished in a PE:
  executingD = *executingD* – { d_e };
  executingI = *executingI* - { i_e };
  If $i_{old} \in$ *pendingI* and clash(i_e, i_{old}))
    Increase the priority of i_{old} such that i_{old} may be considered for execution.
3. **Updating the priorities of intentions.**
For each intention $i \in$ *intentions*
  Update the priority of i_i according to IM instructions or by other factors, like time.
  If priority of i_i equals removalPriority
  executingD = *executingD* - { d_i };
  intentions= *intentions* - { i_i }.
  Send an interrupt to the PE to remove i_i
  If priority of i_i equals suspensionPriority and $i_i \in$ *executingI*

executingI = *executingI* - { i_i and its *peers* intentions};
inactiveI = *inactiveI* ∪ { i_i and its *peers* intentions}.
Send an interrupt to the PE to suspend i_i and its *peers* intentions.

4. **Checking *predecessors*.**
For each intention $i \in$ *intentions*
If $i_i \in$ *pendingI* and a new *predecessor is added* to it
pendingI = *pendingI* - { i_i and its *peers* intentions};
inactiveI = *inactiveI* ∪ { i_i and its *peers* intentions}.
If $i_i \in$ *executingI* and a new *predecessor is added* to it
executingI = *executingI* - { i_i and its *peers* intentions};
inactiveI = *inactiveI* ∪ { i_i and its *peers* intentions}.
If $i_i \in$ *inactiveI* and all *predecessors* of it and its *peers* intentions are completed or removed
inactiveI = *inactiveI* - { i_i and its *peers* intentions};
pendingI = *pendingI* ∪ { i_i and its *peers* intentions}.

Figure 3-8 Operations of IS.

3.2.4 Synchronization among peer intention plans

With multiple PEs, the agent is able to carry out multiple actions simultaneously. Some actions need to be synchronized and some are completely asynchronous. Synchronization is needed among peer intention plans and we classify the synchronization into the following three forms.

1) Time stepped synchronization. The execution of intention plan P in one PE has to be at the same 'speed' as those of several other intention plans in several other PEs. IS can play the centralized coordinator in controlling the PEs.
2) Barrier style synchronization. The execution of several intention plans in several PEs may progress asynchronously until a certain 'barrier' then they synchronize with each other and continue. This can be managed in the same way as the barrier synchronization in parallel discrete event simulation. Again, IS can coordinate.

Execute intention plan P1 in PE_1 while intention plan P2 is being executed in PE_2. IS start P1 and P2 in the 2 PEs at the same time and it will terminate P1 in PE_1 as soon as P2 in PE_2 has completed execution.

3.2.5 General Remarks

The operations define how the devices in the belief manager, the intention generator and the intention executor work to process the incoming events. All the devices work in parallel. The interrupt mechanism ensures that an emergency can be dealt with first. Thus, the agent obtains the ability of quick reaction to emergencies and the capacity for careful deliberation when required. With the parallel components, the agent can handle several matters at once. The agent is also able to 'change his mind' towards his desires/intentions according to the changing environments. The requirements for real time performance, as we proposed in Section 3.1, are satisfied. One method to realize the operations is by priority control, which will be discussed in Chapter 5. In next section, we make a comparison between the sequential BDI model and the parallel BDI model.

3.3 Comparison between the Parallel BDI Model and the Sequential Ones

In this section, we evaluate the performance of the parallel agent by comparing it with the sequential agents. There are 3 main or coarse computational components in a BDI agent, the belief manager, the intention generator and the intention executor. In a sequential agent, only one computational component is running at any time. However it is possible to control and manage the 3 components in several different ways in an attempt to get better performance from a sequential agent. On the parallel BDI agent side, under the general framework the maximum parallelism can be realized by having all the processing elements, like EMs, BG, DG, running in parallel. To demonstrate that parallel BDI agents constructed according to the general framework are able to offer the benefit of parallel actions, we have a conservative parallelism where only the 3 main components, i.e. the belief manager, the intention generator, and the intention executor, operate in parallel. This means the processing elements in the same component will run sequentially. We first describe the five sequential BDI agents each with their own way of controlling and managing their computational components. Then the simulation experiments are presented and the results are analyzed.

3.3.1 Sequential BDI agents

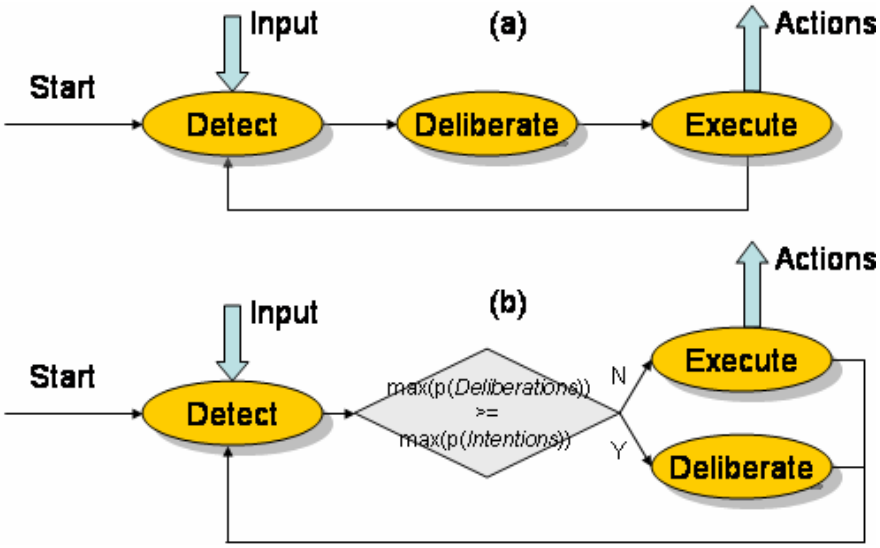


Figure 3-9 Sequential BDI agents.

The five kinds of sequential BDI agents are:

(1) As shown in Figure 3-9a, the 3 components run in a cyclic way and each uses up the pre-allocated and fixed time resource. The deliberation/intention cannot be suspended and resumed. If the remaining time of a component (only the deliberate and the execute components) is not sufficient for a deliberation/intention to be finished, the remaining time will be wasted.

(2) This is a variant of agent 1. It suspends a task when the time allocated to the current component is used up and resumes it when the component's turn comes in the next cycle. For example, the execute action can start an intention which costs 5 time units when there is only 1 time unit remaining.

(3) A more flexible way is to allocate time resources to the deliberate and execute components only when needed. If a component has nothing to do, it terminates and the next component starts. In order to keep the agent vigilant, the detect component always uses up all its allocated time. The actual time used for deliberate/execute should not exceed the maximum pre-allocated time to such a component. This agent has a cycle time ranging from a minimum that is the fixed time for the detect component to a maximum that is the sum of the allowable times of the 3 components. The tasks cannot be suspended.

(4) Different from agent 3, the tasks can be suspended.

(5) This agent has a cycle as shown in Figure 3-9b, in each cycle, after the detect component, the agent will choose to deliberate or execute based on the maximum priority of deliberations and intentions. After deliberate/execute is finished, another cycle begins. This makes the agent more watchful for emergencies.

The characteristics of the five sequential BDI agents are summarized in Table 3-1. In all the sequential agents, when there is more than one deliberation/intention to handle in the *deliberate/execute* component, the one with highest priority will be processed first. The performance of these agents will be compared with the parallel BDI agent.

Table 3-1 Sequential agents

Agent no	Flexible time allocation?			suspend-resume?	Illustration
	detect	deliberate	execute		
1	N	N	N	N	Figure 3-9 (a)
2	N	N	N	Y	Figure 3-9 (a)
3	N	Y	Y	N	Figure 3-9 (a)
4	N	Y	Y	Y	Figure 3-9 (a)
5	N	Y	Y	Y	Figure 3-9 (b)

With the different time allocation schemes for the three components, the sequential agent will show different performances. In the experiments, we used three time allocation schemes for sequential agents according to their emphasis on the three components. For a maximum cycle of 15 time units, three schemes showing the fixed or maximum allowable time quota for each component of the BDI agent are given in Table 3-2.

Table 3-2 Allocation schemes

Configuration	detect	deliberate	execute
C1	1	4	10
C2	3	4	8
C3	5	3	7

The sequential agent with configuration C1 puts more emphasis on executing intentions with the risk of overlooking emergencies. C3 gives more time to the detect component to be more vigilant. But the time for the execute component is cut. C2 is a compromise between C1 and C3.

Each sequential agent described in Table 3-1 will be configured according to C1, C2 and C3 respectively in the experiments to compare them with the parallel agent.

3.3.2 The input data

The evaluation of the sequential and parallel agents is done by simulation of the processing of events by agents. All the sequential agents and the parallel agent will process some sequences of events. Each event will be processed by the 3 computational components of the BDI agent, namely, the belief manager(*detect*), the intention generator(*deliberate*) and the intention executor(*execute*).

In the experiment, the system time is represented as continuous time units. There is a system clock to control the increase of the time. The system time is started from 0. For a vessel agent, the events may include new topological findings, nearby obstacles, and user commands. According to the details of these events, the priorities of the events and the costs in time used to execute the corresponding plan can be decided. In this simulation, we will discard the actual details of the events. Only the processing time of the events and the priority are used to identify an event. These properties are related to the analysis. To evaluate the agent ability to handle events of different importance or urgencies, events will have one of the four different priority levels 1 to 4, with 4 being the highest. We assume that an event can be detected and a belief generated in 1 time unit and each deliberation to generate an intention takes 1 to 3 time units. The intention execution time of events at all priority levels is uniformly distributed in the range from 1 to 7 time units. So the average deliberation execution time is 2 time units and the average intention execution time is 4 time units. This also means the average time required to handle an event is 7 (1+2+4) time units.

We use the exponential density function to represent the inter-arrival time between any two events. As shown in [113], the exponential density function is memoryless and often

used to model lifetimes or waiting times. The cumulative distribution function is shown as:

$$\mathrm{cdf}(t) = 1 - \mathrm{e}^{-\lambda t} \qquad 3\text{-}1$$

where $1/\lambda$ is the mean, and t is the time units.

cdf(t) shows the probability that the inter-arrival time between 2 events is less than t. So given a total number of events, *sum*, to be used in our experiments, we can decide the number of intervals with length of n time units by:

$$\mathrm{G}(n) = \left(\mathrm{cdf}(n) - \mathrm{cdf}(n-1)\right) * sum \qquad 3\text{-}2$$

The intervals are kept in a vector. Then the intervals are selected randomly. The current time plus the interval length is the arrival time of the next event. The event priority is selected randomly from 1 to 4.

We consider three sequences of events with different average inter-arrival times. The average inter-arrival times of the 3 sequences of events are respectively smaller than, equal to and larger than the average processing time required by an event. The events statistics used in the experiments are shown in Table 3-3.

Table 3-3 Events statistics

Set	Expected average interval 1/λ	Events count					Actual average interval	Average deliberation time	Average intention time
		Priority				sum			
		1	2	3	4				
a	4	262	250	235	251	998	4.48	1.99	3.96
b	7	240	260	247	247	994	7.31	1.97	4.08
c	15	268	244	239	234	985	14.58	1.98	3.89

When an event arrives and the agent is not doing detection, the event will be stored in an event buffer. The agent can retrieve the new events later. After receiving the event, the agent will create one *deliberation* for it. The *deliberation* will be added in a deliberation queue. The *deliberation* and *intention* plan selected later will have the same priority as the event.

3.3.3 Comparison results and analysis

We will use the *response time* to evaluate how well the agent processes the event. The *response time* is defined as the time between the arrival of the event and end of the execution of the intention plan chosen for the event. The response time is calculated as the sum of the time for detecting the event, deliberation and execution. The overhead transmit messages between the three parallel components in the parallel agent is omitted as it is felt that the delay in passing the interrupts is very small.

The results of the experiments are presented in Table 3-4. ART is the Average Response Time of all events. ART_p stands for the ART of the events with priority p. ART_w is the weighted ART by the priorities of the events.

$$ART = \frac{\sum_{i=1}^{Count} T_i}{Count},\ ART_w = \sum_{p=1}^{4} \frac{ART_p * p}{1+2+3+4} \qquad 3\text{-}3$$

where T_i is the reaction time for event i, and $Count$ is the number of events.

Parallel agent vs sequential agents

Looking at the response times of the parallel agent and the various sequential agents (rows with agent nos. 1 to 5), we can see that the parallel agent responds at least 3 times faster than the sequential agents in most the cases. This shows that the parallel agent can process events quicker than the sequential ones. Especially for the events set *a* and *b*, which are in high-density for the processing ability of the sequential agents, some ARTs of them reach over one thousand. Normally it is very irrational to complete a task 1000 time units later after the event happened. So in such environments, the sequential agents are not applicable. The main reason for this difference comes from the fact that the parallel agent uses 3 times CPU time as the sequential agents do. So in order to get comparable ARTs, we should prepare for the sequential agents a single CPU which is 3 times as powerful as the CPUs used in the parallel agent. In most cases, it is more applicable and economic to prepare 3 less powerful CPUs for a parallel agent.

Table 3-4 ART of the events by the agents.

Set	Config.	Agent no	ART_p				ART	ART_w
			1	2	3	4		
a	C1	1	3731.59	1577.26	90.22	22.41	1401.62	724.64
		2	4443.1	1621.04	64.09	23.66	1593.54	797.21
		3	2706.14	927.34	58.8	19.54	961.49	481.54
		4	3300.26	635.3	42.51	20.86	1040.8	478.18
		5	3318.06	651.88	38.66	20.87	1048.72	482.13
	C2	1	4416.69	2114.99	144.25	23.12	1729.08	917.19
		2	4581.51	1818.47	73.33	26.18	1682.14	854.32
		3	4046.55	1735.6	115.54	22.51	1529.96	795.44
		4	4496.36	1599.33	71.64	25.2	1604.25	801.07
		5	4531.08	1651.86	63.06	23.13	1623.98	811.65
	C3	1	6523.12	3832.43	656.97	27.9	2834.22	1627.05
		2	6621.4	3990.95	573.05	32.56	2881.14	1645.27
		3	5959.58	3193.37	371.31	23.75	2457.89	1355.53
		4	6017.61	3415.7	252.62	28.57	2502.08	1372.12
		5	6112.63	3523.51	312.36	26.88	2567.67	1420.42
	Parallel		48.81	17.59	11.18	7.75	21.8	14.85
b	C1	1	961.05	78.29	28.69	18.4	264.22	127.73
		2	456.77	50.65	26.58	20.26	135.17	71.88
		3	68.91	26.33	18.15	13.45	31.38	22.98
		4	65.3	27.65	19.14	15.74	31.67	24.1
		5	73.74	28.82	20.72	15.81	34.42	25.68
	C2	1	2177.75	132.21	35.53	19.48	574.07	262.67
		2	1075.42	73.53	33.39	23.32	292.98	141.59
		3	1051.26	86.14	29.45	17.5	288.02	138.19
		4	903.78	78.73	34.39	23.95	253.31	126.02
		5	1015.03	62.61	31.53	21.31	274.58	132.01
	C3	1	4520.94	728.04	57.62	19.52	1301.18	622.8
		2	4814.97	408.03	44.98	24.77	1286.63	586.5
		3	3688.76	459.78	45.62	18.09	1026.74	481.75
		4	4109.05	208.17	39.25	23.95	1062.28	473.89
		5	4590.59	267.35	37.85	22.12	1193.23	532.73
	Parallel		13.12	10.0	8.58	7.35	9.74	8.83
c	C1	1	34.46	25.52	20.15	16.4	24.48	21.16
		2	27.16	22.91	19.26	16.65	21.69	19.73
		3	11.19	9.73	8.79	9.03	9.73	9.32
		4	11.11	9.61	8.96	9.18	9.76	9.39
		5	14.5	12.14	10.41	9.59	11.76	10.84
	C2	1	44.23	28.39	21.44	16.93	28.29	23.3
		2	31.07	23.59	20.99	17.61	23.57	21.16
		3	17.63	13.29	11.49	10.69	13.41	12.14
		4	16.19	13.64	11.82	11.09	13.28	12.32
		5	27.78	20.31	16.24	13.88	19.83	17.27
	C3	1	76.7	34.8	22.0	15.69	38.55	27.5
		2	51.63	31.1	22.37	17.95	31.44	25.27
		3	35.13	20.25	14.54	12.06	20.97	16.75
		4	29.42	19.42	15.31	13.85	19.82	16.96
		5	41.46	26.29	18.28	15.89	26.0	21.24
	Parallel		8.92	7.75	7.52	6.87	7.8	7.45

One thing to note is that in a few cases the parallel agent responds less than 3 times faster than the sequential ones. This happens with sequential agents 3, 4, 5 when the event sequence is set *c*. ARTs and the weighted ARTs of agents 3, 4 with configuration C1 and C2 are less than 2 times those of the parallel agent. This means when the inter-arrival time is long enough, the parallel agent will not show its advantage. Because the inter-arrival time is 14.58, the sequential agents have sufficient time to process the incoming events. Though configuration C3 is allocating too much time to the *detect* component of the agent, the performance of the sequential agents with C3 is not too bad. However, the parallel agent is still valuable to process the real emergencies with the highest priority. As seen in Table 3-4, ART_4 to process the events set *c* is 6.87 for parallel agent and 9.03 for the sequential one with best performance.

The priority 4 events are the highest priority events in the experiments. The average time needed to process one such event in the ideal case (ATN_4) is calculated by:

$$ATN_4 = \frac{\sum_{i=1}^{Count_4} (TDetection_i + TDeliberation_i + TExecution_i)}{Count_4} \qquad 3\text{-}4$$

where $TDetection_i$ is the time used to detect the event *i*, $TDeliberation_i$ is the time used to deliberate the event *i*, $TExecution_i$ is the time used to execute the plan generated based on the event *i*, and $count_4$ is the number of events with priority 4.

In this experiment, *TDetection* equals to 1. *TDeliberation* and *TExecution* can be gotten from the events list as shown in Table 3-3. Using the equation, ATN_4 equals to 7 for set *a*, 7.02 for set *b*, 6.79 for set *c*. Compared to the ART_4(column 7 in Table 3-4) we can see that the parallel agent spends just 0.75 time units more for set *a*, 0.33 time units more for set *b* and 0.08 time units more for set *c*. Set *a* is a sequence of events with an inter-arrival time smaller than the processing time required for an event. So the intention generator and the intention executor are busy handling other beliefs and intentions when a high priority belief/intention comes to them. Here the interrupt mechanism in the parallel agent is able to guarantee immediate handling of higher priority items while the

sequential agents are not able to do this. It is clear that the parallel agent has a big advantage over the others on processing real emergencies.

For events with lower priority, the difference of ART between the sequential agents and the parallel agent is bigger.

Different time resources allocation in the sequential agents

Looking at the ART and ART_w columns and comparing the corresponding rows for configurations C1, C2 and C3, we conclude that the performances of the sequential agents with configuration C3 are significantly worse than those with configuration C1 and C2. This shows that the sequential agents perform badly if they spend more time on detecting and less time on deliberation and intention execution. The processing of emergencies is often postponed, though the emergencies are detected earlier in configuration C3. This can be seen that in most cases the processing of the highest priority events also have longer response time. This indicates that in real life, the agent is not reacting to high priority events quickly and is taking a longer time to react to other events.

We also observe that the performance of the sequential agents with configuration C1 is significantly better than C2. This shows that the sequential agents perform much better if they spend short time on detecting and more time on deliberation and intention execution. Because the deliberating and executing components get more time resources, the beliefs and intention plans get cleared faster so the events experience shorter response time.

Different ways of controlling the computational components in the sequential agents

Looking at the ART and ART_w columns and comparing the corresponding rows among the sequential agents, we see that agent 1 and 2 are the losers in all cases. This is expected because of their rigid way of controlling the *detect*, *deliberate* and *execute* components. In the best performing configuration C1, agents 3, 4 and 5 have comparable performance in all event sequences a, b and c. So we conclude that if a component has nothing to do, it is better to give way to the next computational component.

For set *b* and *c* in configurations C2 and C3, generally agents 3 and 4 are better with 4 being the best more often but agent 5 is not the loser every time. We may conclude that it is better to allow the computational components to start processing an item if there is one, then suspend it when its time quota is up and resume in the next round. But this policy is not always effective. At first, it is interesting to notice that the ART_4 is normally better for the agent 3 than agent 4 (Just one exception is found: set *a* Configuration C1). This is because that the above police will cause a lower-priority task is being dealt with when a higher-priority event enters. The sequential agent has no interruption schema, so the new task will not be executed immediately. In Table 3-4, we can see that the ART_4 is from 1 to 5 time units bigger for agent 4 than agent 3. This also affects the ART_3 and ART_2 in some cases. Another interesting phenomenon is that for set *a*, even the ARTs are mostly worse for the agents with the suspend-resume mechanism (2, 4) than the agents without such mechanism (1, 3). It is because in agents 1 and 3, the execution of the tasks is not strictly scheduled according to the policy that the higher-priority tasks get processed first. If the current remaining time-slot is not sufficient to execute a task with high-priority, the low-priority task with suitable execution time will be processed earlier. Then the tasks with shorter execution time are prone to be processed first. For a high-density events set, as the set *a*, such fact will decrease the average waiting time of the tasks. So ART is increased. It is suggested to make verification before adopting this policy.

3.4 Theoretical Analysis

In the following, theoretical analysis is carried out to predict the performance of the parallel agent when some attributes of the incoming events sequence are known. The estimated ART is shown as ART_e. The following facts are assumed:

- The inter-arrival time of the events is exponentially distributed and only one event can happen in any time unit;
- The time to process an incoming event and generate a belief is 1 time unit;
- The processing time for a deliberation is chosen randomly in [1, D_{max}];
- The time to execute an intention is chosen randomly in [1, E_{max}];

- The priority value for an intention is chosen randomly in [1, P_{max}];
- The overheads for transferring data between the devices are very small compared to the time needed for processing the event sequence and ignored.

The following symbols are used in the subsequent discussion. The values of them can be gotten given an event sequence.

N – number of events;

$\overline{interval}$ - average interval between two events;

$\overline{S_1}$ - average time needed to process an incoming event and generate a belief;

$\overline{S_2}$ - average processing time needed to complete a deliberation job;

$\overline{S_3}$ - average processing time needed to execute an intention job;

The performance of the agent is analyzed when an event sequence is input for processing. The agent performance is evaluated by ART for the events in the serial. A smaller ART shows that the agent can process the events quicker. The ART_e can be calculated by:

$$ART_e = \overline{WT_1} + \overline{S_1} + \overline{WT_2} + \overline{S_2} + \overline{WT_3} + \overline{S_3}$$

Where:

$\overline{WT_1}$: average WT_1, waiting time between the occurrence of an event and its detection;

$\overline{WT_2}$: average WT_2, waiting time between the generation of a deliberation, that is, the insertion of the deliberation into the queue and the start of the deliberation;

$\overline{WT_3}$: mean of WT_3, waiting time between the generation of an intention, that is, the insertion of the intention into the intention queue and the start of intention execution.

The following figure is used to show the data flow in the agents.

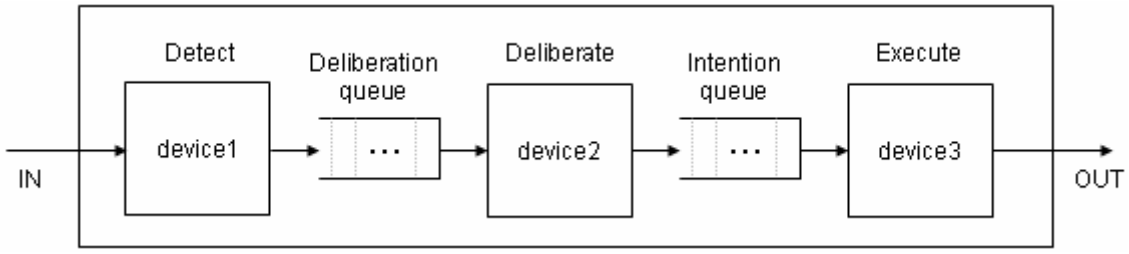


Figure 3-10 The data flow in the agent.

First, the total time needed to process all the N events can be estimated by $T = N * \max\left(\overline{S_1}, \overline{S_2}, \overline{S_3}, \overline{interval}\right)$. Because device1 runs at all time and it can detect a new event immediately after the event happens and $\overline{interval} \geq S_1$ for all events, therefore $\overline{WT_1} = 0$. There will be two cases when estimating the ART_e according to the different $\overline{interval}$.

Case 1: $\overline{interval} >= \max\left(\overline{S_2}, \overline{S_3}\right)$

If $\overline{interval} >= \max\left(\overline{S_2}, \overline{S_3}\right)$, it is expected that the devices in the parallel agent have sufficient time to process an event in the interval between two events. The waiting time for services will be 0. So the ART_e equals to $\overline{S_1} + \overline{S_2} + \overline{S_3}$. This can be verified by the statistics in the previous experiments. The event sequences, *a*, *b* and *c*, shown in Table 3-3 are used as examples to show the analysis.

Table 3-5 Experiment statistics

Set	$\overline{interval}$	ART in experiments	$ART_e = \overline{S_1} + \overline{S_2} + \overline{S_3}$
a	4.48	21.8	6.96
b	7.31	9.74	7.05
c	14.58	7.8	6.87

From the table, it can be seen that the difference between the actual ART and ART_e is smaller if $\overline{interval}$ is larger. This is because with a larger $\overline{interval}$, there are few cases that the interval between two events is smaller than $\max\left(\overline{S_2}, \overline{S_3}\right)$. In most cases, the agent can complete processing the previous events before the next one comes in. Then the actual ART is closer to the expectation.

Compared to sets *b* and *c*, there are more cases in set *a* that the interval between two events is actually smaller than $\max(\overline{S_2}, \overline{S_3})$. So the agent may not have enough time to complete processing the previous intentions, which makes the actual ATR (21.8) much longer than the expectation (6.96).

Case 2: $\overline{interval} < \max(\overline{S_2}, \overline{S_3})$

If $\overline{interval} < \max(\overline{S_2}, \overline{S_3})$, the estimation will be done using the following equation. We use *b* to show the index of devices in which the average processing time is larger among the device2 and device3 as shown in Figure 3-10. Thus, $\overline{S_b} = \max(\overline{S_2}, \overline{S_3})$. So device *b* is the bottleneck.

After the arrival of the mth event, the number of tasks waiting to be processed in device *b* on average is:

$$N_{remaining} = m * \left(1 - \frac{\overline{interval}}{\overline{S_b}}\right)$$

The highest priority of the remaining tasks is:

$$P_{high} = \frac{N_{remaining}}{\frac{m}{P_{\max}}} = P_{\max} * \left(1 - \frac{\overline{interval}}{\overline{S_b}}\right)$$

Thus, we can see that the probability that a task can be executed immediately is:

$$\text{Pro}(p > P_{high}) = 1 - P_{high}/P_{max} = \frac{\overline{interval}}{\overline{S_b}}.$$

where p is the priority of the task. The $\overline{interval}$ bigger, the possibility is bigger. A blocked task is expected to get executed after all the events are inserted. The number of remaining tasks when all the events are inserted is $N_{rN} = N * \left(1 - \frac{\overline{interval}}{\overline{S_b}}\right)$. Then the average waiting time for execution in the device *b* is:

$$\overline{WT_b} = \left(1 - \frac{\overline{interval}}{\overline{S_b}}\right) * \left[\left(\frac{1}{N} * \sum_{i=1}^{N}(N-i) * \overline{interval}\right) + \frac{N_{rN} - 1}{2} * \overline{S_b}\right]$$

$$= \left(1 - \frac{\overline{interval}}{\overline{S_b}}\right) * \left(\frac{N-1}{2} * \overline{interval} + \frac{N_{rN} - 1}{2} * \overline{S_b}\right)$$

ART_e can be estimated as $ART_e = \overline{S_1} + \overline{WT_b} + \overline{S_2} + \overline{S_3}$. We designed three experiments to empirically validate this equation. The results are shown in Table 3-6. It can be seen that the ART_e is quite close to the real ART.

Table 3-6 Experiments statistics

N	$\overline{S_2}$	$\overline{S_3}$	$\overline{interval}$	ART in experiments	ART_e
999	1.98	4.01	1.09	1468.53	1463.68
999	2	4.01	2.05	1014.92	981.78
998	2	3.88	3.01	460.74	439.38

3.5 How Much Parallelism

In Section 3.2, we proposed that when an agent is simulating a certain physical system, the parallel agent should be configured such that it has the same number of Environment Monitors (EMs), Plan Generators (PGs) and Plan Executors (PEs)as the number of parallel physical devices that exist in the physical system to perform the corresponding functions. In this section, we consider agents that are not simulating a physical system. In this case, the constraint is the parallelism that can be supported by the physical computer. For example, if a computer has 2 CPUs, only 2 processing elements can be running in parallel. However, given a fixed number of CPUs, there is still the issue of how to distribute the CPU power to the processing elements. For example, 6 CPUs can be used by the PGs and PEs, how do we decide whether to have 2 PGs and 4 PEs or 3 PGs and 3 PEs? In this section, we present experimental results on agents with different configurations based on the general parallel BDI agent framework. The agents will process a sequence of events. The events sequences are designed in the same way as in Section 3.3. The statistics about the sequences used are shown in Table 3-7.

Table 3-7 Events statistics

Set	Expected average interval 1/λ	Events count					Actual average interval	Average PG time	Average PE time
		Priority				sum			
		1	2	3	4				
a	2	244	286	234	235	999	2.54	1.99	4.16
b	4	270	237	255	236	998	4.48	1.99	4.01
c	7	235	244	264	251	994	7.31	2.01	3.98

For simplicity, we assume there is only one type of sensor input so there is just one EM (*K1* in Figure 3-2 is 1). The following are assumed in the experiment:

1. The incoming event can be detected and beliefs updated in 1 time unit. So processing time in BG is 1 time unit.
2. The DG takes 1 time unit to generate a desire.
3. Each deliberation to generate an intention takes 1 to 3 time units, uniformly distributed. This is the processing time in PG. So the expected average processing time in a PG is 2 time units.
4. The intention execution time of events at all priority levels is uniformly distributed in the range from 1 to 7 time units. This is the processing time in PE. And the average is 4 time units.
5. The times used in DS, IM, IS are significantly shorter than the processing times in other processing units like BG, DG, PG etc and can be ignored.
6. The overhead of transmitting messages between the various processing units in the parallel agent is not included as it is assumed that the agent is running on a machine with multiple processors sharing memory. So the delay in passing the interrupts is very small.

All these assumptions mean the expected average time required to process an event is 8 (1+1+2+4) time units.

The results of the experiments with different combinations of *K2* and *K3* devices(refer to Figure 3-2) are presented in Table 3-8.

The priority 4 events are the highest priority events in the experiments. The average time needed to process one such event in the ideal case (ATN_4) is calculated as the sum of the detection time (1), desire generation time (1), average PG time and average PE time for all the events with priority 4. In the events sequence used, ATN_4 equals to 8.01 for set *a*, 8.28 for set *b*, 8.0 for set *c*. Compared to the ART_4 in Table 3-8 we can see that the parallel agent spends just a little more or the same amount of time for processing the events with priority 4. This confirms that the interrupt mechanism in the parallel agent is able to guarantee immediate handling of higher priority items.

Table 3-8 ART of the events by the agents

Set	*K2*	*K3*	ART_p				ART	ART_w
			1	2	3	4		
a	1	1	2364.78	813.83	18.64	9.26	817.12	408.54
	1	2	22.32	11.8	8.89	8.17	12.84	10.53
	2	2	18.21	10.28	8.4	8.06	11.25	9.62
	2	3	9.12	8.59	8.01	8.01	8.45	8.24
	2	4	8.49	8.35	7.99	8.01	8.22	8.12
	3	3	9.0	8.55	8.0	8.01	8.41	8.21
	3	4	8.4	8.28	7.97	8.01	8.17	8.09
b	1	1	49.33	17.36	11.3	8.84	22.45	15.33
	1	2	9.77	9.07	8.36	8.34	8.91	8.64
	2	2	8.87	8.52	8.13	8.28	8.46	8.34
	2	3	7.99	8.03	7.96	8.28	8.06	8.1
	2	4	7.84	8.0	7.96	8.28	8.01	8.08
	3	3	7.96	8.04	7.96	8.28	8.05	8.1
	3	4	7.82	7.99	7.96	8.28	8.0	8.08
c	1	1	14.35	10.84	9.85	8.31	10.77	9.88
	1	2	9.02	8.2	8.27	8.05	8.37	8.24
	2	2	8.47	7.97	8.09	8.01	8.13	8.07
	2	3	8.12	7.81	8.05	8.0	8.0	7.99
	2	4	8.08	7.81	8.05	8.0	7.99	7.99
	3	3	8.11	7.81	8.05	8.0	7.99	7.99
	3	4	8.07	7.81	8.05	8.0	7.98	7.98

In the following, we will show the waiting time for deliberation and execution. Average waiting time for deliberation (AWTD) means the time a desire spent in *pendingD* waiting. This includes the time before the plan generation is started and the time when the plan generation for this desire is suspended. AWTE, average waiting time for

intention execution, is defined in a similar way. A shorter waiting time means that the processing is quicker.

The AWTD is only related to *K2*, the number of PGs. Table 3-9 confirms that a larger *K2*, the deliberations can be finished quicker. In the three sets of environments, 3 PGs are enough to provide the agent the ability to deliberate any event immediately after it is received.

Table 3-9 Average waiting time for deliberation

Set	*K2*	$AWTD_p$				AWTD
		1	2	3	4	
a	1	5.95	2.13	0.62	0.13	2.24
	2	0.09	0.09	0.01	0.0	0.05
	3	0.0	0.0	0.0	0.0	0.0
b	1	1.26	0.74	0.31	0.07	0.61
	2	0.03	0.01	0.0	0.0	0.01
	3	0.0	0.0	0.0	0.0	0.0
c	1	0.63	0.25	0.19	0.04	0.27
	2	0.01	0.0	0.0	0.0	0.0
	3	0.0	0.0	0.0	0.0	0.0

AWTE is affected by both *K2* and *K3*(the number of PEs). The statistics is shown in Table 3-10. With a same *K3*, the agent with a larger *K2* can produce intentions earlier. So in such case, the AWTE may be increased. But referring to Table 3-8, we can see that the total ART is decreased. It is easy to see that with *K2*=1 and *K3* increased from 1 to 2, the AWTE is greatly decreased. Because in an event-congested environment like set *a* (interval=2.54), the agent with 1 PE cannot process all the intentions in time.

Table 3-10 Average waiting time for execution

Set	*K2*	*K3*	$AWTE_p$				AWTE
			1	2	3	4	
a	1	1	2350.52	803.46	10.04	1.12	806.74
	1	2	8.06	1.43	0.29	0.03	2.45
	2	2	9.81	1.94	0.41	0.05	3.06
	2	3	0.72	0.25	0.03	0.0	0.25
	2	4	0.09	0.01	0.0	0.0	0.02
	3	3	0.69	0.3	0.03	0.0	0.26
	3	4	0.09	0.03	0.0	0.0	0.03
b	1	1	40.27	8.64	3.04	0.5	13.84
	1	2	0.71	0.34	0.1	0.0	0.3

	2	2	1.05	0.53	0.16	0.01	0.45
	2	3	0.16	0.04	0.0	0.0	0.05
	2	4	0.02	0.0	0.0	0.0	0.01
	3	3	0.17	0.05	0.0	0.0	0.06
	3	4	0.02	0.0	0.0	0.0	0.01
c	1	1	5.65	2.78	1.61	0.27	2.51
	1	2	0.32	0.14	0.03	0.01	0.12
	2	2	0.39	0.16	0.04	0.01	0.14
	2	3	0.04	0.0	0.0	0.0	0.01
	2	4	0.0	0.0	0.0	0.0	0.0
	3	3	0.04	0.0	0.0	0.0	0.01
	3	4	0.0	0.0	0.0	0.0	0.0

The overall performance of the agent is decided by the slowest processing unit of the system. In the parallel agent, plan generation and intention execution are the most time consuming. The balance between *K2* and *K3* should be:

$$\frac{K2}{K3} = \frac{AVG_d}{AVG_e} \qquad 3\text{-}5$$

where AVG_d is the average PG time, and AVG_e is the average PE time.

Tables 3-8 and 3-10 confirm that having *K2* = 2 and *K3* = 4 is better than having *K2* = 3 and *K3* = 3, more so when the arrival rate of events is high. If the average inter-arrival time between events equal to AVG_i, the agent needs $K2 = \frac{AVG_d}{AVG_i}$ PGs to generate plans to prevent build-up of desires in *pendingD*. In the situation where the arrival of events are very dynamic, that is, the inter-arrival time changes drastically, the best solution will be for the Desire Scheduler(DS) and the Intention Scheduler(IS) to dynamically adjust the number of PGs and PEs in response to the changes in the arrivals of events. In other words, the DS and IS should self organize what is the best ratio of PGs to PEs.

If there is at most 1 event coming in 1 time unit, maximum *K2* and *K3* needed should be smaller than the maximum PG and PE time respectively. For example, in the experiment, maximum PG time is 3, so 3 PGs are enough to ensure that waiting time for a PG is 0 and AWTD will as a result be 0.

In general, if it is critical to keep the waiting time for PG to zero, the number of PGs required can be calculated by

$$K2 = \lceil b/a \rceil \qquad 3\text{-}6$$

where a is the minimum time interval between two events, and b is the maximum processing time for PG to generate an intention plan for a desire.

This also applies to the value of *K3* where a is the minimum time interval between the arrival of two intention plans produced by PGs and b is the maximum processing time for PE to execute an intention plan for a desire. With these values for K2 and K3, whenever a new desire is generated or a new intention plan is generated, there is always a PG or a PE available to process them. It can be stated that the minimum time interval between the arrivals of two intention plans produced by PGs is the smaller value between the minimum inter-arrival time of events and the minimum processing time for PG to generate an intention plan for a desire.

Under a relative static situation where the demand for deliberation power and that for plan execution power are not changing drastically, the computing power of the agent can be allocated to reflect the demands using this simulation method. But in a more dynamic situation, a dynamic approach will be needed. Learning algorithm will be a good way to decide the number of EMs, DGs and PEs dynamically based on feedbacks of agent performance.

3.6 Possible Advantages and a Limitation

This parallel model should be helpful in the research of continual planning. Continual planning means that the agent will be continuously planning, interleaving planning with execution, because its plans can undergo continual evaluation and revision [30]. For a parallel agent, it will be able to continue planning while executing an intention plan. Several techniques have been produced for continual planning. For example, the continuous planning and execution framework (CPEF) [90] integrates HTN planning

technique [34, 33] to implement open-ended reasoning. Open-ended planning process allows the agent not to generate full level plans before execution. In UM-PRS [77], the hierarchy of the plans is kept for monitor plan execution and replanning. Obviously, the techniques for continual planning increase the needs for time resource. In order to replan, it is necessary for the agent to detect new situations frequently. The idea of parallelizing the basic behaviours of the agent is helpful to support continual planning: the new situations can be detected quickly. More, in a parallel agent, a high-level intention can be subdivided into several sub-intentions. The problems of resource sharing and coordination of the sub-intentions may be solved by utilizing some parallel algorithm. Thus, the intention can be finished quickly and computation resource is used wisely.

Another possible advantage of the architecture is that it can provide the agent some adaptive behaviour by combining automatic learning algorithms for some special problems. Adaptive ability is an important attribute for agents to show the autonomy and proactiveness properties [44]. With adaptive ability, the agents can respond to dynamic environments more intelligently. The agents can improve performance continually without human interference. Many mature learning algorithms have been produced and are utilized in the machine learning areas [88]. But in the plan-based architectures, such as PRS, it is hard to combine the learning algorithms within the reasoning process. An experimental step was taken in [46]. The learning is implemented by applying the Top-down induction of decision trees on the agent's action models. The models are labeled with success or fail tag. The models are organized as the decision trees. In the situations with fixed action steps, the agent can interact with environment with past experiences. But for agents working in continuous environments the limitation is obvious: the models may be too voluminous to save. We have proposed to extend the original BDI model by adding an experience function library [141]. Some complicate algorithms can be coded in this library and called. Combining that extension and the parallel architecture, it is possible to incorporate learning algorithms as experience functions. For example, in a vessel agent, it is possible to implement the obstacle avoidance function with the Q-learning algorithm. The agent can accumulate and take advantage of experience gained through its moving. Using the traditional sequential BDI architectures, the agents may not

be able to react to emergencies in time when calling an experience function. However, the parallel BDI agents have the abilities of detecting emergencies immediately, suspending some low-priority executing experience functions and resuming them at a proper later time. We will see details about the experience function library in next chapter.

A problem for implementing the parallel architecture is that it needs more processors because multi-threads will demand more CPU resources. In our implementation of vessel agents, one agent consists of about 15 threads. This is not surprising if we consider how many little thinking and controlling processes are working in parallel in a human body but it requires a lot of system resources. A system with multi-CPUs will be very useful to have the activities of the agent run in a real concurrency and the responsiveness of the agent can be simulated better.

3.7 Conclusions

In this chapter, we show our proposal for a parallel BDI agent architecture. In the architecture, the three basic behaviours of the BDI agent are parallelized. With the parallelism, the agent obtains the improved ability to work in dynamic environment. It is also a more natural way of working: the three behaviours of an agent are running concurrently.

A comparison experiment between sequential BDI agents and the a parallel BDI agenis shown and a theoretical analysis of the aerformance of the a parallel BDI agenis made. Then the aroblem of how to araocate the computation resource to the devices is discussed. At the end, the aossible utilities of the a parallel BDIrchitecture and a limitation when applying the framework are discussed.

CHAPTER

4

AGENT CHARACTER

As have discussed in the first chapter, vessels navigation control is affected by human and natural factors. Each vessel agent has its own character, which does not have proper representation in the BDI model. We aim to improve the BDI model by incorporating the components into the BDI model to realize the agent character. In this chapter, we analyze the effect of human character and propose an extended BDI architecture for designing human-like agent. Different agent behaviour is a result of: 1. different initial parameter setting; 2. different experience from reinforcement learning. In experiment, a vessel captain is built based on this architecture. Cautious captain, adventurous captains and the like can be created with different parameter settings and experience accumulated through its individual navigation.

Touschapter is structuredas follows. In the first section, we give an introduction about the background. In Section 4.2, we analyze the agent character and explain the two basic components of the agent character. In Section 4.3, an extended BDI agent architecture with character components is illustrated, and the implementations of the agent character are explained within this architecture. The experiment of implementing the agent is shown in Section 4.4. A conclusion is given in Section 4.5.

4.1 Introduction

Most of the previous agent architectures are designed to provide the agents with rational abilities to detect, deliberate and act. Thus, in the same situations, different agents will all make the same decisions and they will have the same behaviour. However, in multi-agent simulation system this is not always desirable. Multi-agent simulation is widely used to enhance knowledge in real worlds and provides the possibility to create artificial worlds for the testing of theories [55]. In multi-agent simulation of human society, agent character is essential for simulating various human beings. The agents will not always work in an ideal and optimal way. Their characters will affect their decisions. For example, in a system which is used for risk analysis by simulating vessels in sea, the vessel agents must show different characters. This is because different captains demonstrate different navigation approaches. Human characters have very important effects on vessel navigation. In order to have meaningful conclusion from the simulation system, different vessel behaviours must be simulated realistically. So the vessel agent should demonstrate human-like character. The agent character should be considered as an important factor when designing real agents.

In this chapter, we analyze the agent character from the agent itself and propose an alternative way to implement the character in the agent architecture. Different from Norling's paper, in which the character is researched from its cause in psychological explanation [97], the character is identified by its effects. The agent character influence will be divided into personality influence and experience influence. We argue that the way people behave is affected mainly by two factors: (1) their personality that seems to come from birth and; (2) the previous life experience of the person. The personality influence shows the agent's initial natures, for example, some babies are more talkative than others and other babies are natural introverts. Another example of this is two twin brothers after going through the same education will still behave differently. Different personalities will be realized as different parameter settings and priority libraries. The life experience comes from the interaction between the agent and the environment. The good

and bad experience from the environment will affect the agent's character and future behaviours. The experience is realized by the reinforcement learning algorithm. These character influences are incorporated into BDI agent architecture. And a vessel agent representing a vessel captain navigating in sea is created using the architecture. Experiment results show that the agents are able to demonstrate different behaviours based on their different characters.

4.2 The Analysis of Agent Character

An agent's uniqueness is called the agent's character. In the following, the agent character is analyzed with the example of a vessel doing navigation. To simulate the navigation behaviours of different vessels, each vessel agent must have its own distinct character. A vessel's distinct character can be seen from its physical specifications and its captain's reasoning behaviour, both of which will have influence on the decision results for navigation. This means, a vessel agent should also have two such kinds of influences on its decision making. A vessel' physical specification is the vessel's physical properties, including the vessels' size, maximum acceleration, maximum translational velocity, and so on. The reasoning behaviour of a captain is determined by the captain's behavioral and mental characteristics and his experience in navigation. The three factors are analyzed one by one.

The physical properties will affect agents' decision result. The influence will be that the agents have to decide the output actions according to their physical capabilities. These physical capabilities do not change with time or the experience of the vessel captain. These are the unchanging factors in the decision making process of the captain.

A person's behavioral and mental characteristics form his personal character. This will have a big impact on the person's behaviour. Each captain is more inclined to make certain decisions. For example, some captain is more inclined to overtake another vessel when being blocked. Some captains are born to be more meticulous than the average and others are born to be more adventurous than others. The difference will result in that a

bold captain is more willing to take a bigger risk in navigation compared to a meticulous or cautious captain. The vessel agent is made to simulate a captain to make decision for navigation. So each vessel agent should have the behavioral and mental characteristics that will affect the whole reasoning process.

Experience in navigation comes from past navigations. Different pasts provide different experiences. Different experiences will make different people. So an expert captain will outperform a green one. A captain who accustoms to navigate in one sea area may not be good in another area.

Of the three factors that affect a captain's decision making, the physical properties have the similar effect as the captain' behavioral and mental characteristics, in that both are not changing with time or experience. From another point of view, we can assume that the physical properties affect the agent' unique personality first, and then they influence the agent's decision making through the personality. So to simulate the agent' decision process, we combine these physical properties with the agent's unique behavioral and mental characteristics. We call it the personality of an agent. To summarize, an agent's character can be divided into two aspects: personality and experience as shown below:

$$\textit{Agent character}\begin{cases}\textit{Personality}\begin{cases}\textit{Physical properties}\\ \textit{Behavioural and mental characteristics}\end{cases}\\ \textit{Experience}\end{cases}$$

The applications of personality and experience are shown in the following individually.

4.2.1 Personality

Personality is the basic element of an agent's character. It will affect every process in decision-making, including the results of experience's accumulation and application. For example, two people with different personalities behave differently even after going through the same experience. The influence of the personality is shown in Figure 4-1. The agent will first detect the world through sensors/input devices/sensory organs. Then

the collected data will be transformed into the world model, which will provide data for decision. The world model is a representation of how the agent personally perceives the world. Then the decision process will make decision based on the world model and personal tendencies. Identified by the index numbers in Figure 4-1, the personality's influence will be shown in two processes:

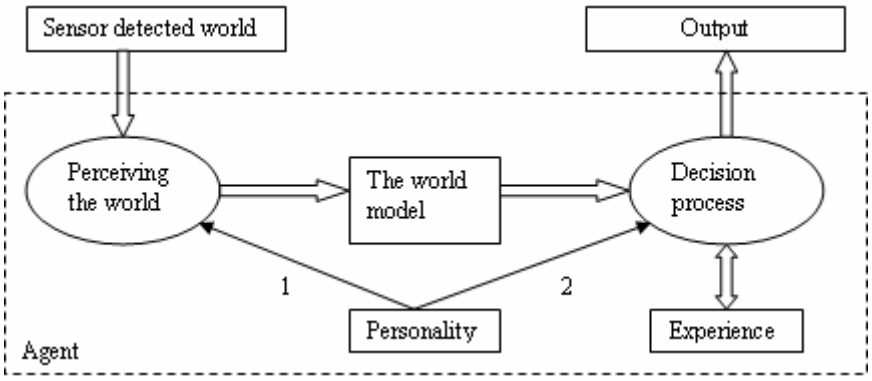


Figure 4-1 Effects of personality.

1. The personality will affect how the agent personally perceives the world.
Humans will have different feelings about the same scene. For example, someone will feel that a vessel is still far away and there is no need to worry about it but other people may feel differently with the same scene. And the decisions are made based on the feelings. So we will build a world model to represent the 'perceived world' (beliefs). The world model consists of the information about the real world.

2. The personality will affect the decision process.
The decision process will deliberate on the choices of plans of actions based on the perceived world. The personality will affect these choices. For example, some vessel captain depends on the past experience more than others. A demonstration of this is shown in Section 4.4.2, where different parameter settings as different personalities for vessels will affect the vessel's final actions.

The personality will also affect the experience's gain and its application in the decision process. One way an agent learns from the experience is using the reinforcement learning

algorithm. Reinforcement learning is completed based on the rewards from the world after each action. Each pair of state and action will have a reward value assigned. The next action will be chosen with the maximum future reward value. And after execution, the value will be updated with the latest rewards. Here, the personality will have two affects:

- The personality will first affect how previous experience is used. There are people who dare to make decisions that they have never made before and people who will only make decisions according previous experience. The learning experience gained will be very different. In other words, the action selection mechanism will be affected by the personality.
- It will affect how previous experience is remembered. The result of an action for a certain state may be viewed differently by different personalities. Some people may feel a certain result is horrible and the action should be avoided completely in future and other people may feel it is still tolerable. These will be reflected by different reward functions

4.2.2 Experience

People with normal intelligence will learn and adapt their behaviour to the environment. So a vessel agent should also have the learning capability and adapt the navigation behaviour based on experience. Normally, the experiences of the vessel captains are not identical and can vary significantly. Two identical vessels under the control of two captains with similar personalities may have different past experiences. They may be trained by different people in different seas and they may have accumulated different working experiences. So vessel agents may be trained using different scenes from different environments.

The human experience is used to make optimal decision (optimal decision is subjective) when facing the similar scenes which have occurred. Human will accumulate his experience along the processes. So the algorithm to simulate the human experience must obtain the two attributes: reusable and accumulative. The reinforcement learning

algorithm is applied to simulate the accumulation of the experience of agents. It qualifies the attribute demanding:

- Reusable

The policy based on reinforcement learning is made by choosing from the value of current state and future action tuples. The action which produces maximum future value is chosen as the next action. This procedure is similar to that human always choose the action which will generate maximum expected reward based on experience.

- Accumulative

After each execution of an action, the agent will update the values of state and action tuples based on the reward from the world. This is also similar to the behaviour that human will modulate his mechanism based on the reaction from the previous actions.

The scenes which can be applied with experience must have the recurring and easy-to-remember features. The recurring feature ensures that human can accumulate experience for the scene. And it also ensures that the experience for the scene is useful for future. Obviously, we cannot get much experience from an action that is executed just once. The Second feature, easy-to-remember, is due to the human biological limitation. We have difficulties in remembering complicated scenes due to the limited brain memory. And the complicated scene often consists of independent small parts. Usually the experiences of the different parts are also independent. Thus, human tend to divide complicated procedures into independent small things to accumulate experience individually. For example, playing football is a whole process. But the experience for defense and offense are accumulated individually. So the experience is applied in specific scenes.

The vessel' main objective is to reach its target safely. The basic behaviours of vessel consist of path planning and obstacle avoidance. For vessels which travel long distances, the path is usually fixed so experience has no much effect on this behaviour. Obstacle avoidance is a skill that is learnt from experience and often used during the navigation of a vessel. The experience for obstacle avoidance is also very meaningful for vessel's future performance when facing the similar obstacle.

From the above, we will apply the reinforcement learning algorithm to obstacle avoidance of vessels. The objective is to simulate captains' experience for obstacle avoidance.

4.3 The Extended BDI Agent Architecture

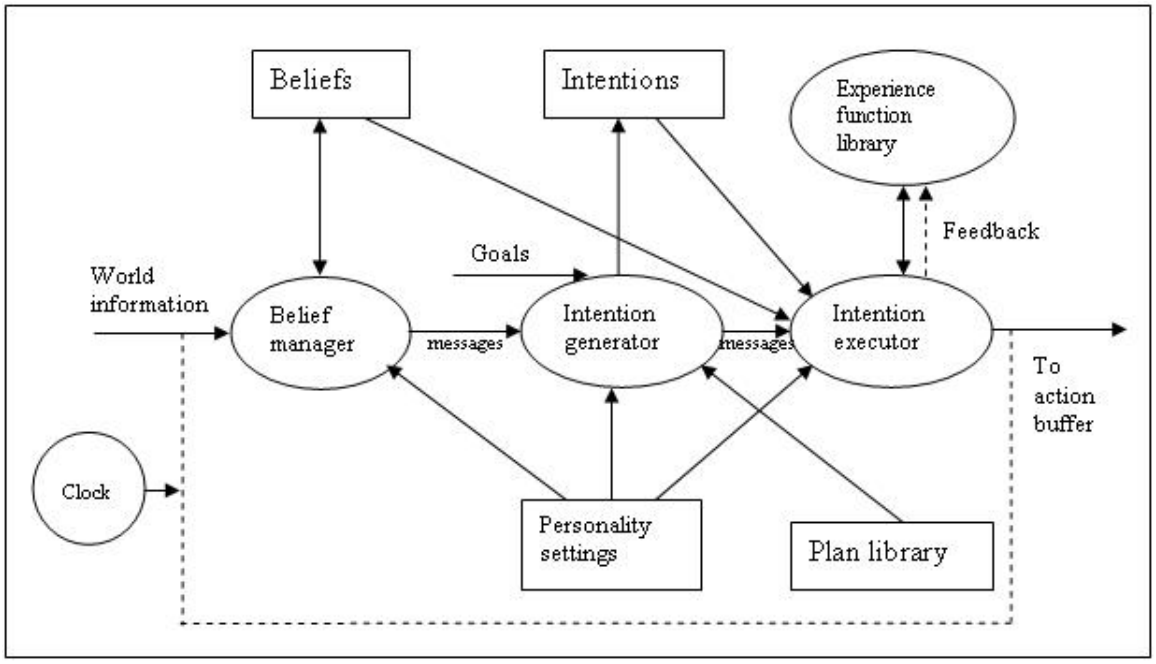


Figure 4-2 BDIE architecture.

The BDI agent architecture is extended as shown in Figure 4-2. It is based on the PRS system [63]. The similar extension can be implemented in the parallel BDI agent. The architecture consists of three main executing components, namely, the belief manager, the intention generator and the intention executor. The belief manager is responsible of receiving information and managing beliefs. Messages will be sent to notify the intention generator for new beliefs. The intention generator will produce intentions according to the incoming beliefs and goals and inform the intention executor about new intentions in messages. Then the intention executor explains and executes the intentions to produce output actions. The action buffer will keep the physical actions output by the intention executor. The plan library consists of plans for dealing with achieving goals. In this

experiment, the plans are represented using the Hierarchical Task-Network (HTN) [34]. HTN organizes actions as a network. A set of HTN grammar have been developed to formalize the HTN networks [33]. An example of HTN network for obstacle avoidance is given in Section 4.4.2. The two extension components, personality settings and experience function library, will be explained in the following.

4.3.1 Personality settings

The parameter settings representing the personality effect are kept in the 'personality settings' component. The belief manager, the intention generator and the intention executor will get the relevant parameters as needed. The parameters are classified into these main categories:

- Physical specifications such as the maximum speed, maximum acceleration, and so on. These will be used by the intention executor to translate actions into operational commands. They are also used by the belief manager to perceive the world to help to decide whether another object detected is a danger to itself. The intention generator will need this information to decide the final applicable actions.
- Personality preferences such as how likely the vessel agent is willing to give way to others. These will be used by the intention generator to decide which plan of actions is preferred. Personal preference also includes risk tolerance such as when an obstacle is a danger and how far a vessel should keep away from an obstacle.
- Priority control scheme and parameter settings for it.
- Parameter settings for the experience functions such as the decay factor for the reinforcement learning algorithms.
- And so on.

Different agents may get different parameter settings. With different settings, agents may demonstrate different behaviours. For example, the priority is used to decide the urgency

level of the incoming information to be notified to the intention generator. The belief manager will use the parameter settings to decide the urgency level of the beliefs. One possible way of classifying priorities of the messages is shown in Table 4-1. Captains with different personalities will have different opinions about new beliefs. For example, a meticulous vessel agent may set the priority of "finding new area" as 3. But for a careless agent, the priority will be set to 1.

Personality settings may be handled in 2 different ways in a multi-agent environment: 1. most agents are 'normal' beings and therefore work by default setting and a few agents will be triggered to have not-so-usual setting. 2. every agent needs to have individual setting. Then psychology experiments need to be conducted to find the distribution of the settings. A human-like method to control the priority changing is shown in Chapter 5.

Table 4-1 Priorities of messages for new beliefs

Priority	Description	Explanation
1	Beliefs at low priority	Something the agent needs to deliberate on when it is free.
2	Beliefs at medium priority	Something the agent needs to deliberate on not immediately but some fixed time in future.
3	Beliefs at high priority	Something the agent needs to deliberate on immediately but still can take time to think carefully.
4	Beliefs at very high priority	Something the agent needs to deliberate on immediately and try to make decisions as soon as possible and act.

4.3.2 Experience function library

The experience functions are successful and proven algorithms, which the agents can invoke to finish some composite actions. Previously, most agents concentrate on doing tasks based on the predefined plans, which consist of the steps of actions. The actions are usually primitive. Such as, in a plan of reaching a location, the actions may consist of renting a car and driving the car to target. The actions can be applied directly without further calculation. In Touringmachine [35], the primitive plans can do some calculations, but these are limited to computing the simple functions, such as the distance between two positions. Normally, a primitive action is not expected to take much execution time before it is completed and the control is returned to the agent. However, in a real human,

he may use any successful tools to achieve his targets. For example, after a captain decides to achieve a target, he may make a plan consisting of two steps: planning a path and navigation. Then he may use some global path planning algorithm to find the shortest path to the target based on the map. And some local obstacle avoidance algorithm is used for navigation. Such algorithms cannot be regarded as the primitive actions or final output actions. So we propose to incorporate an experience function library to the BDI agent to realize such algorithms.

The experience functions are saved in the experience functions library. In Figure 4-2, the experience function library is shown in ellipse as the three main components because the functions will be invoked to execute as part of the reasoning of the agent. The dashed lines mean feedback from action effectors. If there are learning algorithms involved, the feedback is used to train the learners. An experience function is an algorithm that has the following properties:

- Specialty. The algorithm can be used to solve a specific problem.
- Successfulness. The function has been proven to be successful for the problem.
- Independency. The function is a stand-alone function and does not depend on other functions' results.
- Complex.

The specialty and successfulness properties ensure that the function can be used to solve the specific problem successfully. The independency property means that the function can be used in the same way as a primitive action. Finally, the complex property decides that the function cannot be implemented as a primitive action, such as in the Touringmachine. The experience functions can be seen as the tools/skills that an agent has been using to solve specific problems and these tools/skills are based on its previous experience. The experience functions library provides several advantages to the agents designed:

- It is easy to understand. It is a folk psychological way to solve some problems. When we are using some tools, we seldom consider why we use it. So the utility is only related to the execution of plans.

- The plan library is easy to create and maintain. It reduces the needs to transfer complex calculation processes into plans. The agent can obtain the abilities that some algorithms can provide by simply incorporating the corresponding functions into the agent.

Here, to give the agent its learning ability, an experience function library is incorporated. Some skills of the vessel agent cannot be realized as primitive actions in the plan library, such as global path planning or obstacle avoidance. These skills will need more execution time to apply than the simple primitive actions. These skills can be improved through experience and time. So in the extended BDI architecture, such skills are realized as experience functions. For example, the reinforcement learning algorithm for obstacle avoidance can be realized in an experience function. These experience functions are grouped into the experience function library. The functions are invoked by the intention executor. If learning algorithms are involved, feedback may be obtained from the new beliefs after an action/plan is executed.

4.4 Experiment

As described above, the agent's characters are realized as different parameter settings, and experiences. A behaviour of obstacle avoidance is shown here. The agent can decide the next action based on the target direction and the experience accumulated previously. The actions of the agents with different parameter settings facing identical situations are investigated. Because experience function is used when demonstrating different parameter settings, the implementation of the learning algorithm is shown first.

4.4.1 Experience

In the experiment, the reinforcement learning algorithm is used to learn the skills for single dynamic obstacle avoidance, as illustrated in Figure 4-3. Reinforcement learning tries to find the state-action tuple with the best reward. The state-action table can be

obtained through training. The agent is recognized as A. The vessel B will present a moving obstacle to the agent A. To identify the status, five variables are used:

- d is distance between two vessels
- V_A is A's translational speed
- V_B is B's translational speed
- θ_a is the angle between A's moving direction and the line AB
- θ_b is the angle between B's moving direction and the line AB

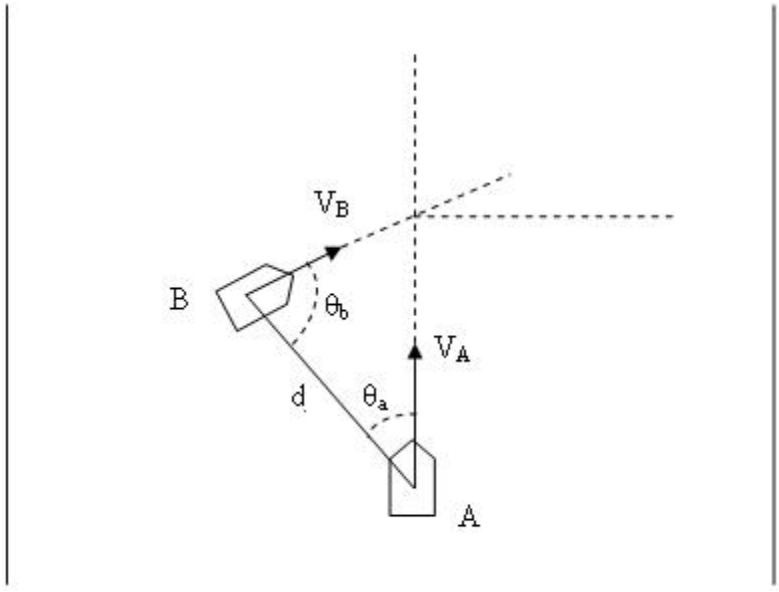


Figure 4-3 Obstacle avoidance.

Q-learning algorithm [133] is used here for learning to avoid obstacle. The Q-function, Q(s,a), represents the expected value of the reward for taking action a from current state s. Here s is represented as (d, V_A, V_B, θ_a, θ_b). And the agent's action part consists of the vessel's translational T_a and rotational speed R_a. Thus the state-action tuple for this question is (d, V_A, V_B, θ_a, θ_b, T_a, R_a). It is obvious that the variables are continuous. The number of records in Q-table should be the multiple of the size of the variables. Thus it is impossible to use the Q-table for this high-dimension input because there may be voluminous records which cannot be processed by a PC.

In the training, a discrete reward function is used to calculate the reward of new status, which is shown as:

$$R = \begin{cases} 2 - \dfrac{d}{max\ distance} - \dfrac{direction\ change}{max\ rotation\ change} & success \\ -1 \quad collision \\ 0 \quad otherwise \end{cases} \qquad 4\text{-}2$$

where *max distance* is the maximum distance between A and B, *direction change* is the angle between the original direction and current direction of A, and *max rotation change* is the maximum rotational acceleration of A.

So after each action, the Q value will be updated by:

$$Q'(s_t, a_t) = (1-\alpha) * Q(s_t, a_t) + \alpha * (R + \gamma * \max_{a'} Q(s_{t+1}, a')) \qquad 4\text{-}3$$

where α, γ are the parameters for training.

Then, the parameters of the RBF network are adjusted by the gradient of the difference of expected value and output value. The difference will be calculated as:

$$\Delta Q = Q'(s_t, a_t) - Q(s_t, a_t) = \alpha[R + \gamma * \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)] \qquad 4\text{-}4$$

In order to avoid that the values of the network parameter vector oscillate or even grow to infinity, we adopt the method to perform gradient descent on the mean squared Bellman residual. Because this defines an unchanging error function, convergence to a local minimum is guaranteed. This means that we can get the benefit of the generality of neural networks while still guaranteeing convergence. Then we can get the equations for the modifications of the parameters as:

$$\Delta w_i(t+1) = \eta_1 * [Q'(s_t, a_t) - Q(s_t, a_t, w_t)] * \left[\gamma \frac{\partial Q(s_{t+1}, a_{t+1}, w_t)}{\partial w_i(t)} - \frac{\partial Q(s_t, a_t, w_t)}{\partial w_i(t)} \right] \qquad 4\text{-}5$$

$$\Delta c_i(t+1) = \eta_2 * [Q'(s_t, a_t) - Q(s_t, a_t, w_t)] * \left[\gamma \frac{\partial Q(s_{t+1}, a_{t+1}, w_t)}{\partial c_i(t)} - \frac{\partial Q(s_t, a_t, w_t)}{\partial c_i(t)} \right] \qquad 4\text{-}6$$

$$\Delta \sigma(t+1) = \eta_3 * [Q'(s_t, a_t) - Q(s_t, a_t, w_t)] * \left[\gamma \frac{\partial Q(s_{t+1}, a_{t+1}, w_t)}{\partial \sigma(t)} - \frac{\partial Q(s_t, a_t, w_t)}{\partial \sigma(t)} \right] \qquad 4\text{-}7$$

We will update the parameters at the end of obstacle avoidance. Then the error from the training can be lessened by summing all derivatives together. If a training consists of t steps, the update equations will be:

$$w_i = w_i + \sum_{k=1}^{t} \Delta w_i(k) \qquad 4\text{-}8$$

$$c_i = c_i + \sum_{k=1}^{t} \Delta c_i(k) \qquad 4\text{-}9$$

$$\sigma = \sigma + \sum_{k=1}^{t} \Delta \sigma(k) \qquad 4\text{-}10$$

The vessel agent will be trained with random inputs. Each set of random input represents a unique sequence of experiences. After the algorithm converge, we can get the value function for calculating the expected reward after taking an action in a status.

The learning algorithm is combined in the experience function library. The interface of the obstacle avoidance function is shown in Table 4-2.

Table 4-2 Interface of obstacle avoidance function

Name	Obstacle avoidance
Input	$(d, V_A, V_B, \theta_a, \theta_b)$
Output	$*(T_a, R_a, Q(d, V_A, V_B, \theta_a, \theta_b, T_a, R_a))$

4.4.2 Parameter setting

In the vessel agent, the HTN for obstacle avoidance task is shown in Figure 4-5. The agent will use the navigation experience to decide the feasible actions to avoid the obstacles. This process will be simulated using a state-action value function. The function is trained using the reinforcement learning algorithm. In a state, the actions with higher state-action value will be better choices for current situation. Thus, we can evaluate the feasible actions using the state-action values. All feasible actions, whose state-action values are higher than a threshold, and their state-action values will be sent to the action decider together.

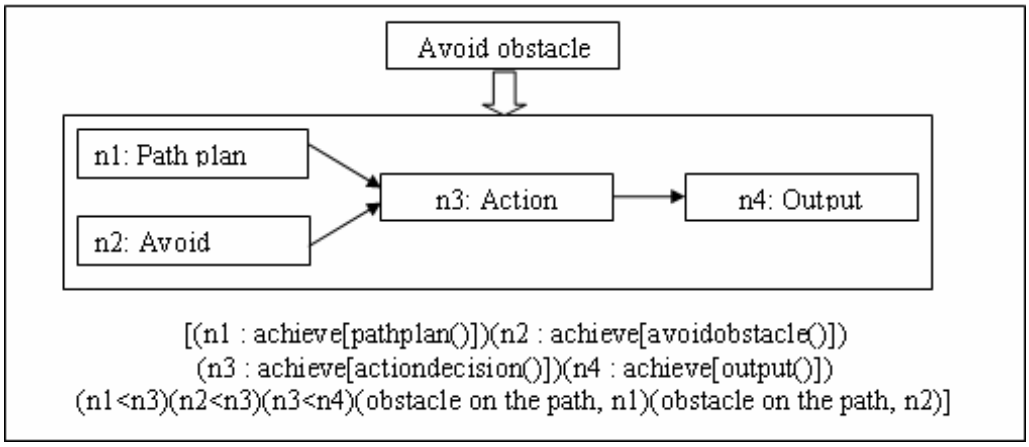


Figure 4-5 HTN for obstacle avoidance.

As identified in Figure 4-5, there are four actions in the plan. Path plan action is to select the next subgoal from the global path. A direction to this subgoal is outputted. The n2 step is implemented by a reinforcement learning algorithm. The output is the desired actions with the corresponding rewards. Then the action decider will decide the final action. The n2 and n3 will be implemented as the experience functions, which will be explained in the following section. The action decider will make decision using the following evaluation function. The inputs of the evaluation function are the direction to the immediate target θ_t, and the output from the obstacle avoidance function.

$$f(x) = \alpha * \frac{V_A}{\max \quad speed * 10} - \beta * \frac{|\theta_t - \theta_a|}{180} + \gamma * Q(\mathrm{d}, \mathrm{V_A}, \mathrm{V_B}, \theta_a, \theta_b, \mathrm{Ta}, \mathrm{Ra}) \qquad 4\text{-}11$$

where a is the action,

α, β, γ are positive discount factors, $\alpha+\beta+\gamma=1$,

θ_t is the direction to the immediate target,

$Q(d, V_A, V_B, \theta_a, \theta_b, T_a, R_a)$ is the RBF approximator.

From the function, we can see that the vessel prefers bigger speed, smaller deviation from the target direction and bigger reward of the action calculated by the learning function approximator. The action with the maximum evaluation value will be chosen as the output of the action decider. Then the action will be outputted to the action buffer for execution. The interface of the action decider function (n3) is shown in Table 4-3.

Table 4-3 Interface of action decider function

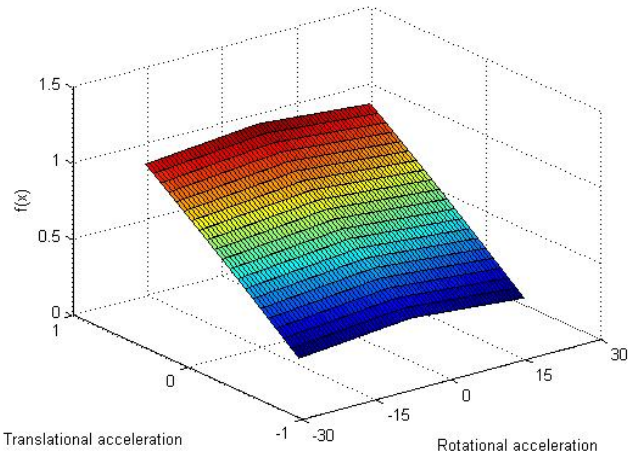
Name	Action decider
Input	*(Ta, Ra, Q(d, V_A, V_B, θ_a, θ_b, Ta, Ra)), θ_t
Output	(Ta, Ra)

An experiment is done with the following initial status in Table 4-4.

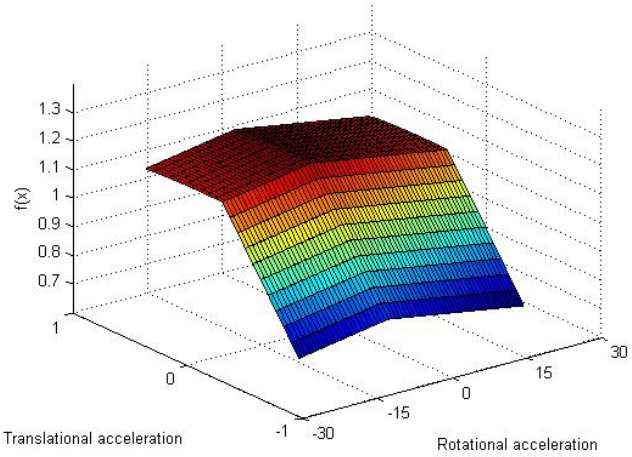
Table 4-4 Initial status of obstacle avoidance

Name	d	θ_a	θ_b	V_A	V_B	α	β	γ
Value	10	45	45	2	2	0.4	0.4	0.2

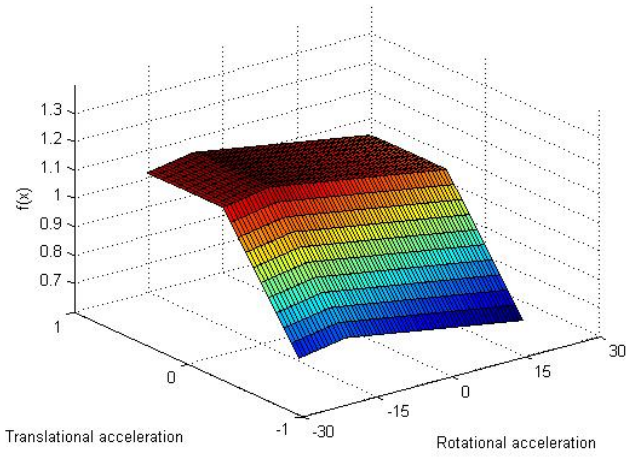
Then the motions of the vessel can be seen in the left-top of Figure 4-7. The red vessel agent is starting from the point (10, 0) to its target (0, 10). The blue vessel will run from the point (0, 0) and keep its original velocity and direction. After detecting the blue vessel, the plan to avoid obstacle is invoked. In each step, the output of the agent is gotten through executing the plan. As stated above, two experience functions are called when executing the plan. After three steps, the agent will be free from the obstacle. We can see that the agent takes translational acceleration actions. Its direction is a little deviated from the target direction because the reward function of the learning algorithm prefers a closer distance after the agent successfully avoids the obstacle. The evaluation function for different actions in the status can be seen in Figure 4-6.



Step 1: Chosen action: (1, 0)



Step 2: Chosen action: (0.3, -6)



Step 3: Chosen action: (0.4, -17)

Figure 4-6 Decision making.

In the following, we will try to design the vessel agents with different personalities by setting different α, β, γ parameters for the valuation function.

Table 4-5 Outputs of the evaluation function

Personality	α	β	γ	Output		
				Ta	Ra	*f(x)*
Adventurous	0.4	0.4	0.2	1	0	1.252284
				0.3	-6	1.26938474
				0.4	-17	1.22478426
Less adventurous	0.3	0.3	0.4	0.5	0	0.14215751
				0.5	-3	0.18508856
				0.3	-14	0.0916117
Less cautious	0.2	0.2	0.6	0.4	0	0.193021119
				0.4	-2	0.2540053
				0.3	-11	0.126752362
Cautious	0.1	0.1	0.8	0.3	0	0.244209826
				0.3	-2	0.323696345
				0.4	-7	0.167052984

The navigations are shown in Figure 4-7. The agent will navigate differently depending on how much the agent will decide based on the experience. As shown in Table 4-5, such difference can be used to show different personalities. From Figure 4-7, we can see that the agent will navigate closer to the target direction when the experience part has a bigger weighter. The cautious agent prefers a slower speed when facing an obstacle. But it ends up closer to the obstacle. In this sense, the meticulous behaviours may not always produce safer results.

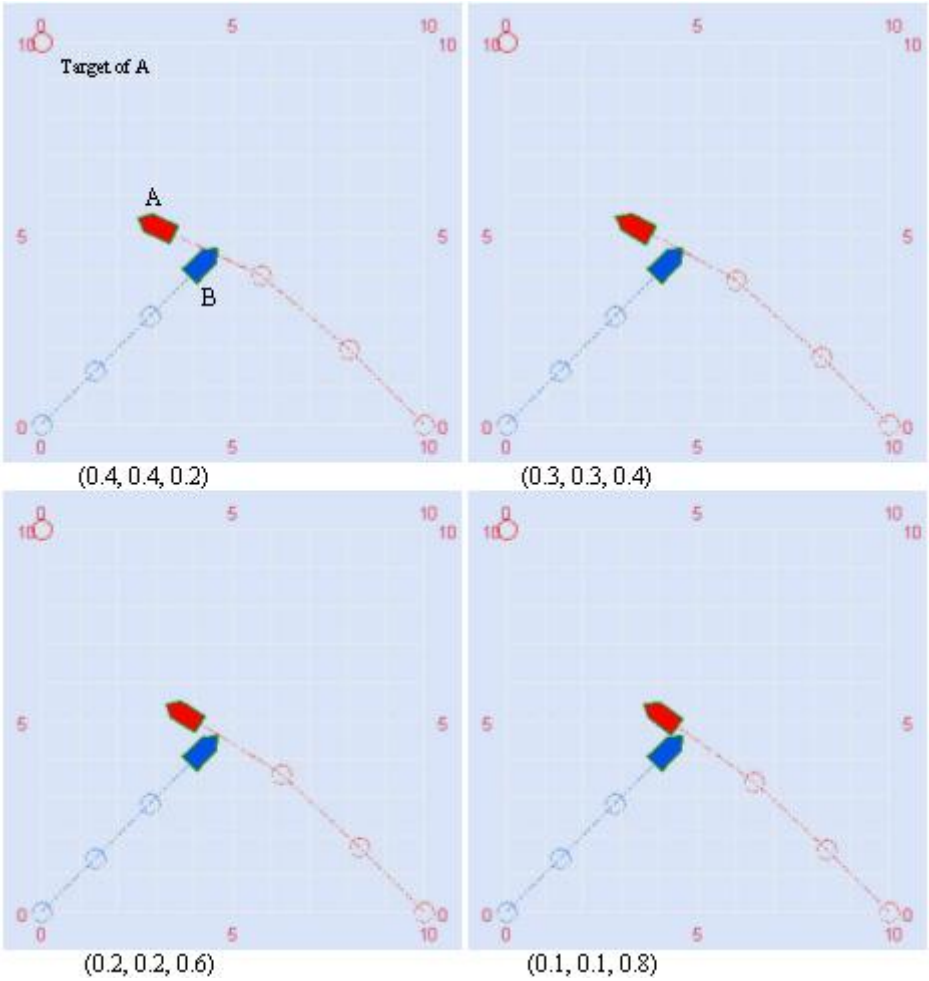


Figure 4-7 Path of avoidance.

4.5 Conclusion

In this chapter, we analyze the agent characters and introduce an extended BDI agent architecture to realize the characters. The character of an agent consists of personalities and experience. In the extended BDI agent architecture, the personalities of the agent are implemented as different parameter settings. And the experience is realized by a reinforcement learning algorithm. The learning algorithm is incorporated into the agent as an experience function. An example of vessel navigation is shown to demonstrate the behaviours of the agent with the characters.

The experience function library can be implemented well in a parallel BDI agent. The parallel agent's abilities of suspension and resumption at any time ensure that the agent can stay alert when calling an experience function. In my current work, each experience function is implemented to provide one specific skill or solve one specific problem. The experience functions are pre-learned and pre-defined. The agent does not create, select or improve an experience function. The agent just utilizes experience functions to make its decisions. Future work may put feedback mechanisms into the agent where the agent will be able to improve his experience functions through continuous learning.

CHAPTER

5

PRIORITY CONTROL

Activity scheduling mechanism plays a critical role in the correct behaviour of BDI agents. The parallel BDI agent framework allows the management of beliefs, generation of intentions and execution of a limited number of intentions to go in parallel. The desire/intention schedule can be done based on the priorities of the desires/intentions, which show the different degrees of importance and urgency. As we can see, the value of priority may change over time. In this chapter, we propose to enrich the framework with an extension which consists of 2 processing components, a Priority Changing Function (PCF) Selector and a Priority Controller. The priorities of the intentions can have different initial values and can be changed over time according to the chosen PCF. As an example, we design a function by simulating human behaviours when dealing with several things at the same time. The priority first increases with time according to a Gaussian function to simulate the fact that people are more inclined to do something which has been in their mind for sometime. After a certain time, if the intention still was not executed because of other higher priority intentions, its priority will decrease according to the Ebbinghaus forgetting curve. External reminders of an intention can also be handled by the Priority Controller. Experiment results show that with this mechanism, the parallel agent can show some human-like characteristics when scheduling intention to

execute. This can be used when simulating agents with human characters. Besides the extension, the agent operations that are facilitated by changing priority are also shown. By controlling priorities in the two ways, the desires/intentions in a parallel agent can be managed effectively.

This chapter is structured as follows. In the first section, we make an introduction to the background work. In Section 5.2, we present the parallel BDI agent framework with the proposed extension of priority control. In Section 5.3, we discuss the mechanisms of priority controls proposed that include some samples of human-like priority changing functions and how these functions are handled to reflect the effect of new beliefs, new desires, and new intentions on the priorities of existing desires and intentions. A simulation experiment is conducted to compare the behaviour of agents with and without the priority control. The experiment results are presented in Section 5.4. An analysis of how agent acts with different reminding functions is shown in Section 5.5. A conclusion is made at the end of this chapter.

5.1 Introduction

Bellman defines AI in [9] as the automation of activities that we associate with human thinking, activities such as decision making, problem solving, learning. One such activity is to decide when is the appropriate time to think about a certain matter or to do something. For an intelligent agent, this means it should know when to deliberate and when to act in addition to being able to deliberate on how to achieve a goal and how to carry out a plan. There has been significant amount of work on solving the "How" problem but not the "When" problem. As described in the survey of agent architectures [135], the world is symbolized and decision is made through logical reasoning of relationship among the symbols in the deliberative architecture. The BDI (belief-desire-intention) model is the most famous one of the deliberative architectures. It provides a folk psychological way by simulating human deliberation. The mental attitudes of belief, desire and intention represent the information, motivational, and deliberative states of the agent respectively [18, 111]. Several successful agent architectures and systems based on

BDI have been developed. PRS (procedural reasoning system) is an implementation of the BDI model. In each cycle, the belief is updated first. Then intentions are selected from the applicable plans. Finally action in the chosen intention is executed. The PRS system obtains the ability of reasoning in complex ways about dynamic processes while keeping appropriate responsiveness and control [63]. In UM-PRS [77], an extension of the PRS system, the hierarchy of the plans is kept for monitoring plan execution and replanning. AgentSpeak(L) [112] and LORA (logic of rational agents) [136] are two sets of operational semantics defined for BDI agents. The decision is made through logic reasoning. All these works are solutions to the "How" question.

The "When" question, that is, the scheduling of deliberation about new beliefs and the scheduling of intention execution is usually omitted in these BDI systems. The researchers mostly concentrate on solving the problem of intention generation. For example, in AgentSpeak(L) [112], the selection function S_I selects an intention to execute from the intention set I. The detailed selection criteria are not specified. We believe the scheduling of intention is crucial in an agent's ability to cope with the changing world. Some scheduling mechanisms appear in subsequent researches. In AgentSpeak(XL) [12], an extension version of AgentSpeak(L), a task scheduler is incorporated into the interpreter to decide how to select intentions. The set of intentions in the AgentSpeak(L) is converted into a corresponding TÆMS task structure. Then the selection is done based on the analyses of the relationship among the plans in the TÆMS task structure. The 'enables' and 'hinders' relationships indicate which plan may be executed first. Another method is shown in the JAM agent architecture [59]. The intention selection is done based on the utility value of the plan. The intention with higher utility will be executed first. Recently, another work of intention scheduling is reported in [79]. The researchers take several properties into consideration when scheduling the intentions, such as the importance of the plan, the estimated running time, the deadline utility function, the degree of completeness and FairFacter.

We consider the problem of deliberation scheduling and intention scheduling in an agent who will behave like an "average human". If people identify and accept an agent as

human and not machine-like, they tend to trust the agent better. For example, a companion to shut-ins or a playmate for a child should display a human way of interacting with its environment. When there are multiple goals to achieve and multiple intentions to execute, the agent needs a rational and human-like way to control the deliberation of plans for the goals and the execution of intention plans. We associate a single priority value with each desire or intention to facilitate the scheduling of deliberations and intention executions. The priority represents the importance and urgency of the goals or intentions to an agent. For humans, their priorities change with time. The priorities may be affected by how close it is to the deadline of a task, or a change in personal interest. The deadlines of tasks may also change, either forward or backward. So the priority of a goal or an intention of an agent should also change with time. In other words, the priority should be a function of time.

While specifying the priority of a goal or an intention of an agent by a function of time, it is also necessary to consider the influence of new beliefs, new desires and new intentions on the priorities of existing desires and intentions. New beliefs, new desires and new intentions may make some existing desires or intentions more important and urgent, or less important and urgent, or may even render them not relevant any more. We propose how to support these changes in the agent's behaviour.

Currently, the control of the priority changing with time has not been adequately researched even though some work has been done in the artificial life community. In [74], a priority control mechanism for behavioural animation is proposed. The priority is set at minimal value immediately after the agent displays a certain behaviour like drinking. Then this priority increases with time. The increased priority will induce the agent to drink again. However, expecting the priorities of all desires and intentions to change in the same manner is not realistic. Different desires and intentions should be allowed to change their priorities in various suitable ways.

We proposed a parallel BDI agent framework in Chapter 3 to achieve better reactivity and rationality in intelligent agents. This framework equips a BDI agent with the natural

abilities of doing several things at the same time and the ability of prioritizing the deliberations and intention executions according to the urgency of the matters. Each desire and intention is at a certain level of priority among the several levels of priorities. The level of priority is used in the scheduling of the desires/intentions in the agent. However this mechanism has the problem that with priorities set at constant levels, some desire or intention may be starved indefinitely by desires or intentions with higher levels of priorities.

In this chapter, we proposed a priority control extension to the parallel BDI agent framework in order to support the capabilities of representing the changing importance of different desires and intentions. Pre-defined *Priority Changing Functions(PCFs)* are associated with the desires and intentions. A Priority Controller will compute the priority value of the desires and intentions to help the scheduling decisions to be made at various time moments. We proposed a few priority changing functions which simulate the human behaviours when dealing with several things at the same time. A popular pattern is that it first increases the priority value according to a certain function and then decreases according to the Ebbinghaus forgetting curve. However other patterns are also possible. With the setting of suitable parameter values, the PCFs are also able to simulate the changing of priority when a person is not very motivated to pursue his goal or put an intention into actions. The function can also represent the changing of desire/intention priority when it will get stronger and stronger and stay at its maximum value until it is carried out. We have also incorporated other controls to realize the effect of new beliefs, new desires and new intentions on the priorities of existing desires and intentions or intentions that need to be executed exactly at a certain moment. This is to simulate human behaviours when dealing with several things at the same time. These controls of priorities for desires and intentions provide a human-like way to control an agent's activities. Other successful human-like systems are, for example, the i-Bid game player agent [70] and adaptive agent designation [134].

5.2 Priority Control Extension

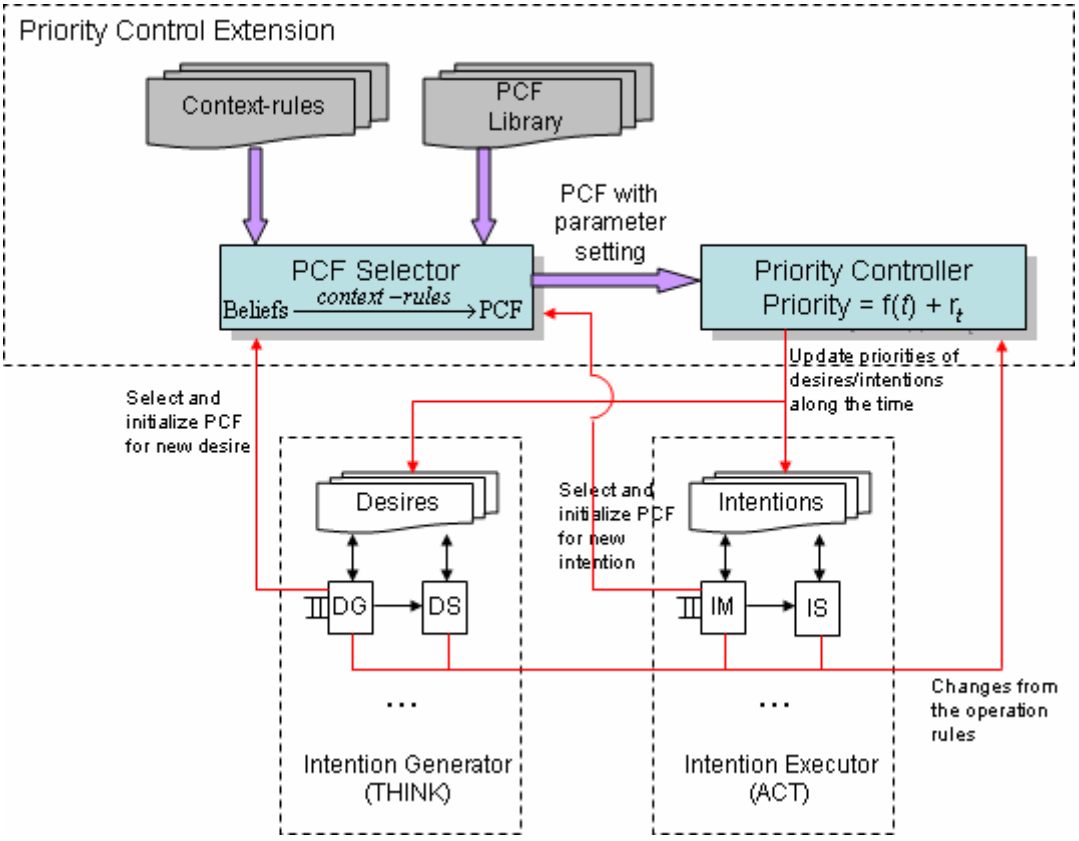


Figure 5-1 Priority control extension to the original parallel BDI framework (only parts of the original framework that interact with the extension are shown).

To represent the dynamic change of priorities of a desire or intention, each desire or intention will be associated with a Priority Changing Function (PCF) which defines how the priority should change with time. The priority control extension to the agent architecture is shown in Figure 5-1. Two processing components are introduced into the BDI agent, a PCF (Priority Changing Function) Selector and a Priority Controller. When a desire/intention is generated, the Desire Generator (DG) or the Intention Manager (IM) will call on the PCF Selector which will, based on some context-rules, (i) select a suitable PCF for the new desire/intention from the PCF Library and (ii) decide on the values of the parameters if any for the PCF. The signature of the function of the PCF Selector is as follows:

$$PCF\_Selector : desires \times PCFs \rightarrow PCFs \quad \text{and}$$

$$PCF\_Selector : intentions \times PCFs \rightarrow PCFs$$

where *desires* and *intentions* are the set of desires and the set of intentions of the agent respectively, and *PCFs* is the set of priority changing functions in the PCF library.

The Priority Controller will be responsible for updating the priorities of the desires/intentions according to their PCFs as time passes. In order to have the priorities of desires and intentions assessed accurately but not computed unnecessarily, the Priority Controller will update the priorities of desires/intentions each time the BDI agent is to select a desire/intention to execute by

$$\text{Priority} = \text{f}(t) \qquad 5\text{-}1$$

where f(t) is the PCF and t is the current time. This extension allows a BDI agent to select suitable PCFs for the desires/intentions and compute the priority values of desires/intentions at various points in time. Then the deliberation of the desires and the execution of the intentions can be scheduled by the Desire Scheduler (DS) or the Intention Scheduler (DS) based on their importance or urgency, represented by their priority values at the time.

Various PCFs suited to different intentions can be pre-defined. This enriches the BDI agent with the ability of realizing the scheduling of the desires/intentions in a more realistic way. As an example, for an intention to be completed before a deadline, t_d, a simple PCF is:

$$\text{f}(t) = \begin{cases} \alpha + \beta * (t - \text{t}_s) & \text{t}_s \le t \le \text{t}_d - t_e \\ 0 & \text{otherwise} \end{cases} \qquad 5\text{-}2$$

where t_s is the time when the execution of the intention plan can be started, t_e is the time required to execute the intention, α is the initial priority value, and β is the rate of changing of the priority value.

By setting a suitable value to t_s, it is very easy to manage clashing intentions or something/some task that needs to be done at a certain future time. For example, t_s may be set to the time when a clashing intention will finish so that it is feasible for this

intention to execute. At the time set for t_s, the intention will be activated from its sleeping status (priority=0). The PCF Selector will decide the values of α, t_s, t_d, t_e and β based on the features of the intention.

DG, DS, IM and IS are responsible for the generation, scheduling and managing of the desires and intentions. The operations for them are defined in Chapter 3. The priority changes are used to resolve obsolete, clashing and urgent desires/intentions. The requirement for such priority controls are summarized in Figure 5-2.

1. If a new belief makes an existing desire obsolete, set the priority of the desire to zero.
2. If a new belief makes an existing desire more/less urgent, change the priority of the desire.
3. If a new desire just generated makes an existing desire obsolete, set the priority of the existing desire to zero.
4. If a new desire just generated makes an existing desire more/less urgent, change the priority of the desire.
5. If a new desire just generated clashes with an existing desire, reduce the priority of the less important desire so that it will be put on hold.
6. If a new belief makes an existing intention obsolete, set the priority of the intention to zero.
7. If a new belief makes an existing intention more/less urgent, change the priority of the intention.
8. If a new desire just generated makes an existing intention obsolete, set the priority of the existing intention to zero.
9. If a new desire just generated makes an existing intention more/less urgent, change the priority of the intention.
10. If a new intention just generated clashes with an existing intention, reduce the priority of the less important intention so that it will be put on hold.
11. If a new intention just generated makes an existing intention obsolete, set the priority of the intention to zero.
12. If a new intention just generated makes an existing intention more/less urgent, change the priority of the intention.
13. If an existing intention just completed clashes with another existing intention which was put on hold, increase the priority of the waiting intention.
14. If an existing desire just achieved clashes with another existing desire which was put on hold, increase the priority of the waiting desire.

Figure 5-2 Requirement for priority changes caused by new beliefs, new desires and new intentions.

In the following parts, we will first show the designation and implementation of the *reminding-forgetting* PCF. Then the function is applied to the parallel BDI model for a comparison experiment.

5.3 Priority Control

With the architectural support as described in the previous section, we present in this section the mechanism of priority control so that what an agent deliberates and acts on are decided by his priorities, just like the humans. The basic control of priorities comes from the Priority Changing Function (PCF) which defines how the priority of a desire or an intention should change with time. The PCF is defined by

$$f(t) = Maximum\ priority * I(t) \qquad 5\text{-}3$$

where *Maximum priority* is the highest value which a desire or an intention may have, and $I(t)$ is the function of influence factor with a range [0, 1]. *Maximum priority* is an intrinsic constant value of a desire or an intention where $I(t)$ controls the changes in priority with time. This PCF with a suitable $I(t)$ will be applied in the parallel BDI agent to provide some human-like behaviour. $I(t)$ can be different for different desires and intentions.

Very often, human interests in a certain goal or intention go through a few phases. One phase is the interests are getting stronger and stronger and we call it the reminding phase. Another phase is the interests will be getting weaker and weaker and occasionally the goal or intention may even be forgotten and this is called the forgetting phase. There are also situations where someone has an unchanging interest to do something and this is called the unchanging phase. These phases may happen to a certain goal or intention one after another or there is just one phase throughout the existence of the goal or intention. For example, the interests in the intention to eat remains a constant at a low level for a short period after a person has just eaten but will then start to increase. An agent may have an interest to reorganize the furniture in the bedroom but he is either too lazy to do it or the interest is just not enough, so from the beginning the intention gradually fades away. An agent may have an interest to learn how to cook better and the interest will

increase till it reaches its maximum and never fades away. We will first describe how I(t) models the various phases of priority changing and then show how the various phases work together. Then we will present how the effect of a new belief/desire/intention on some existing desire/intention is modeled.

5.3.1 The reminding phase of a PCF

This is the phase where an agent has increasing inclination to deliberate on how to achieve a goal or to execute an intention plan. The priority of the goal or intention should be increased gradually until it reaches its maximum value. We call this phase the 'reminding' phase. The manner by which the priority of a desire or an intention increases may be different from that of the others. We propose three different functions to model the way a priority may increase, the Sigmoid, Gaussian, and ramp functions shown in Figure 5-5.

For the three functions, the value I(t) at $t = 0$ is y_0. The value of y_0 when multiplied by the maximum priority as shown in Equation 5-3 will return f(t), the initial priority of a desire or intention. At $t = t_m$, the value of I(t) should be 1 or very close to 1. t_m is the time when f(t) is to reach its maximum priority value. It should be the time till when the agent keeps interests to the deliberation/intention or the latest time a deliberation or an execution of a plan should start in order to meet a deadline. The ramp function is a model where the priority increases at a constant rate. It is realized by:

$$y = y_0 + \frac{1 - y_0}{t_m} * t \qquad 5\text{-}4$$

The Sigmoid function is commonly used to model the growth of some set p. Here it is used to model the growth of interests in a deliberation or an execution of a plan. It is shifted to right by $\ln\left(\frac{1}{\alpha} - 1\right)$ in x-axis. Then the function between [α, 1- α] in y-axis will be resized to [0, t_m] in x-axis and [y_0, 1] in y-axis. We get the equation:

$$y = \left[\frac{1}{1+\exp\left[\left(1-\frac{2t}{t_m}\right)*\ln\left(\frac{1}{\alpha}-1\right)\right]} - \alpha \right] * \frac{(1-y_0)}{(1-2\alpha)} + y_0 \qquad 5\text{-}5$$

where α will decide the figure of the function, 0<α<0.5.

Five sets of Sigmoid functions with different α, t_m and y_0 are shown in Figure 5-3. It is noticed that with a smaller α, it takes a longer time before the priority starts to increase sharply. The bigger the α is, the closer the function is to the ramp function.

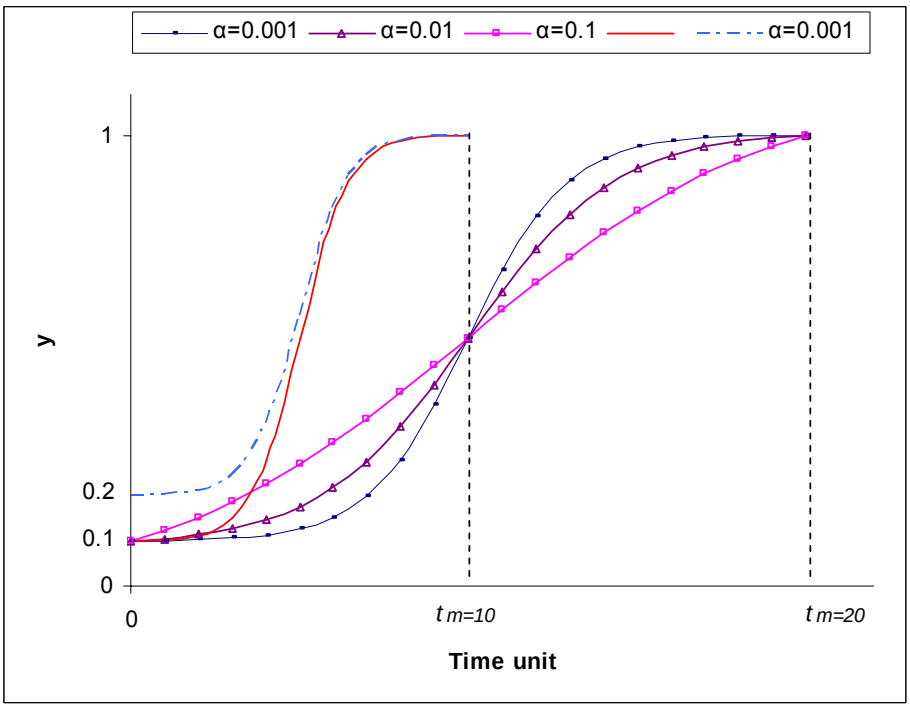


Figure 5-3 Sigmoid functions.

The Gaussian function is the function:

$$G(x) = \eta * \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-x_0)^2}{2\sigma^2}\right) \qquad 5\text{-}6$$

where η is a constant value, σ is the width of Gaussian function, and x_0 is the middle point of the function.

In order to make I(t) reach the maximum value 1 at the middle point, we set η to $\sqrt{2\pi}\sigma$. To make I(0)= α, we calculate the width of Gaussian function σ by:

$$\sigma = \frac{\sqrt{2}}{2} * t_m * \sqrt{\frac{-1}{\ln \alpha}} \qquad 5\text{-}7$$

These produce the I(t) function as below:

$$I(t) = \left[\exp\left(\left(\frac{t}{t_m} - 1 \right)^2 * \ln \alpha \right) - \alpha \right] * \frac{1 - y_0}{1 - \alpha} + y_0 \qquad 5\text{-}8$$

Five sets of Gaussian functions with different t_m and y_0 are shown in Figure 5-4. With a smaller α, the increasing is slower at the initial period. Normally, we set $\alpha = y_0$. So the equation 5-8 will be simplified to:

$$I(t) = \exp\left(\left(\frac{t}{t_m} - 1 \right)^2 * \ln y_0 \right) \qquad 5\text{-}9$$

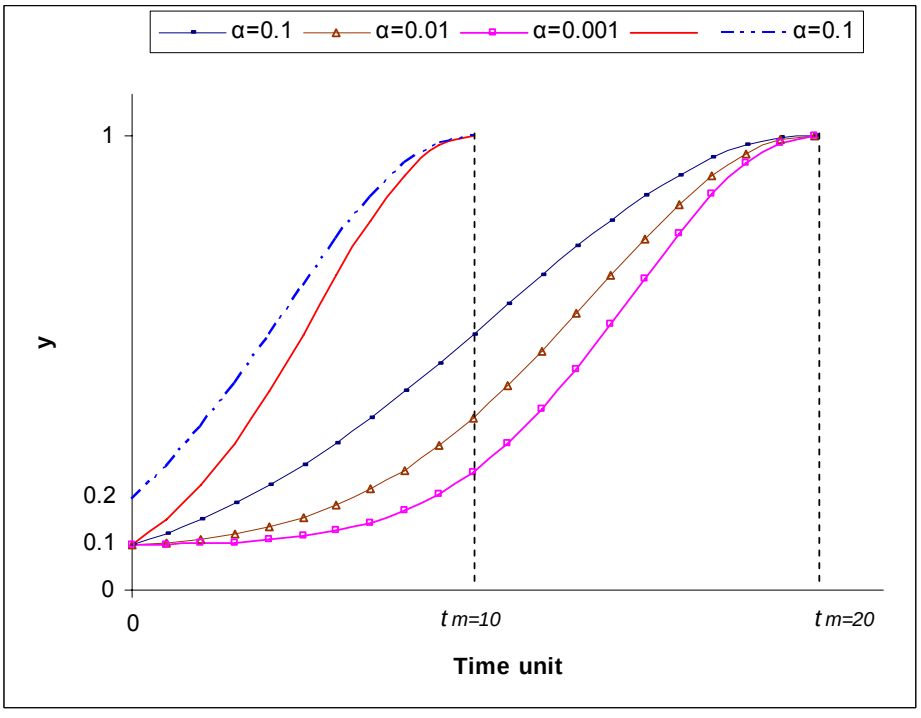


Figure 5-4 Gaussian functions.

Both the Sigmoid and the Gaussian function have the property that the gradient of I(*t*) gradually decreases to zero when t is approaching t_m. This simulates the increase in the agent's interest in a goal or an intention gradually stops as the interest reaches its maximum. For example, for the I(t) function that is based on the Gaussian function, (1) the function value is increasing for $t \leq t_m$; (2) the rate of increase first increases for $0 \leq t \leq t_m - \sigma$ ($I''(t) \geq 0$) and then decreases for $t_m - \sigma \leq t \leq t_m$ ($I''(t) < 0$); (3) the increase in function value and the rate of increase at t_m are 0, which means that the trend to increase the priority has stopped. The difference between these two functions is that Sigmoid function initially increases more slowly than the Gaussian function. Sigmoid function models people who tend to leave things to "last minute" where Gaussian function models people who tend not to do so. The three kinds of reminding functions are shown in Figure 5-5. And an analysis of the agent behaviours with the different reminding functions is shown in the last section of this chapter. In the other following parts, Gaussian function is chosen as an example to demonstrate the agent's reminding function.

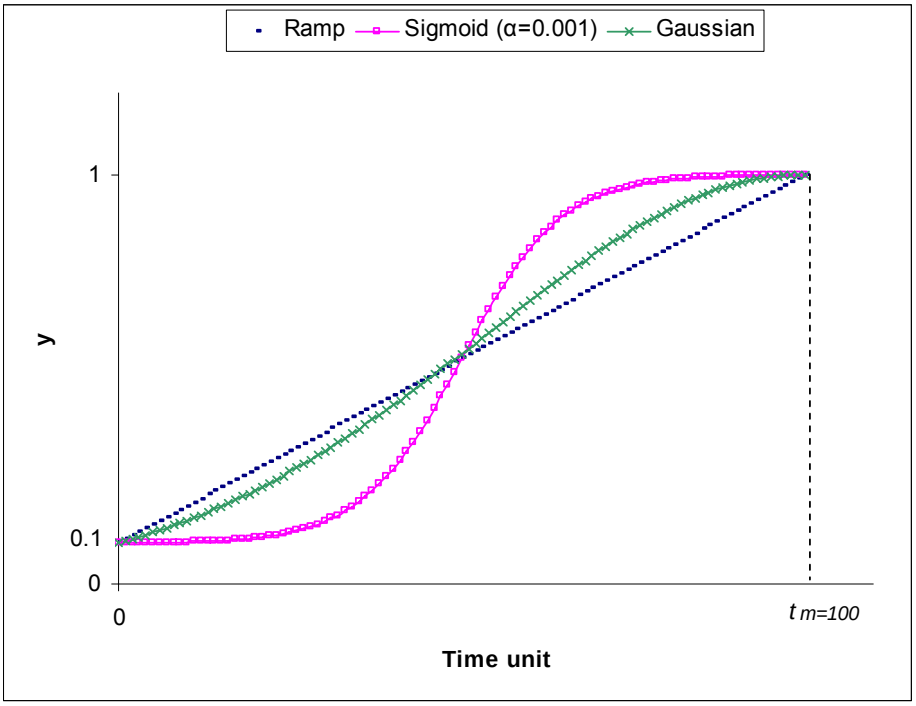


Figure 5-5 Comparison of three functions.

5.3.2 The forgetting phase and the unchanging phase of a PCF

The forgetting phase is the phase where an agent's interest in a goal or an intention plan is fading. The priority of the goal or the intention should be decreased gradually. This may happen if an intention is deferred for a long time because intentions with higher priority keep on coming and the agent do not manage to carry out the intention plan which has a lower priority. This is similar to that humans forget to do something when they are doing something else more important. In biological science, this is a protective mechanism to ensure that human can learn new things. We tend to forget things that the external environment does not remind us of. So as time passes, the priority of the intention in an agent is decreased. If the priority decreases to a value below a threshold, the intention will be removed (forgotten). For human-like agent, proper parameter settings should be gotten through studying human behaviour models. The first significant study on memory was performed by Hermann Ebbinghaus and published in 1885 as *On Memory*. Ebbinghaus was the first to describe the shape of the forgetting curve [1]. This curve is the biological base on which we simulate the process of intention retention. In [3], the forgetting curve is described as:

$$\mathrm{R} = e^{-t/S} \qquad 5\text{-}10$$

where R is the retention, which means the ability to retain things in memory, t is the elapsed time, and S is the strength of memory, which means the duration of things in memory.

The forgetting curves with different S are shown in Figure 5-6. It can be seen that with a larger S, the retention decays more slowly. When S is a very big value, the retention will keep constant as 1, that is, no forgetting.

The unchanging phase is the phase where an agent's interest in a goal or an intention plan is at a constant level. The priority of the goal or the intention is specified by a constant.

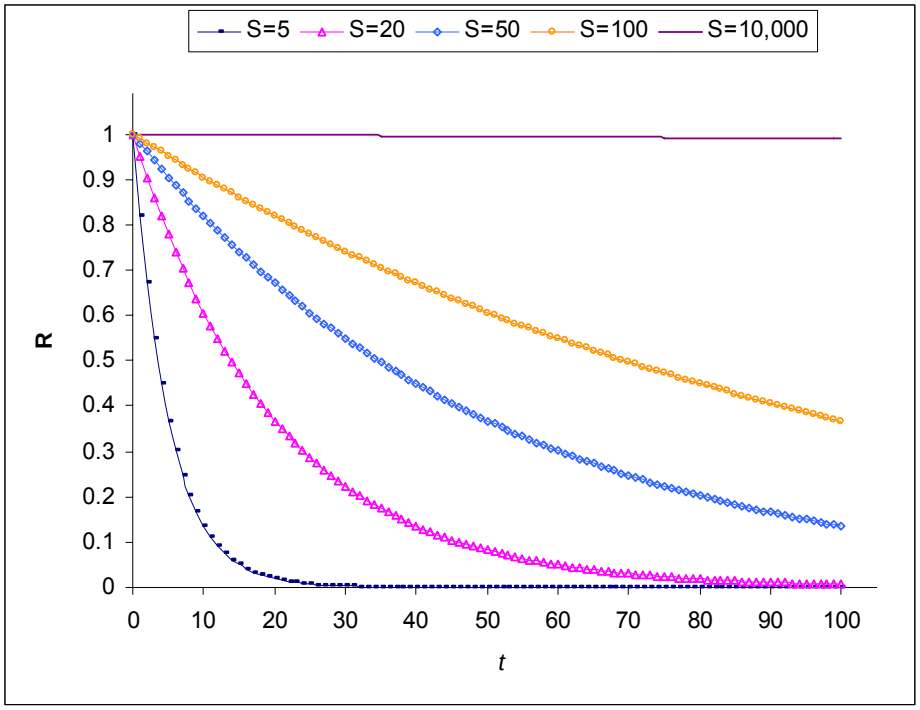


Figure 5-6 Forgetting curves with different *S*.

5.3.3 The complete PCF

The complete PCF, or more specifically, the complete I(t) is formed by either any one of the single phased functions described earlier, or it is a concatenation of two or more single phased functions. For example, we can compose an I(*t*) by combining the functions of the reminding phase and the forgetting phase together. The result is a *reminding-forgetting* function:

$$\mathrm{I}(t) = \begin{cases} \exp\left(-\dfrac{(t-\mathrm{t_m})^2}{2\sigma^2}\right) & \text{if } t <= \mathrm{t_m} \\ \exp\left(-\dfrac{t-\mathrm{t_m}}{\mathrm{S}}\right) & \text{otherwise} \end{cases} \qquad 5\text{-}11$$

An example of the situation modeled by this I(t) is that an agent intends to search for a piece of information which is 'hot' recently but he has more urgent things to do so does not find time to search. Then after a while, the information he wanted is no longer 'hot' so the interest and therefore the priority goes into the forgetting phase.

It is easy to show that I(*t*) is continuous at every point. The influence factor does not change significantly at any time. By proving the continuity of the influence function, we intend to show the fact that the simulated human behaviour is consistent without outside disturbance.

We will use the *reminding-forgetting* function in (9) as an example to show how the I(t) function is composed. The following initial parameters need to be decided:

Table 5-1 Parameters related to the reminding-forgetting function

Name	Type	Explanation
IP (Initial Priority)	Float	It is the initial urgency of the desire/intention.
MP (Max Priority)	Float	The maximum priority the desire/intention can have.
t_m	Integer	The time when the *forgetting* process begins.
S	Integer	Strength of memory. It is assumed that a higher initial priority will have a longer retention.
Threshold	Float	In *forgetting* progress, if the priority is below the threshold, the intention will be removed.

Table 5-2 Intentions with different PCF parameter settings

Intention	IP	MP	t_m	S	Threshold
1	1	1.5*IP	20	10*IP	10%
2	2	1.5*IP	20	10*IP	10%
3	1	2*IP	20	10*IP	10%
4	1	1.5*IP	10	10*IP	10%
5	1	1.5*IP	20	20*IP	10%
6	1	1.5*IP	20	10*IP	20%

Figure 5-7 shows the change of priority for 6 sample intentions with the parameter settings shown in Table 5-2. The intentions 2-6 each has one different PCF parameter as compared with the intention 1.

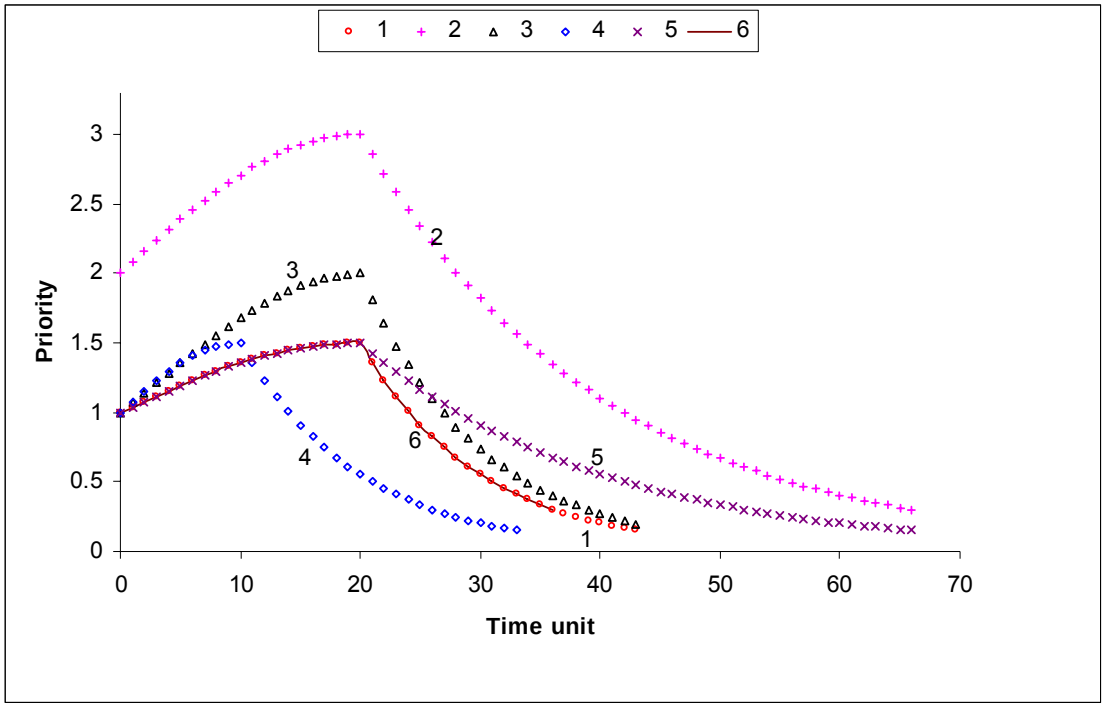


Figure 5-7 Priority Control of Four Intentions.

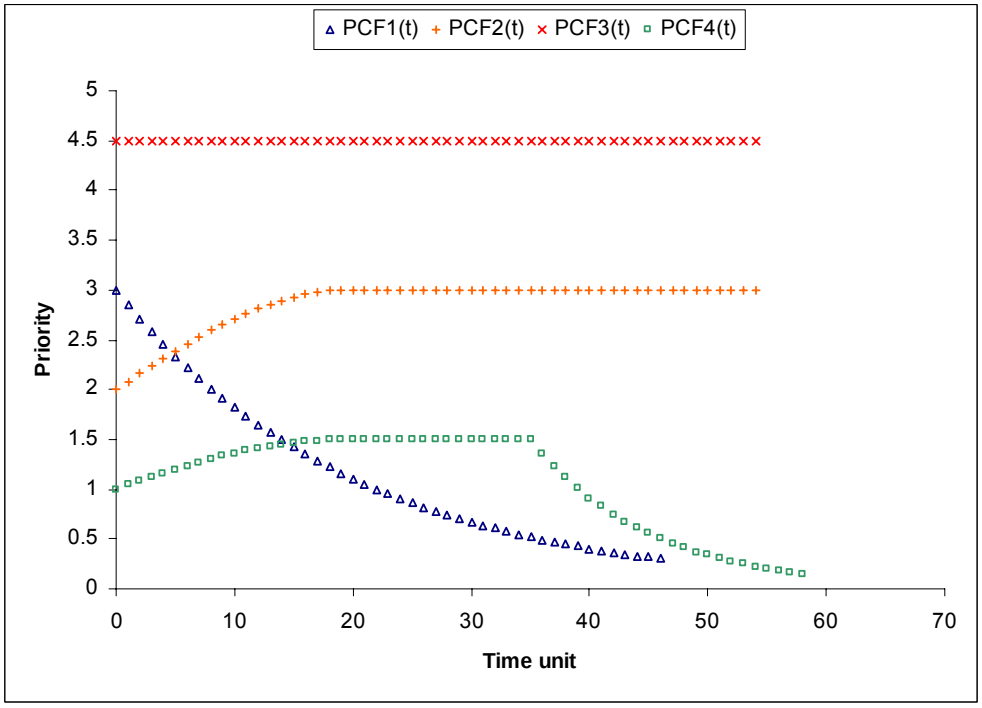


Figure 5-8 Examples of several PCF(t).

Figure 5-8 shows several examples of I(t). PCF1(t) is a single phased function where the priority depreciates from the beginning. This happens when someone has the intention to do something but he is either too lazy to do it or the interest is just not enough, he(the agent) starts to forget about it according to the forgetting curve from the beginning. PCF2(t) is the concatenation of reminding phase and the unchanging phase. Notice that this can also be achieved by concatenating the reminding phase and the unchanging phase and setting S to infinity in the forgetting phase. So before the desire/intention is completed, it will never be forgotten. PCF3(t) is a single unchanging phase function. PCF4(t) is the concatenation of the reminding phase, the unchanging phase and the forgetting phase. It can be used in the case that the priority is kept at the maximum value for a period of time before the *forgetting* period starts.

5.3.4 Priority change caused by other desires/intentions

It is noticed that the I(t) function as described earlier changes the priority of a desire or an intention in the absence of the effect of new beliefs, desires and intentions. However, as listed in Figure 5-2, there are situations where a new belief, a new desire or a new intention may make an existing desire or intention more urgent or less urgent, therefore the priority of the affected desire or intention needs to be increased or decreased. For example, suppose the human master asked his robotic agent to wash his car while the agent is doing cleaning in the house and the robotic agent also has a few other things to do. The agent has the intention to wash the car but the priority is not as high as his other intentions. After a little while the master reminds the agent about washing his car. At this point the priority for washing the car should be increased. So the new belief that the car needs to be washed sooner should have the effect of increasing the priority of the intention of washing the car. Another scenario that will change the priority of an existing intention: the robotic agent has the intention to tidy up a room but his master tells him to iron a shirt in the next 10 minutes. The robotic agent generates the intention to iron the shirt and has to lower the priority of tidying up the room. In situations like these examples, the priority of an existing desire/intention at t and beyond is affected. t is the

time when a new belief, a new desire or a new intention is generated and it is the moment when the priority of an existing desire/intention should be changed.

To model the effect on the I(t) of the existing desire/intention, we have

$$\begin{aligned} \mathrm{I}_{\mathrm{new}}(t) &= \min(\mathrm{I}(t)+r, 1) \\ \mathrm{I}_{\mathrm{new}}(t) &= \max(0.1, \mathrm{I}(t)-r) \end{aligned} \qquad \text{5-12}$$

where r is the increase/decrease of the influence factor. r has a value in the range (0, 1] and will be decided according to the relative urgency of the affecting and the affected desire/intention. The maximum value of I(t) is kept at 1 such that the priority of the existing desire/intention will not be increased beyond its maximum priority. This is to make sure that the increase in priority will not render it to have a higher priority than the desire/intention should have and to overpower other desires/intentions that are more important and critical by nature, for example, life-saving intentions.

In the following discussion, we look at the computation of $\mathrm{I}_{\mathrm{new}}(t)$ where the priority will be increased, that is, r is added to I(t). Suppose I(t) is the *reminding-forgetting* function in Equation 5-11. The increase in I(t) may come in *reminding* phase or the *forgetting* phase.

In the first case, the increase of I(t) by r can be realized by shifting the reminding function left by Δ_{t1} on the time scale as given below.

$$\Delta_{t1} = \begin{cases} 0 & \text{if } \mathrm{I}(t)+r \geq 1 \\ \sqrt{2}\sigma * \left(\sqrt{-\ln \mathrm{I}(t)} - \sqrt{-\ln \mathrm{I}(t)+r}\right) + \Delta_{t1} & \text{otherwise} \end{cases} \qquad \text{5-13}$$

where Δ_{t1} is zero before the first time I(t) is increased by a new belief/desire/intention and at subsequent times, Δ_{t1} is a further shift from the previous shift.

If the increase in I(t) comes in the *forgetting* phase where the priority is decreasing, the forgetting curve will be shifted right by Δ_{t2} on the time scale as given below.

$$\Delta_{t2} = \begin{cases} 0 & \text{if } \mathrm{I}(t)+r \geq 1 \\ \mathrm{S} * \left(\ln \mathrm{I}(t) - \ln \mathrm{I}(t)+r\right) + \Delta_{t2} & \text{otherwise} \end{cases} \qquad \text{5-14}$$

Δ_{t2} is zero before the first time I(t) is increased by a new belief/desire/intention in the forgetting phase and at subsequent times, Δ_{t2} is a further shift from the previous shift. Similarly, the decrease in I(t) when I(t) is in the reminding phase is achieved by a shift to right on the time scale and the decrease in the forgetting phase is by a shift to left by an appropriate amount.

With the time shifting values $\Delta_{t1}$and Δ_{t2}, the function I(t) is:

$$\mathrm{I}(t) = \begin{cases} \exp\left(-\dfrac{(t+\Delta_{t1}-t_m)^2}{2\sigma^2}\right) & \text{if } t \le t_m - \Delta_{t1} \\ \exp\left(-\dfrac{t+\Delta_{t1}+\Delta_{t2}-t_m}{S}\right) & \text{otherwise} \end{cases} \qquad 5\text{-}15$$

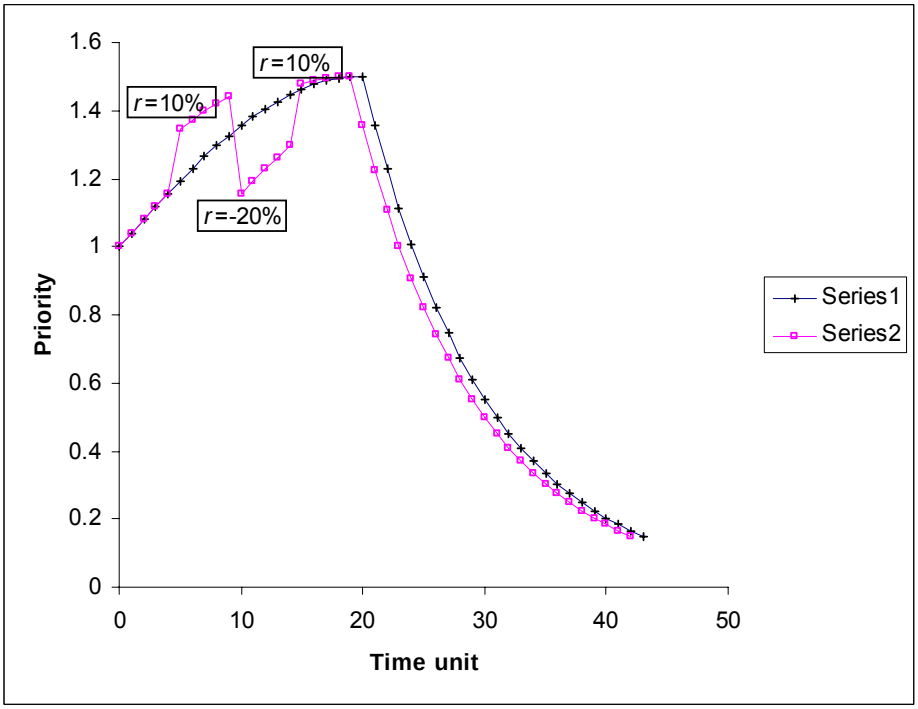


Figure 5-9 I(*t*) shifting in the Reminding Phase.

An example is shown in Figure 5-9. The intention is created with an initial priority of 1. The line 2 shows the priority changing with three shifts in the *reminding* phase. The

change in I(t) is an increase of 0.1 at time 5, a decrease of 0.2 at 10 and another increase of 0.1 at 15. Another example is shown in Figure 5-10. The intention is created with initial priority 1. The line 2 shows the priority changing with four shifts in the *forgetting* phase. Two are at time 25 and 30, the increase in I(t) is 0.5. This will make the priority value rise to the maximum value. The 3rd is at time 35, the priority will be decreased by 10%. The increase at time 40 is by 20%. The cases in Figure 5-10 are easy to understand in an imaginary scene that an absent-minded person acts under others' reminders. This is similar to active recall in mnemonic techniques [1]. The difference here is that here the outside reminders do not change the strength of memory.

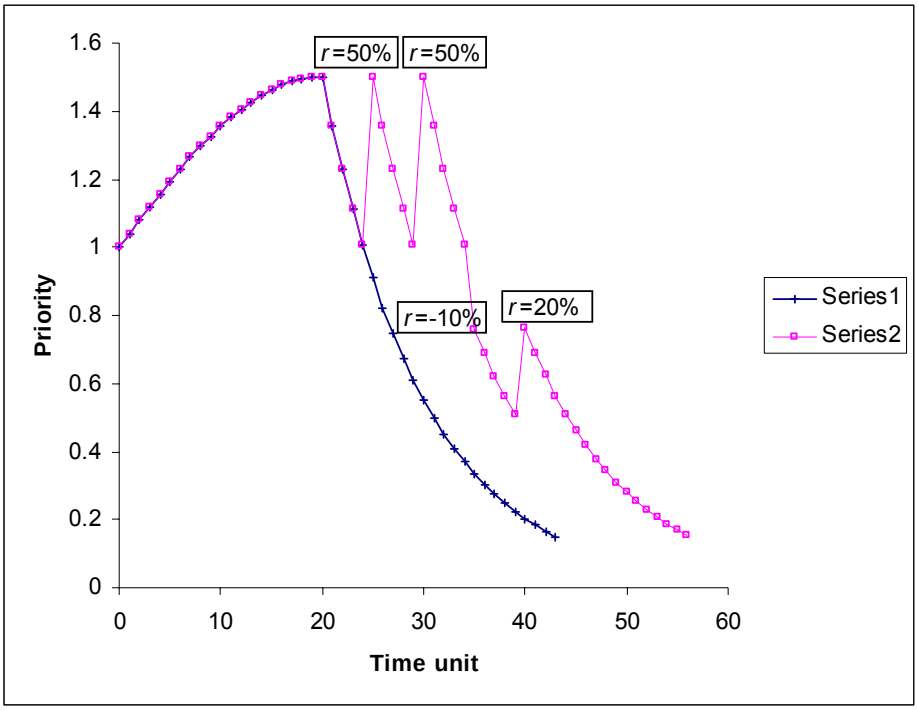


Figure 5-10 Outside Reminders in Forgetting Process.

5.4 Comparison of Parallel Agents Without and With the Reminding-forgetting PCF

In this experiment, parallel BDI agents without or with priority control are simulated. A set of events are input to them. Then the performance of them is analyzed. The events designation is the same as demonstrated in Section 3.3.2.

We consider three sequences of events with different average inter-arrival times. The average inter-arrival times of the 3 sequences of events are respectively smaller than, equal to and larger than, the average processing time required by an event. The events statistics used in the experiments are shown in Table 5-3.

Table 5-3 Events statistics.

Set	Expected average interval 1/λ	Events count					Actual average interval
		Priority				sum	
		1	2	3	4		
a	2	25	22	26	25	98	2.42
b	4	25	26	24	21	96	4.04
c	6	26	22	24	23	95	5.64

We experiment with 6 agents: agent 1 has a constant PCF so the initial priority is the priority all the time; agent 2 to 6 each has a different parameter settings for the PCF as shown in Table 5-4. The initial priority value of an intention is used as the basic priority.

Table 5-4 Agents types.

Agent no	PCF	PCF parameter settings			
		S	T_m	MP	Threshold
1	-	-	-	-	-
2	√	100*IP	20	1.5*IP	10%*MP
3	√	100*IP	20	1.5*IP	1%*MP
4	√	100*IP	20	2.5*IP	10%*MP
5	√	100*IP	40	1.5*IP	10%*MP
6	√	300*IP	20	1.5*IP	10%*MP

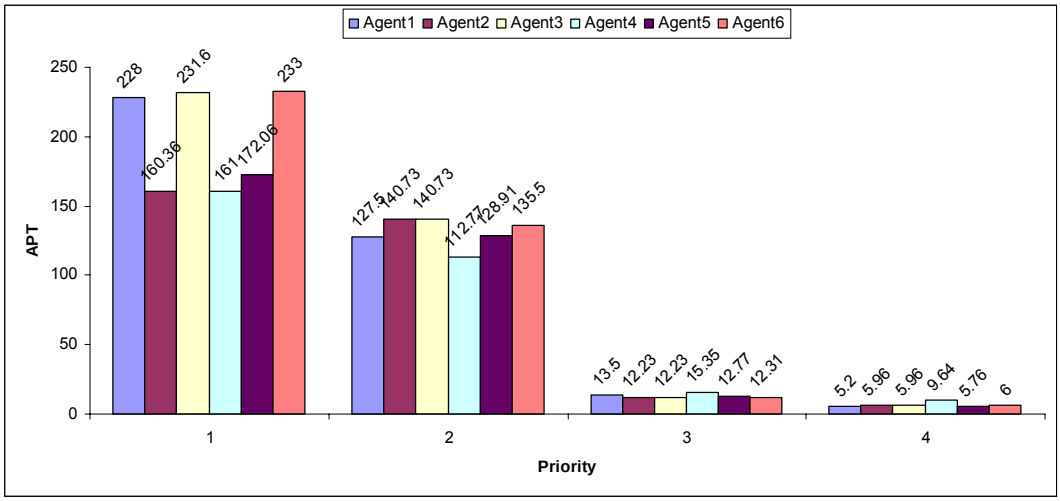
The intention processing time is defined as the duration from the time when the intention is created to the time when the execution of the intention is finished. The average processing time (APT) of the three sets of events by the agents is shown in Figure 5-11. In set *a*, compared to the expected average processing time 4, agent 1 does not have sufficient time to finish processing an event before the next event arrives. So the intentions with lower priorities have to wait for a long time. The time to process the events with priority 1 is 228 time units. So the intention is starved for a very long time. With the increased average inter-arrival time, the agent 1 has more time to process an event before next event begins and the corresponding APT is decreased to 38.6 for set *b*

and 10.8 for set *c*. In agents 2, 4 and 5, the APTs are the APTs of the events that get processed and they are smaller than those for agent 1. Here some intentions with priority 1 in set *a* are forgotten due to a long waiting time, which can be seen from the statistics in Table 4. For events with priority 4, the APT is not affected too much, because the urgent events will be scheduled first. What we see is that some of those low priority events that experience terribly long waiting time in agent 1 are forgotten in agent 2, 4 and 5 for event set *a*.

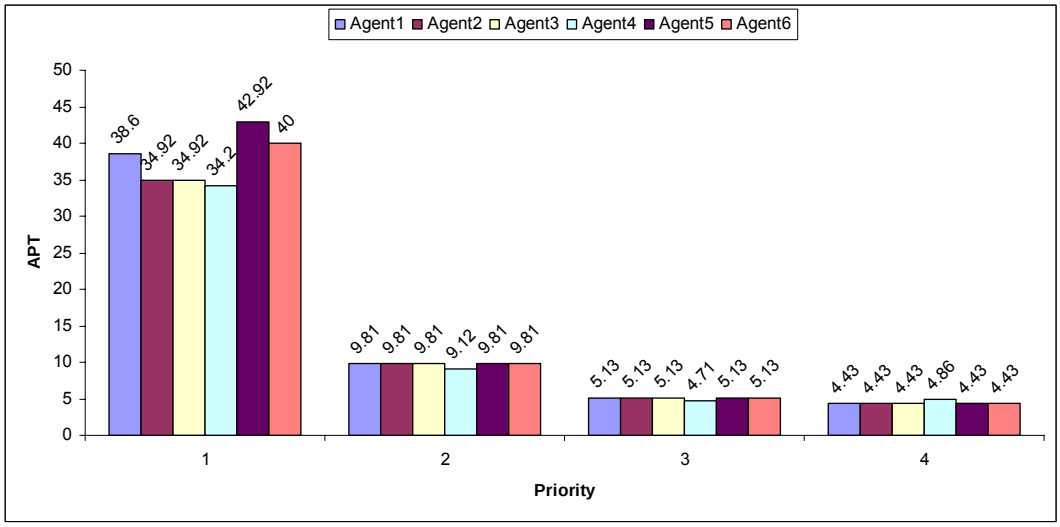
Table 5-5 Events processed statistics.

Agent no	Set	completed / forgotten				
		Priority				sum
		1	2	3	4	
2	a	14/11	22/0	26/0	25/0	87/11
	b	25/0	26/0	24/0	21/0	96/0
	c	26/0	22/0	24/0	23/0	95/0
3	a	25/0	22/0	26/0	25/0	98/0
4	a	14/11	22/0	26/0	25/0	87/11
5	a	16/9	22/0	26/0	25/0	89/9
6	a	25/0	22/0	26/0	25/0	98/0

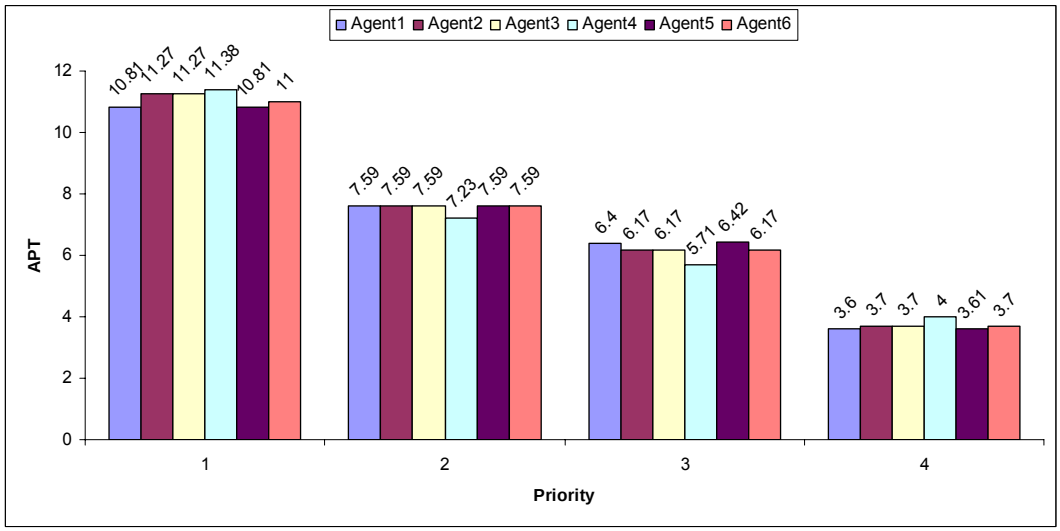
Looking at Table 5-5, we see the effect of the parameters of the PCF. For set *a*, 11 events are forgotten by agent 2 (row 2a). This is because the agent has no enough time to process all the crowded events. Rows 2b and 2c have all the events processed because of the longer event inter-arrival time. Comparing row 2a and 3a, more intentions are processed because of a lower threshold of retention. Comparing row 2a and 4a, the numbers of intentions processed are same because a higher maximum priority will not change the retention time of the intention. Comparing row 2a and 5a, 2 more intentions are processed because of a longer *reminding* period before forgetting starts. Comparing row 2a and 6a, 11 more intentions are processed because of a bigger strength of memory. The APT of them are bigger for intentions with lower priority in set *a*.



a. average interval = 2.42



b. average interval = 4.04



c. average interval = 5.46

Figure 5-11 APT of events.

5.5 Agent Behaviours with Different Reminding Functions

In this section, an analysis of how agent will behave is given with the three different reminding functions proposed in Section 5.3. The three functions are Ramp, Sigmoid (α=0.001) and Gaussian functions. We will estimate the probability that in the agent an intention with a reminding function is running or starts its execution first time at a time point before the t_m.

The following are assumed:

- The highest priority allowed in the agent is 100. Thus MP of the intention is in the range [1, 100]. IP of the intention is set as 10%*MP. At time unit t, the priority of the intention is calculated as *PCF(t)*.
- At any time *t*, there is random number of other intentions existing in the agent. The highest priority of the intentions is p_{high_t}. Then we define:

$$D(\alpha) = \text{Pro}(p_{high\_t} < \alpha) \qquad 5\text{-}16$$

Then the probability that the intention is running at a time point *t* is calculated as:

Pro(the intention is running at time unit *t*)=
Pro(p_{high_t} is bellow the current priority of the intention)=
Pro($p_{high\_t} < PCF(t)$)=
$D(PCF(t))$ 5-17

The probability that the intention is started first time at *t* is calculated as:

Pro(the intention is started first time at time unit *t*)=
Pro(the intention is not started previously)*Pro($p_{high\_t} < PCF(t)$)=

$$\left[\prod_{i=1}^{t-1}\left(1 - D(PCF(i))\right)\right] * D(PCF(t)) \qquad 5\text{-}18$$

In an agent, the distribution of p_{high_t} can be in any forms in various situations. The following analysis is made in the case that it is distributed in [1, 100] according to the standard normal distribution. Thus we have:

$$\begin{aligned} D(PCF(t)) &= D_{standard-normal}\left(scale(PCF(t))\right) \\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{scale(PCF(t))} e^{-x^2/2}\,dx \\ &= \frac{1}{2} * \left[erf\left(\frac{scale(PCF(t))}{\sqrt{2}} \right) + 1 \right] \end{aligned} \qquad 5\text{-}19$$

where *erf()* is the "error function" encountered in integrating the normal distribution; *scale()* is the function to scale the priority of the intention at time *t*, *PCF*(*t*), from the domain [1, 100] to [-3 ,3]. The domain [-3, 3] is selected because for standard normal distribution function, the probability that the variable is outside [-3, 3] is very small ($D_{standard\text{-}normal}(-3)\approx 0.00135$, $D_{standard\text{-}normal}(3)\approx 0.99865$). A figure of this distribution function is shown in Figure 5-12.

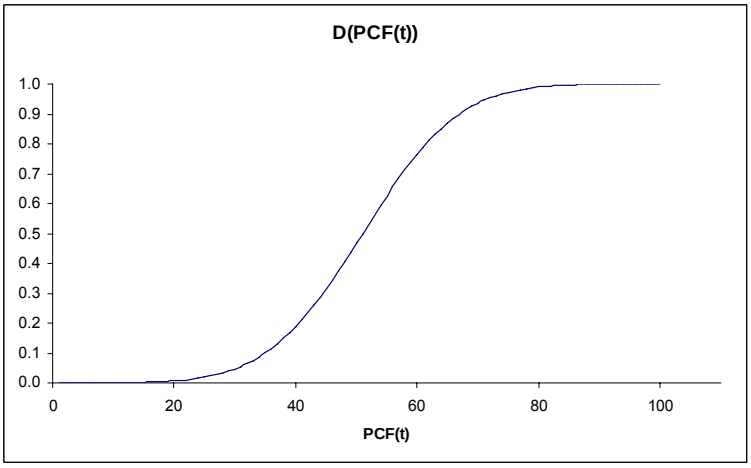


Figure 5-12 Demonstration of function *D(t)*.

The calculation of *erf()* function can be seen in many mathematics articles about normal distribution, such as [113], [2]. Here we show it for reference.

$$erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{n!(2n+1)} \quad 5\text{-}20$$

The calculation results of the two probability functions are shown and analyzed in the following.

5.5.1 Probability that the intention is running at *t*

The following results are calculated by Equation 5-17. Four intentions with different parameter settings of t_m and MP are designed. The probabilities that the intentions are running at t are shown in Figure 5-13.

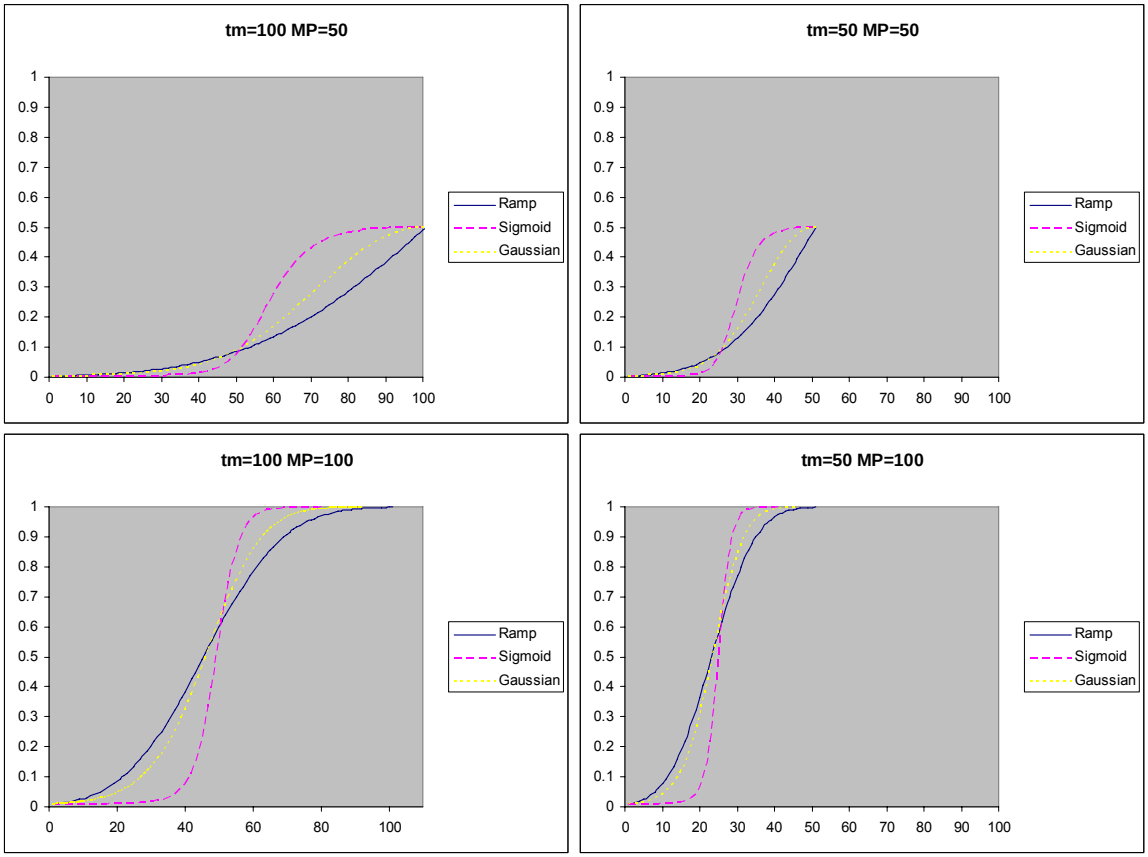


Figure 5-13 Probability that the intention is running at *t*.

From the figure, it is shown that with a same t_m, a higher MP will increase the probability that the intention is running. With a same MP, a shorter t_m will also increase the probability because the priority is increased more quickly. With the same t_m and MP, the intention with Sigmoid reminding function will be running in the initial period with the smallest probability. However, when approaching t_m, the probability that the intention with Sigmoid reminding function is running is the greatest.

5.5.2 Probability that the intention is started first time at *t*

The following results are calculated by Equation 5-18. The results will show when the intention will be started first time at the largest probability. MP of the intention is set as 50. The intention is associated with different PCF functions and parameter settings of t_m. The results are shown in Figure 5-14.

It can be seen that if t_m is larger than 30 in the experiment setting, the probability that the intention is started before t_m is very close to 100%. However, with a small t_m, such as 10, the intention may not be started before t_m. With the same t_m, the probability that an intention with Sigmoid PCF is started is smallest in the initial period. Then it is increased quickly. The intention is likely to be started in a period shorter than the intention with other PCFs.

From the above analysis, it can be seen that with a Sigmoid reminding function, the intention will be started later than with a Gaussian reminding function as we expect: Sigmoid function models people who tend to leave things to “last minute” where Gaussian function models people who tend not to do so.

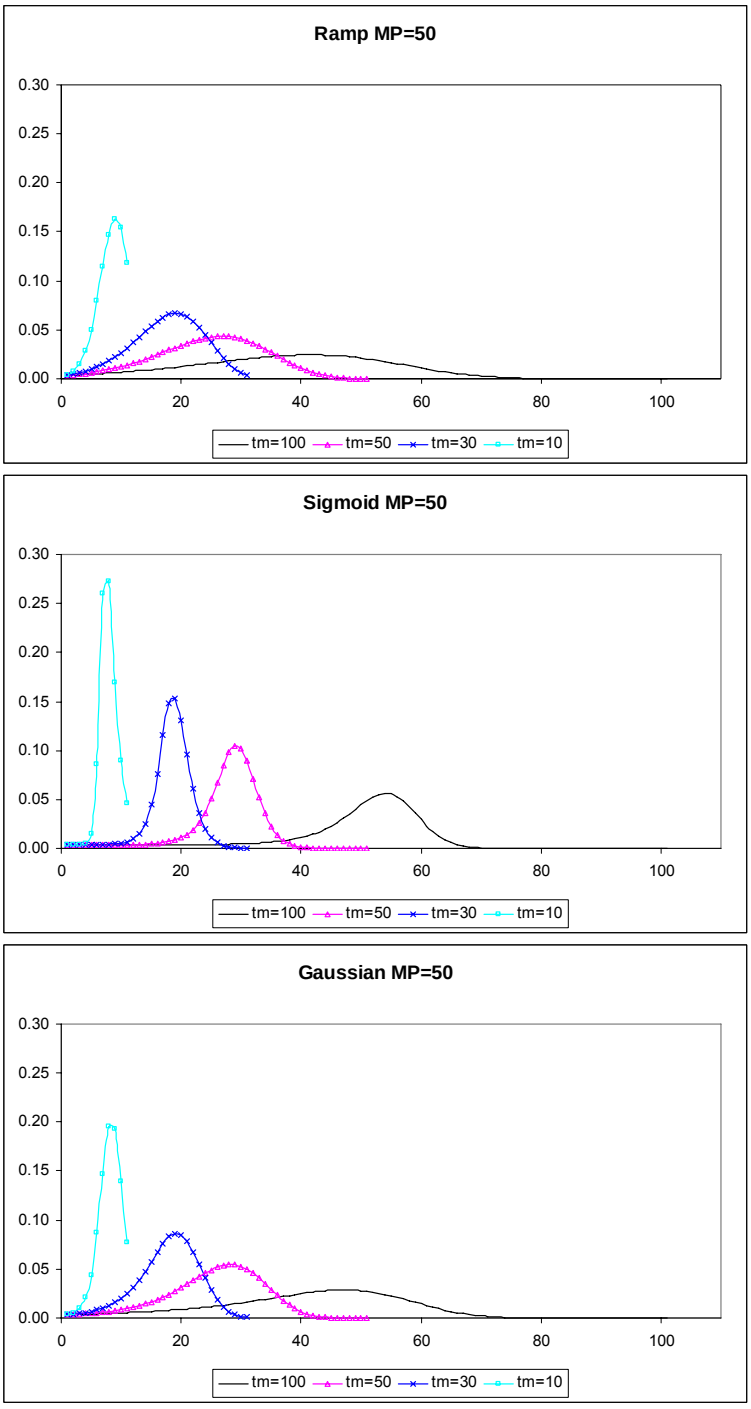


Figure 5-14 Probability that the intention is started at *t* first time.

5.6 Conclusion

In this chapter, we first argue that the priority of the deliberations/intentions in an agent should be changing with time. Then we design a "priority control" extension to the parallel BDI agent. For each deliberation/intention, the agent will choose a PCF for it from the PCF library. By priority, it provides a way to schedule the deliberations/intentions in the agent.

We design a *reminding-forgetting* PCF by simulating human bahaviours when dealing with several tasks together. We propose three functions to simulate the reminding phase and use the forgetting curve function for the forgetting phase. A comparison experiment of the agents with or without the *reminding-forgetting* PCF is shown. The agent behaviours with the three reminding functions are analyzed.

CHAPTER

6

A VESSEL CAPTAIN AGENT

In this chapter, we will show a software agent which simulates the behaviours of a vessel captain navigating in the sea. The agent architecture for implementing the software agent is an instantiation of the parallel BDI agent framework with the two agent character extensions. Then the simulation experiments are shown.

The purpose of this experiment is to apply the parallel agent framework to make a real software agent. The behaviour records of the vessel agent demonstrate the abilities that we expect from a parallel agent. The software agent architecture we show here can be used to make software agents in other contexts.

This chapter is structured as follows. In Section 6.1, the software agent architecture is shown and explained. The experiment designation and results are shown in Section 6.2. A conclusion is given in the last section.

6.1 Software Agent Architecture

When an agent is simulating a certain physical system like a human being, the parallel agent under the general framework should be configured such that it has the same number

of Environment Monitors (EMs), Plan Generators (PGs) and Plan Executors (PEs)as the number of parallel physical devices that exist in the physical system to perform the corresponding functions. Figure 6-1 shows the detailed software agent architecture for a vessel captain navigating a vessel in the sea.

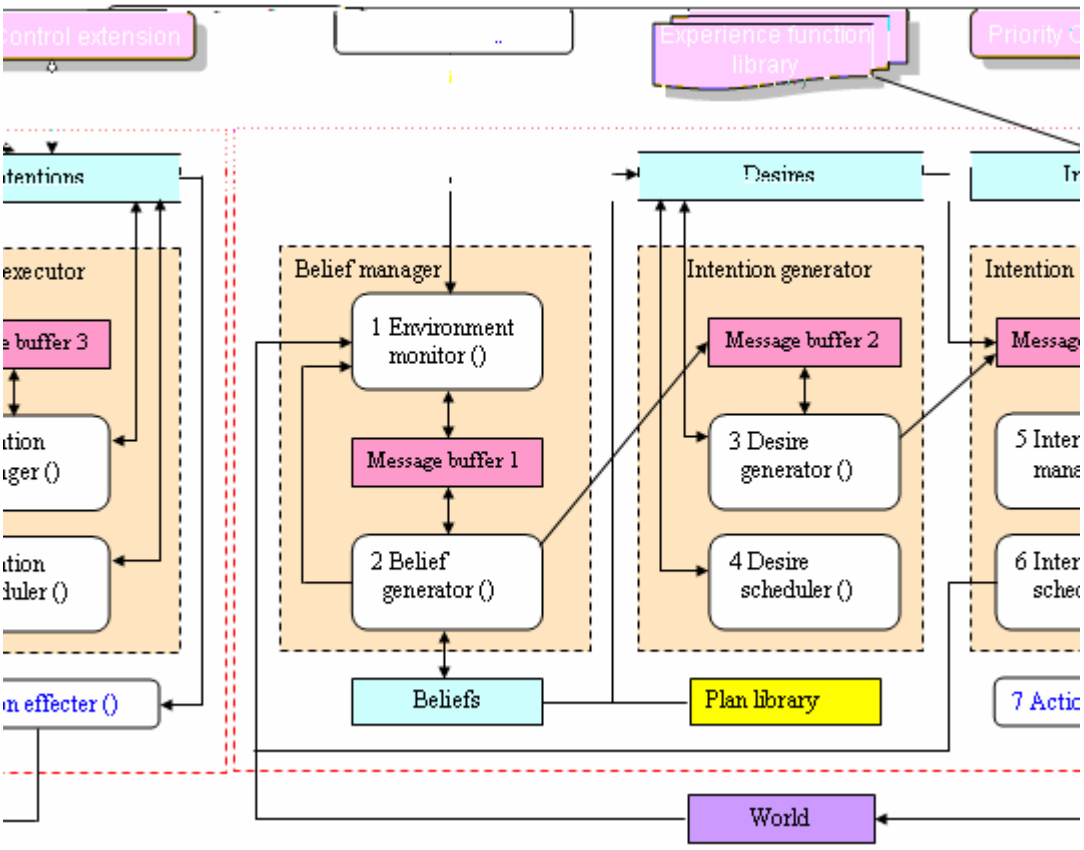


Figure 6-1 Software implementation architecture.

Threads are used to simulate the parallel processing elements in the agent. In Figure 6-1, these threads are shown in white boxes and identified with numbers (1-6). For example, the belief manager consists of two threads, numbered by 1 and 2. Similarly, the intention generator and the intention executor are made of several threads of their own. The information flows in the architecture are shown by the dark arrow lines. The link from the *Intention scheduler* to EM is to inform it that a certain intention plan has been completed.

The PG and PE are not explicitly shown in Figure 6-1. Desires and intentions are implemented as separate threads. A running thread of desire/intention is to simulate a PG/PE. This makes it very easy to manage the suspension and the subsequent resuming of the plan generation of a desire in the PG and that of the plan execution of an intention in the PE. If a thread is suspended, the current working status of the thread is saved in computer memory automatically for resuming the thread later on. The *Desire scheduler* and the *Intention scheduler* will schedule the running of the threads according to the priorities of the desires and intentions respectively.

In the following experiment of vessel agent, there is just one PG in the agent. This means that at any time, just one desire thread can be activated. This is to simulate the human behaviour that at any time, we deliberate on or think about one matter. To keep the experiment simple, the agent's actions only include vessel maneuvering and only one PE is simulated in the agent. Of course, it is straightforward and easy to realize to start another PE thread so that the vessel is able to sound the siren and one more PE thread to communicate with other vessels or marine authorities. The number of EMs is also one for the collection of information about its surroundings by the vessel.

Two more threads, the action effecter (7) and the interface thread, do not belong to the BDI agent architecture. They are components of the simulation system. The action effecter thread will execute output physical actions from the intention executor. In our simulation, this action effecter thread is responsible for updating the world map with the new position of the vessel which is calculated using physical motion laws. The Interface thread will receive commands through user input and send the commands to the belief manager.

All the threads run in parallel. Message buffers in the three components are used to save the incoming messages from one processing element to another. The messages in these buffers are sequenced by their priorities so that high priority messages are handled first.

A global path planning algorithm and RBF neural network learning algorithm are included in the experience function library to provide the agent abilities of path planning and obstacle avoidance. The priority control extension works in the parallel agent as described in Chapter 5. The *reminding-forgetting* PCF is used to control the priorities.

6.2 Experiment

The agent is simulated using Visual J# 2003. The navigation of the vessel is shown in a windows graphic interface. The history of processing of the agent is recorded in a Microsoft Access database. With these records, we can find out how the agent processes the events. In addition, a map editor is designed to create the world map which consists of some islands. The maps can be loaded by the agent simulation program. In the following, we will first show the designation details of the simulation program. Then an example of the vessel navigation is shown and the records of the agent behaviours are analyzed.

6.2.1 System design

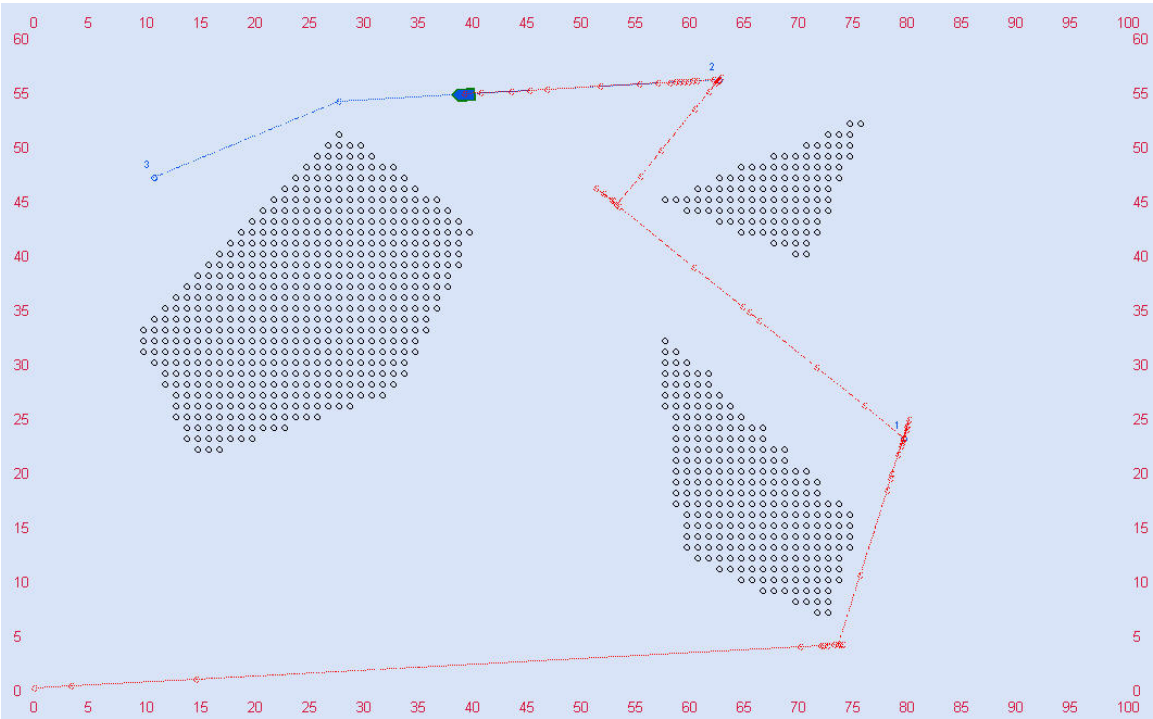


Figure 6-2 Program interface.

The agent can detect the world information through sensors and the output actions consist of commands of changing translational and rotational speeds. Vessel agents are created with their specific physical parameters. The vessel agent's destinations are given when the vessel is created. User can give commands to the vessel at any time through the command input window, such as changing or adding commands. The objective of the agent is to move to the destinations safely and quickly. A snapshot of this program can be seen in Figure 6-2. In the experiment, the agent will reach three destinations denoted in the figure. The agent needs to avoid the islands while navigating. A global path planning algorithm is adopted to search the paths among the islands. The algorithm is shown first. Then the details about the agent are explained.

6.2.1.1 Path planning algorithm

We adopt the visibility graph method [76] for path planning. The algorithm demands that the nodes of the obstacles are prior known. Then all the possible combinations of the nodes are searched to find an optimal path with the shortest distance. A demonstration is shown in Figure 6-3.

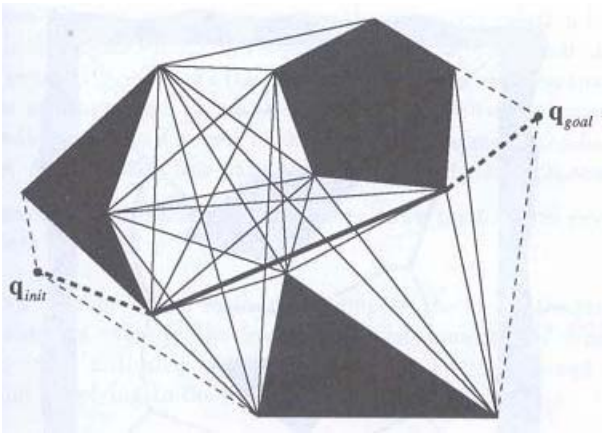


Figure 6-3 The visibility graph (from [76]).

We use a recursive algorithm to realize the method. The algorithm is shown in Figure 6-4.

```
compute (Vector path, Vector candidates)
{//originally, path consists of the starting point, candidates consists of the nodes of the obstacles
    If the last point in the path and the destination are not blocked
        Add the destination in the path
        Calculate the length of the path
        If the length is shorter than the saved_length
            saved_length=the length
            saved_path=path
        End if
        return;
    End if

    For each point in the candidates
        If the point and the last point in the path are not blocked
            Vector newpath=path
            Vector newcandidates=candidates
            Add the point in the newpath
            Remove the point from the newcandidates
            compute (newpath, newcandidates)
        End if
    End
}
```

Figure 6-4 Algorithm for calculating the global path.

Obviously, the time needed for this algorithm will be increased with the number of the nodes of the obstacles. In our experiment, it takes over 4 seconds to calculate a path in a map with 12 nodes.

6.2.1.2 The agent implementation

In order to achieve its objective, the vessel agent should have the abilities to plan its path, to output actions which make it travel to the destination, and to avoid obstacles. These behaviours are triggered by new beliefs of the agent. The corresponding plans are designed in reaction to the new beliefs. The descriptions of the four levels of priorities for the vessel agent's beliefs are given in Table 4-1.

For a simulation system, all physical data and structure need to be symbolized. We apply object-oriented approach to program the vessel simulation system. The vessel agents and

environment are implemented as objects. Each vessel agent exchanges information with the environment individually. The processing of the *Environment monitor* in the belief manager is shown in Figure 6-5.

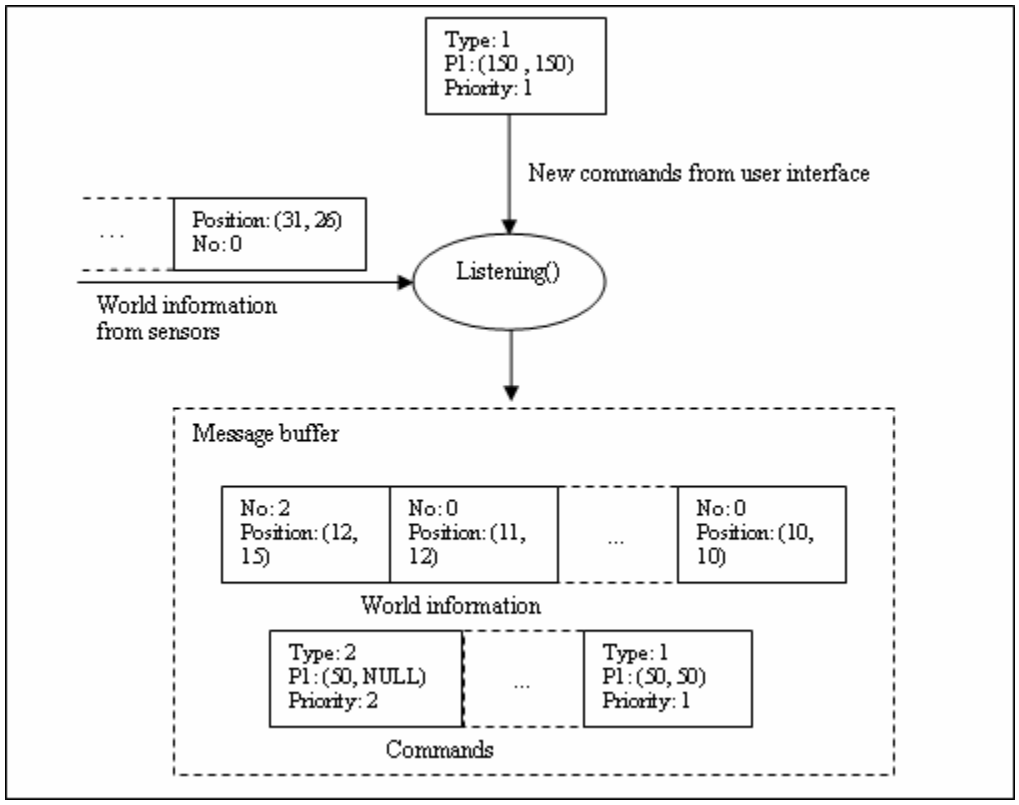
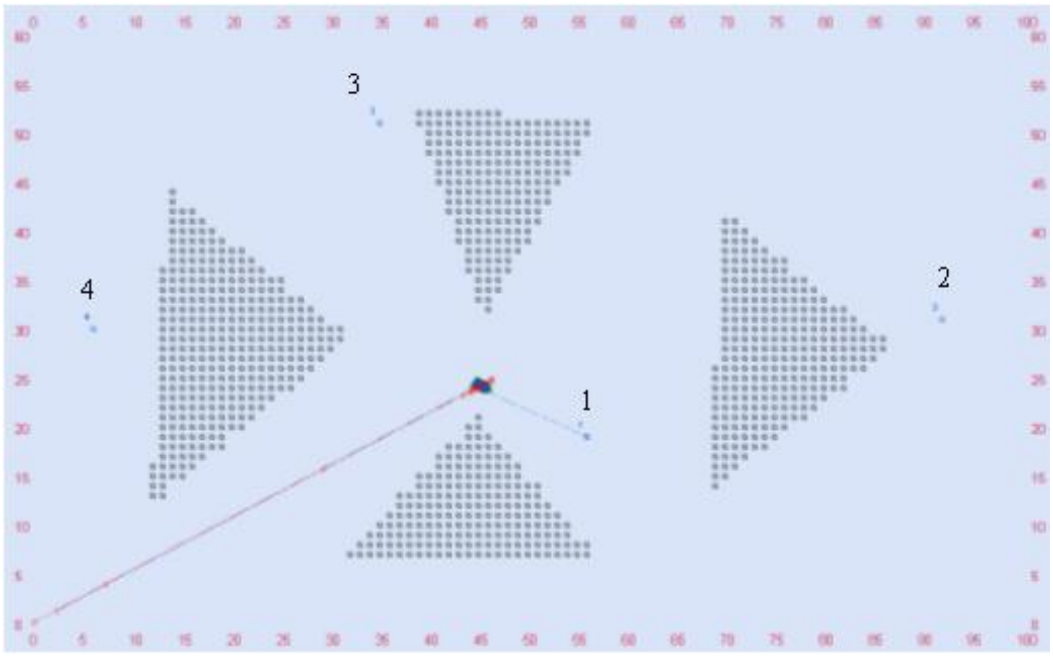


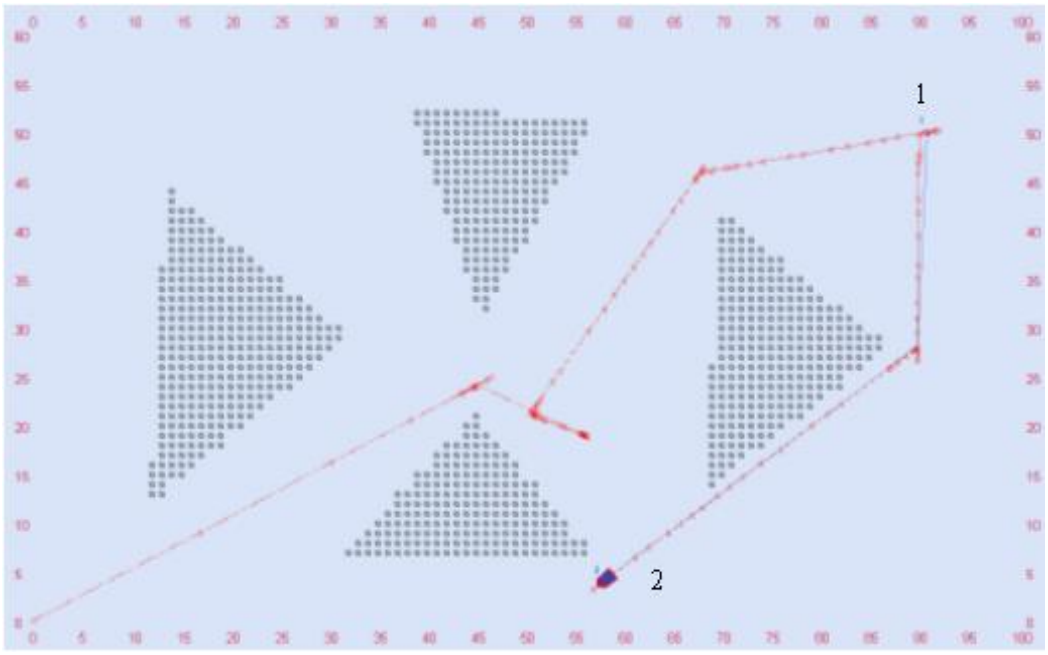
Figure 6-5 Environment monitor thread in Belief manager.

6.2.2 Experiment result

A navigation example is shown in Figure 6-6. The area of interests is shown in a two-dimensional rectangular sea area. There are four islands shown by the shaded areas in the map. As shown in Figure 6-6 a, the vessel agent at start point (0,0) is given four destinations 1, 2, 3, and 4 to move to. Then a change of plan, that is, new destinations 1 and 2, as shown in Figure 6-6 b are given when the vessel is moving to the old destination 1. Then the vessel navigates to the new destinations. The behaviours of the corresponding threads are recorded. Only the most relevant activities are shown in Table 6-1. These activities are carried out by the belief manager, intention generator and the intention executor of the agent as stated before.



a. Destinations are changed at this moment.



b. The vessel navigates to the destinations 1, 2.

Figure 6-6 Vessel navigation.

Table 6-1 Processing records of the vessel agent

ID	Insert_Time	Type	Behaviour
492	7:32:20 PM	Agent1	Start!
493	7:32:20 PM	deliberation thread0	Created. Status: New Target
494	7:32:20 PM	deliberation thread1	Created. Status: Idle
495	7:32:21 PM	deliberation thread0	Start. Status: New Target
496	7:32:21 PM	deliberation thread0	finished. Status: New Target
497	7:32:21 PM	intention thread0	Created. Intention: 0/1.
498	7:32:21 PM	deliberation thread1	Start. Status: Idle
499	7:32:21 PM	intention thread0	Start. Intention: 0/1.
…			
504	7:32:23 PM	intention thread1	Start. Intention: 10.
…			
507	7:33:06 PM	deliberation thread2	Start. Status: The global path updated
508	7:34:38 PM	deliberation thread2	finished. Status: The global path updated
509	7:34:55 PM	intention thread2	Created. Intention: 2/3.
510	7:34:56 PM	suspend intention1.0	2.2 Start.
511	7:34:56 PM	suspend intention1.0	2.2 Suspended.
512	7:34:57 PM	intention thread2	Start. Intention: 2/3.
513	7:34:57 PM	Subgoal reached	(0,0)
514	7:34:57 PM	intention thread2	finished. Intention: 2/3.
515	7:35:05 PM	resume intention	intention1
…			
523	7:35:57 PM	Action	Created. action: Accelerate to max speed.
…			
534	7:37:40 PM	Remove obsolete intention	1
535	7:37:40 PM	intention thread5	Created. Intention: 2/3.
536	7:37:40 PM	deliberation thread6	Created. Status: Targets changed
537	7:37:40 PM	deliberation thread7	Created. Status: New Target
538	7:37:40 PM	deliberation thread8	Created. Status: Idle
539	7:37:39 PM	intention thread4	Start. Intention: 4/5.
540	7:37:40 PM	Action	Created. action: Decelerate to min speed.
541	7:37:40 PM	intention thread4	finished. Intention: 4/5.
542	7:37:39 PM	deliberation thread5	finished. Status: Subgoal reached
543	7:37:41 PM	deliberation thread6	Start. Status: Targets changed
…			
568	7:40:16 PM	deliberation thread11	Start. Status: The global path updated
569	7:40:17 PM	deliberation thread11	finished. Status: The global path updated
570	7:40:13 PM	intention thread9	Start. Intention: 4/5.
571	7:40:17 PM	intention thread11	Created. Intention: 2/3.
572	7:40:17 PM	Action	Created. action: Decelerate to min speed.

The activities of the agent are shown in Table 6-1. The sequence of the activities illustrates the behaviour of the agent.

1. The new beliefs of the agent are processed in the order of priority. When the agent is started, new beliefs of the need to go to destination 1 followed by 2, 3 and 4 are received. After the beliefs are sent by the belief manager, the intention generator create deliberation threads 0 and 1 (events 493, 494) and starts deliberation thread 0 (event 495) while deliberation thread 1 stays in the deliberation queue. Deliberation thread 1 is something the agent needs to deliberate on when it has time as shown by event 498.
2. The agent is able to carry out several activities at once. As shown by event 498 and 499, the agent is executing deliberation thread 1 and intention thread 0 simultaneously, that is, the agent is processing the belief of the need to go to destination 2, 3, 4 and the intention of planning the path to destination 1 at once.
3. When there is a more urgent intention, the intention with low priority can be suspended and resumed later. As shown by events 509, 510 and 515, intention 2 is one of the steps in following the path plan to go to destination 1 while intention 1 is to compute the path to destination 2, 3, 4. Intention 2 preempts intention 1.
4. When an intention becomes obsolete because of some new beliefs, the agent is able to stop and remove it. As shown by event 534, intention 1 which is being executed by the intention executor is stopped and removed. This is caused by an interrupt from the intention generator when it is processing the belief that the vessel should change course and go to the new destinations 1 and 2.
5. The agent can respond to circumstance changes rapidly. Event 523 shows the agent is moving to the old destination 1. Event 543 shows the agent is processing the new belief that the destination is changed. Event 569 shows the processing of the belief that vessel path is updated in response to the new destinations. Event 572 shows the vessel is changing course when it is moving to the old destination 1.

In summary, the agent is able to prioritize its deliberation and intention execution and is able to reconsider its goals and intentions after they are generated.

The complete records for this example can be found in Appendix A.

6.3 Conclusion

In this chapter, the software agent architecture for implementing a parallel BDI agent is demonstrated. Then a vessel captain agent is realized. The behaviours of the vessels are demonstrated and historical behaviour records are explained.

The software agent architecture is also useful when designing agents in other aspects. In some sense, this proves the applicability of the general parallel BDI agent framework we propose in Chapter 3.

CHAPTER

7

CONCLUSIONS AND FUTURE WORK

In this thesis, parallel BDI agent architecture is designed for simulating the vessel captain. The agent architecture is general enough to make other agents. Two components are incorporated into the architecture to provide the agent some human-like characters. In this final chapter, we will first make a conclusion about the contributions of our research. Then some possible future researches and applications are proposed.

7.1 Conclusions

In Chapter 3, a general framework for real time performance in the BDI model is proposed. It is a parallel agent architecture that supports the following agent abilities at architecture level: (1) the ability to respond to emergencies timely; (2) the ability to reconsider and modify goals, intentions and actions in reaction to unexpected or new information; (3) the ability to perform multiple actions at once; (4) the ability to perceive, deliberate and act simultaneously; (5) the ability to prioritize the deliberations and intention executions. We defined the functions and the operations of the processing units in the agent and how these units interact and cooperate with each other. With the advances in semiconductor technology which allow multiple CPUs to be implemented on

the same silicon chip, a parallel BDI agent will be an effective way to enable it to perform in real time when the arrival rate of events is high.

To evaluate the parallel BDI model, a comparison experiment is done with the sequential BDI agents. Five ways of organizing and controlling a sequential BDI agent are studied. The sequential agents and the parallel agent are evaluated by simulating their operations in processing events of different priorities and examining their performance. We analyze their performance in handling the same sequences of events. The results show that the parallel BDI agent outperforms the sequential ones in offering significantly shorter response time to events with various inter-arrival times. The parallel BDI agent with its interrupt mechanism is able to guarantee to immediately react to high priority events where none of the sequential ones are capable of.

Then the agent character is studied. We analyze the agent character from the agent itself and propose an extended BDI architecture for implementing such characters. The basic character consists of two parts: personality and experience. The personality will affect the whole process of the agent. In the vessel agent, it is represented by different initial parameter settings. The parameter settings will decide the agent's basic physical properties and actions. The experience is implemented using the reinforcement learning algorithm. With the extended BDI architecture, the learning algorithm is combined into the agent as an experience function. In the experiment, the learning algorithm is used for improving the agent's skills of obstacle avoidance. The behaviours of the agent will be affected by the experience accumulated by the learning algorithm. The experiment results show that the agent built using this method will have different behaviour with different parameter settings and different past experience. This is important for realistic simulation of systems made up by different people.

Furthermore, we propose to enrich the BDI agent architecture with 2 processing components, a PCF (Priority Changing Function) Selector and a Priority Controller. The priorities of the desires/intentions can have different initial values and can change with time according to the chosen PCF. The desires/intentions can be scheduled according to

their own nature. It is a method to realize the operations defined in the parallel BDI agent framework by controlling priority of the desires/intentions. This also provides us a chance to control the scheduling of the intentions in a more human-like manner. As an example, a *reminding-forgetting* PCF is designed that simulates the way that human deals with several intentions together. The PCF goes through a rising/reminding phase and then a descending/forgetting phase. We proposed to use the Ramp, Sigmoid or Gaussian function to simulate the reminding processes of the intentions. And a forgetting function based on Ebbinghaus forgetting curve is used to simulate the function in forgetting process. With the setting of different parameter values, this PCF may also simulate a priority function that has the forgetting phase alone or the reminding phase alone. From the experiment results, we can see that the resulted intention scheduling behaviour and the effect of setting the different parameter values. The agent can show some human behaviour.

At the end, the software architecture for realizing the parallel BDI agent framework is proposed. The experience function library and the *reminding-forgetting* PCF are combined to simulate the agent characters. The architecture is used to implement the vessel agent. The experiment demonstrates an example of agents based on the parallel BDI agent architecture.

7.2 Proposals

7.2.1 A parallel hybrid agent architecture

The current parallel BDI agent architecture is a kind of deliberative agent architecture. Though the parallel designation promotes the processing speed of the agent, the disadvantages of the deliberative agent still limit the parallel agent. So we propose that the architecture can be extended into a hybrid agent architecture. As shown in Figure 7-1, the layers in the reactive architecture can be run parallel with the intention generator and the intention executor. The original belief manager will have a new function to filter the events into the reactive or deliberative components. Issues of how to decide what needs

quick reaction and what needs deliberation will be interesting to explore. Then the outputs from the two components will be combined by some context-rules.

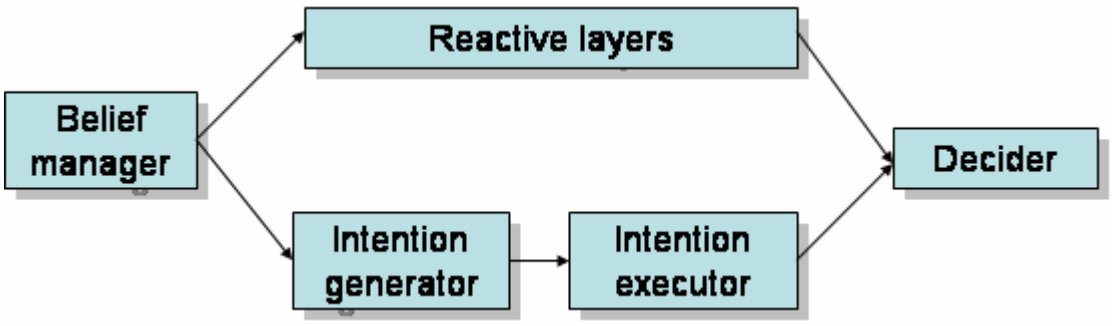


Figure 7-1 A parallel hybrid agent architecture.

7.2.2 Applications to real robots

Because the parallel BDI agent model demands several CPUs to produce its best performance, currently the most directive and possible application will be robot control. The robot may have several parallel processors to use instead of the threads in the previous software simulation experiments. The robot with such designation is expected to show more natural human behaviours. The robot can detect, think and act at the same time. And the emergencies can be dealt with immediately by suspending some normal processing. The human-like ability to learn from experience and to prioritize tasks to be done will be very useful for robots which are taking care of some patients or old people.

Reference

[1] *Forgetting curve - Wikipedia, the free encyclopedia*, http://en.wikipedia.org/wiki/Strength_of_memory.

[2] *Erf -- from Wolfram MathWorld*, http://mathworld.wolfram.com/Erf.html.

[3] *The Merriam-Webster Dictionary*, Merriam-Webster, incorporated, Springfield, Massachusettes, U.S.A., 1994.

[4] S. Ambroszkiewicz and J. Komar, A Model of BDI-Agent in Game-Theoretic Framework, in *Proceedings of Formal Models of Agents, ESPRIT Project ModelAge Final Workshop*, 1997, pp. 8-19.

[5] E. Amir and P. Maynard-Reid II, Logic-Based Subsumption Architecture, in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, July 31-August 6, 1999, pp. 147-152.

[6] E. Amir and P. Maynard-Reid II, LiSA: A robot driven by logical subsumption, in *Proceedings of Working Notes of the CommonSense01 Symposium*, New York, 2001.

[7] M. Balmer, N. Cetin, K. Nagel and B. Raney, Towards Truly Agent-Based Traffic and Mobility Simulations, in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, New York, NY, USA, August 19-23, 2004, pp. 60-67.

[8] G. Bauzil, M. Briot and P. Ribes, A Navigation Sub-System Using Ultrasonic Sensors for the Mobile Robot HILARE, in *Proceedings of 1st In. Conf. on Robot Vision and Sensory Controls*, Stratford-upon-Avon, UK, 1981, pp. pp. 47-58 and pp. 681-698.

[9] R. E. Bellman, *An Introduction to Artificial Intelligence: Can Computers Think?*, Boyd and Fraser Publishing Company, 1978.

[10] J. A. Benayas, J. L. Fernάndez, R. Sanz and A. R. Diéguez, The Beam-Curvature Method: A New Approach for Improving Local Real-time Obstacle Avoidance, in *Proceedings of 15th Triennial World Congress*, Barcelona, Spain, 2002.

[11] R. P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. P. Miller and M. G. Slack, Experiences with an Architecture for Intelligent, Reactive Agents, *Experimental & Theoretical Artificial Intelligence*, **9**, (1997), pp. 237--256.

[12] R. H. Bordini, A. L. C. Bazzan, R. d. O. Jannone, D. M. Basso, R. M. Vicari and V. R. Lesser, AgentSpeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling, in *Proceedings of International Conference on Autonomous Agents*, 2002, pp. 1294-1302.

[13] J. Borenstein and Y. Koren, Real-time Obstacle Avoidance for Fast Mobile Robots, in *Proceedings of International Conference on CAD/CAM, Robotics and Factories of the Future (CARS&FOF)*, Detroit, Michigan, August 17, 1988.

[14] J. Borenstein and Y. Koren, Fast Obstacle Avoidance for Mobile Robots, in *Proceedings of Tenth Israel Convention on CAD/CAM and Robotics*, Tel-Aviv, Israel, December 27-29, 1988.

[15] J. Borenstein and Y. Koren, High-speed Obstacle Avoidance for Mobile Robots, in *Proceedings of the IEEE Symposium on Intelligent Control*, Arlington, Virginia, August 24-26, 1988, pp. 382-384.

[16] J. Borenstein and Y. Koren, Obstacle Avoidance With Ultrasonic Sensors, *IEEE Journal of Robotics and Automation*, **RA-4**(2), (1988), pp. 213-218.

[17] J. Borenstein and Y. Koren, The Vector Field Histogram -- Fast Obstacle-Avoidance for Mobile Robots, *IEEE Journal of Robotics and Automation*, **7**(3), (1991), pp. 278-288.

[18] M. E. Bratman, *Intentions, Plans, and Practical Reason*, Harvard University Press, Cambridge, MA, 1987.

[19] F. Brazier, B. Dunin-Keplicz, J. Treur and R. Verbrugge, Beliefs, Intentions and DESIRE, The Netherlands and Poland., in *Proceedings of the 10th Workshop on Knowledge Acquisition and Knowledge Base Systems*, Voyager Inn, Banff, Alberta, Canada, November 8-14, 1996.

[20] O. Brock and O. Khatib, High-Speed Navigation Using the Global Dynamic Window Approach, in *Proceedings of Video Proceedings of the International Conference on Robotics and Automation*, San Francisco, USA, 2000.

[21] R. A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, **2**(1), (1986), pp. 14-23.

[22] R. A. Brooks, *Elephants Don't Play Chess*, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, The MIT press, London, 1991, pp. 3-15.

[23] R. A. Brooks, Intelligence without Representation, *Artificial Intelligence*, **47**, (1991), pp. 139-159.

[24] D. Caragea, A. Silvescu and V. Honavar, Multi-Agent Learning from Distributed Data Sources, in *Proceedings of the Workshop on Multi-Agent learning: Theory and Practice, International Conference on Machine Learning (ICML-2000)*, Stanford University, Stanford CA, 2000.

[25] D. Castro, U. Nunes and A. Ruano, Reactive Local Navigation, in *Proceedings of 28th Annual Conference of the IEEE Industrial Electronics Society - IECON'02*, Sevilla, November 5-8, 2002.

[26] D. Choi, M. Kaufman, P. Langley, N. Nejati and D. Shapiro, An Architecture for Persistent Reactive Behavior, in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, New York, NY, USA, August 19-23, 2004, pp. 988-995.

[27] M. C. Choy, D. Srinivasan and R. L. Cheu, Cooperative, Hybrid Agent Architecture for Real-Time Traffic Signal Control, *IEEE Transactions on Systems, Man and Cybernetics - Part A: systems and humans*, **33**(5), (2003), pp. 597- 607.

[28] M. d'Inverno, D. Kinny, M. Luck and M. Wooldridge, A Formal Specification of dMARS, in *Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages*, Providence, Rhode Island, USA, July 24-26, 1997, pp. 155--176.

[29] M. d'Inverno and M. Luck, Engineering AgentSpeak(L): A formal computational model, *Logic and Computation*, **8**(3), (1998), pp. 233--260.

[30] M. E. DesJardins, E. H. Durfee, J. Charles, L. Ortiz and M. J. Wolverton, A survey of research in distributed, continual planning, *AI Magazine*, **21**(4), (2000).

[31] K. Dresner and P. Stone, Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism, in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, New York, NY, USA, August 19-23, 2004, pp. 530-537.

[32] K. Dresner and P. Stone, Multiagent Traffic Management: An Improved Intersection Control Mechanism, in *Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, Utrecht, The Netherlands, July 25-29, 2005, pp. 471-477.

[33] K. Erol, J. Hendler and D. S. Nau, *Semantics for Hierarchical Task-Network Planning*, Technical report CS-TR-3239, UMIACS-TR-94-31, ISR-TR-95-9 at the University of Maryland, 1994.

[34] K. Erol, J. Hendler and D. S. Nau, HTN planning: Complexity and expressivity, in *Proceedings of the Twelfth National Conf. on Artificial Intelligence*, 1994, pp. 1123-1128.

[35] I. A. Ferguson, *TouringMachines: An architecture for dynamic, rational, mobile agents*, PhD Thesis, Comput. Sci. Lab, University of Cambridge, Cambridge, U.K, 1992.

[36] I. A. Ferguson, *Autonomous agent control: a case for integrating models and behaviors*, *Working Notes AAAI Fall Symposium on Control of the Physical World by Intelligent Agents*, New Orleans, LA, 1994, pp. 46---54.

[37] P. Fiorini and Z. Shiller, Motion Planning in Dynamic Environments using Velocity Obstacles, *Int. Journal on Robotics Research*, **17**(7), (1998), pp. 711-727.

[38] D. Fox, W. Burgard and S.Thrun, The dynamic window approach to collision avoidance, *IEEE Robotics and Automation*, **4**(1), (1997).

[39] S. S. Ge and Y. J. Cui, Path Planning for Mobile Robots Using New Potential Functions, in *Proceedings of the 3rd Asia Control Conference*, Shanghai, China, July 4-7, 2000.

[40] P. Gebhard, M. Kipp, M. Klesen and T. Rist, Authoring scenes for adaptive, interactive performances, in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, Melbourne, Victoria, Australia, July 14-18, 2003, pp. 725-732.

[41] P. Gebhard, M. Kipp, M. Klesen and T. Rist, Adding the Emotional Dimension to Scripting Character Dialogues, *Lecture Notes in Computer Science*, **2792**, (2003), pp. 48 - 56.

[42] P. Gebhard, M. Klesen and T. Rist, Coloring Multi-character Conversations through the Expression of Emotions, in *Proceedings of Affective Dialogue Systems, Tutorial and Research Workshop, ADS 2004*, Kloster Irsee, Germany, June 14-16, 2004 pp. 128-141.

[43] P. Gebhard, ALMA – a layered model of affect, in *Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, Utrecht, The Netherlands, July 25-29, 2005, pp. 29-36.

[44] M. Georgeff, B. Pell, M. Pollack, M. Tambe and M. Wooldridge, *The Belief-Desire-Intention Model of Agency*, in M. J.P., S. M. and R. A., eds., *Intelligent Agents V: Theories, Architectures, and Languages*, Springer-Verlag, Berlin, 1999.

[45] C. Goldman and J. Rosenschein, Mutually Supervised Learning in Multiagent Systems, in *Proceedings of IJCAI-95 Workshop on Adaptation and Learning in Multiagent Systems*, 1995.

[46] A. Guerra-Hernandez, A. El Falla-Seghrouchni and H. Soldano, Learning in BDI Multi-agent Systems, in *Proceedings of CLIMA IV- Computational Logic in Multi-Agent Systems*, Fort Lauderdale, FL, USA, January 6-7, 2004, pp. 218-233.

[47] B. Hamner, S. Scherer and S. Singh, Learning Obstacle Avoidance Parameters from Operator Behavior, in *Proceedings of the NIPS 2005 Workshop on Machine Learning Based Robotics in Unstructured Environments*, Dec, 2005.

[48] B. Hayes-Roth, Architectural Foundations for Real-Time Performance in Intelligent Agents, *Real-Time Systems*, **2**, (1990), pp. 99-125.

[49] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan, New York, 1994.

[50] J. Heinström, The impact of personality and approaches to learning on information behaviour, *Information Research*, **5**(3), (2000).

[51] J. Heinström, Five personality dimensions and their influence on information behaviour, *Information Research*, **9**(1), (2003).

[52] A. E. Henninger, R. M. Jones and E. Chown, Behaviors that emerge from emotion and cognition: Implementation and evaluation of a symbolic-connectionist architecture, in *Proceedings of the Seventh International Conference on Autonomous Agents*, Melbourne, Australia, 2003.

[53] C. Hewitt, Viewing Control Structures as Patterns of Passing Messages, *Artificial Intelligence*, **8**(3), (1977), pp. 323-364.

[54] K. V. Hindriks, F. S. d. Boer, W. v. d. Hoek and J. C. Meyer, A formal embedding of agentspeak(L) in 3APL, in *Proceedings of Advanced Topics in Artificial Intelligence, 11th Australian Joint Conference on Arti cial Intelligence (AI'98)*, Brisbane, Australia, July 13-17, 1998.

[55] B. Horling, V. Lesser and R. Vincent, Multi-Agent System Simulation Framework, in *Proceedings of the 16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation*, Lausanne, Switzerland, August 21-25, 2000.

[56] N. Howden, R. Ronnquist, A. Hodgson and A. Lucas, JACK Intelligent Agents™ - Summary of an Agent Infrastructure, in *Proceedings of the 5th International Conference on Autonomous Agents*, Montreal, Canada, May 28-June 1, 2001.

[57] H. Hu, I. Kelly, D. A. Keating and D. Vinagre, Coordination of Multiple Mobile Robots via Communication, in *Proceedings of SPIE'98, Mobile Robots XIII Conference*, Boston, USA, November, 1998, pp. 94-103.

[58] H. Hu, K. Kostiadis and Z. Liu, Coordination and Learning in a team of Mobile Robots, in *Proceedings of the IASTED Robotics and Automation Conference*, Santa Barbara, CA, USA, October, 1999, pp. 28-30.

[59] M. J. Huber, JAM: A BDI-theoretic mobile agent architecture, in *Proceedings of third International Conference on Autonomous Agents (Agents '99)*, 1999, pp. 236-243.

[60] M. J. Huber, *JAM Agents in a Nutshell*, 2001.

[61] M. Hunter, K. Kostiadis and H. Hu, A Behaviour-based Approach to Position Selection for Simulated Soccer Agents, in *Proceedings of 1 st European Workshop on RoboCup*, Amsterdam, May 28 - June 2, 2000.

[62] K. Hwang, H. C. Hsu and A. Liu, A Homogeneous Agent Architecture for Robot Navigation, in *Proceedings of IEEE 2003 International Conference on Neural Network And Signal*, 2003, pp. 310-313.

[63] F. F. Ingrand, M. P. Georgeff and A. S. Rao, An architecture for real-time reasoning and system control, *IEEE Expert*, **7**(6), (1992), pp. 34--44.

[64] D. Isla, R. Burke, M. Downie and B. Blumberg, A layered brain architecture for synthetic creatures, in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, Seatle, Washington, USA, August, 2001.

[65] N. R. Jennings, K. Sycara and M. Wooldridge, A Roadmap of Agent Research and Development, *Autonomous Agents and Multi-Agent Systems Journal, N.R. Jennings, K. Sycara and M. Georgeff (Eds.), Kluwer Academic Publishers, Boston.*, **1**(1), (1998), pp. 7-38.

[66] L. P. Kaelbling, Rex: A symbolic language for the design and parallel implementation of embedded systems, in *Proceedings of the AIAA Conference on Computers in Aeorospace VI*, Wakefield, MA, 1987, pp. 255--260.

[67] P. W. Keller, F.-O. Duguay and D. Precup, RedAgent-2003: An Autonomous Market-Based Supply-Chain Management Agent, in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, New York, NY, USA, August 19-23, 2004, **3**, pp. 1182-1189.

[68] O. Khatib, Real-time Obstacle Avoidance for Manipulators and Mobile Robots, *The International Journal of Robotics Research*, **5**(1), (1986).

[69] N. Y. Ko and R. Simmons, The lane-curvature method for local obstacle avoidance, in *Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 1998.

[70] S. Koolmanojwong, R. Jiamthapthaksin and J. Daengdej, An agent architecture for competitive application environment, in *Proceedings of the 2004 IEEE Aerospace Conference*, Big Sky, Montana, USA, March 6-13, 2004 **5**, pp. 3079-3089.

[71] Y. Koren and J. Borenstein, Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation, in *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacramento, California, April 7-12, 1991, pp. 1398-1404.

[72] K. Kostiadis and H. Hu, Reinforcement Learning and Co-operation in a Simulated Multi-agent System, in *Proceedings of IEEE/RSJ International Conference on Intelligent Roots and Systems*, Kyungiu, Korea, October 17-21, 1999, pp. 990-995.

[73] K. Kostiadis and H. Hu, *A Multi-threaded Approach to Simulated Soccer Agents for the RoboCup Competition*, *RoboCup-99: Robot Soccer World Cup III*, Springer-Verlag, London, UK, 2000, pp. 366-377.

[74] F. Lamarche and S. Donikian, The orchestration of behaviours using resources and priority levels, in *Proceedings of the Eurographics Workshop*, Manchester, U.K., September 2-3, 2001, pp. 171-182.

[75] F. Lamarche and S. Donikian, Automatic orchestration of behaviours through the management of resources and priority levels, in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, Bologna, Italy, July 15-19, 2002, pp. 1309-1316.

[76] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.

[77] J. Lee, M. J. Huber, E. H. Durfee and P. G. Kenny, UM-PRS: an implementation of the procedural reasoning system for multirobot applications, in *Proceedings of AIAA/NASA Conference on Int. Robots in Field, Factory, Service and Space*, American Institute of Aeronautics and Astronautics, 1994.

[78] M. Leyden, D. Toal and C. Flanagan, *A Fuzzy Logic Based Navigation System for a Mobile Robot*, Automatisierungssymposium, Wismar, 1999.

[79] Z. N. Lin, H. J. Hsu and F. J. Wang, Intention Scheduling for BDI Agent Systems, in *Proceedings of International Conference on Information Technology Coding and Computing (ITCC2005)*, 2005, **2**.

[80] J. Liu, *Autonomous agents and multi-agent systems Explorations in learning, self-organization, and adaptative computation*, World Scientific, Singapore, 2001.

[81] P. Lokuge and D. Alahakoon, A Motivation Based Behavior in Hybrid Intelligent Agents for Intention Reconsideration Process in Vessel Berthing Applications, in *Proceedings of the 4th International Conference on Hybrid Intelligent Systems*, Kitakyushu, Japan, December 5-8, 2004, pp. 124-129.

[82] K. H. Low, W. K. Leow and J. M. H. Ang, A hybrid mobile robot architecture with integrated planning and control, in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, Bologna, Italy, July 15-19, 2002, pp. 219-226.

[83] R. Machado and R. H. Bordini, *Running AgentSpeak(L) Agents on SIM_AGENT*, *Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, Springer-Verlag, London, UK LNCS, 2001, pp. 158-174.

[84] P. Maes and R. A. Brooks, Learning to Coordinate Behaviors, in *Proceedings of the American Association of Artificial Intelligence*, Boston, MA, August, 1990, pp. 796-802.

[85] S. Mahadevan and J. Connell, Automatic programming of behavior-based robots using reinforcement learning, *Artificial Intelligence*, **55**, (1992), pp. 311-365.

[86] S. Marsella and J. Gratch, A step toward irrationality: using emotion to change belief, in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, Bologna, Italy, July 15-19, 2002, pp. 334-341.

[87] S. Marsella, D. V. Pynadath and S. J. Read, PsychSim: Agent-based modeling of social interactions and influence, in *Proceedings of the International Conference on Cognitive Modeling*, Pittsburgh, Pennsylvania, USA, July 30-August 1, 2004, pp. 243--248.

[88] T. M. Mitchell, *Machine Learning*, McGraw Hill, 1997.

[89] H. P. Moravec and A. Elfes, High resolution maps from wide angle sonar, in *Proceedings of IEEE Int. Conf. On Robotics and Automation*, March, 1985, pp. 116-121.

[90] K. L. Myers, CPEF: A continuous planning and execution framework, *AI Magazine*, **20**, (1999), pp. 63-70.

[91] H. Nakashima and I. Noda, Dynamic Subsumption Architecture for Programming Intelligent Agents, in *Proceedings of the 3rd International Conference on Multi Agent Systems*, July 3-7, 1998, pp. 190.

[92] M. Namee and B. P. Cunningham, A Proposal for an Agent Architecture for Proactive Persistent Non Player Characters, in *Proceedings of the 12 th Irish Conference on AI and Cognitive Science*, 2001, pp. 221 -- 232.

[93] E. Norling, L. Sonenberg and R. Rnnquist, Enhancing Multi-Agent Based Simulation with Human-Like Decision Making Strategies, in *Proceedings of Multi-Agent Based Simulation: Proceedings of the Second International Workshop, MABS*, 2000.

[94] E. Norling, Learning to notice: Adaptive models of human operators, in *Proceedings of the Second International Workshop on Learning Agents*, Montreal, Canada, May, 2001.

[95] E. Norling and F. E. Ritter, Embodying the JACK Agent Architecture, in *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, 2001, pp. 368-377.

[96] E. Norling, Capturing the Quake Player: Using a BDI Agent to Model Human Behaviour, in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, Melbourne, Australia, July, 2003.

[97] E. Norling, Folk Psychology for Human Modelling: Extending the BDI Paradigm, in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, New York, NY, July, 2004.

[98] E. Norling and F. E. Ritter, Towards Supporting Psychologically Plausible Variability in Agent-Based Human Modelling, in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, New York, NY, July, 2004.

[99] E. Norling and L. Sonenberg, Creating Interactive Characters with BDI Agents, in *Proceedings of Australian Workshop on Interactive Entertainment Sydney*, Australia, February 2004.

[100] H. S. Nwana, *Software agents: an overview*, *The Knowledge Engineering Review*, 1996.

[101] D. d. Oliveira, A. L. C. Bazzan and V. Lesser, Using Cooperative Mediation to Coordinate Traffic Lights: a Case Study, in *Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, Utrecht, The Netherlands, July 25-29, 2005, pp. 563-470.

[102] E. Oliveira and L. Sarmento, Emotional advantage for adaptability and autonomy, in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems (AAMAS 2003)*, New York, USA, July 14-18, 2003, pp. 305-312.

[103] P. Paruchuri, A. R. Pullalarevu and K. Karlapalem, Multi agent simulation of unorganized traffic, in *Proceedings of the First International Joint Conference on*

Autonomous Agents and Multiagent Systems (AAMAS 2002), Bologna, Italy, July 15-19, 2002, pp. 176-183.

[104] P. R. Peter Stone, and Manuela Veloso, *The CMUnited-99 champion simulator team*, *RoboCup-99: Robot Soccer World Cup III*, Springer Verlag, Berlin, 2000, pp. 35-48.

[105] A. Pokahr, L. Braubach and W. Lamersdorf, A Goal Deliberation Strategy for BDI Agent Systems, in *Proceedings of Third German conference on Multi-Agent System TEchnologieS (MATES-2005)*, Koblenz, Germany, September 11-13, 2005, pp. 82-94.

[106] A. Pokahr, L. Braubach and W. Lamersdorf, A Flexible BDI Architecture Supporting Extensibility, in *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2005)*, Compiègne University of Technology, France, September 19-22, 2005, pp. 379-385.

[107] D. Poole, A. Mackworth and R. Goebel, *Computational Intelligence - A Logical Approach*, Oxford University Press, New York, 1998.

[108] C. Ramos, An Architecture and a Negotiation Protocol for the dynamic Scheduling of Manufacturing Systems, in *Proceedings of the IEEE International Conference on Robots & Automation*, 1994, pp. 3161 – 3166.

[109] O. Rana, M. Winikoff, L. Padgham and J. Harland, Applying Conflict Management Strategies in BDI Agents for Resource Management in Computational Grids, in *Proceedings of Computer Science 2002, Twenty-Fifth Australasian Computer Science Conference (ACSC2002)*, Monash University, Melbourne, Victoria, Australia, January/February, 2002.

[110] A. S. Rao and M. Georgeff, Modeling rational agents within a BDI architecture, in *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, Cambridge, MA, USA, April 22-25, 1991, pp. 473-484.

[111] A. S. Rao and M. Georgeff, BDI agents: From theory to practice, in *Proceedings of the First International Conference on Multiagent Systems*, San Francisco June 12-14, 1995.

[112] A. S. Rao, AgentSpeak(L): BDI agents speak out in a logical computable language, in *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away*, Einhoven, The Netherlands, 1996, pp. 42-55.

[113] J. A. Rice, *Mathematical Statistics and Data Analysis*, Duxbury Press, 1999.

[114] M. Rigolli and M. Brady, Towards a behavioural traffic monitoring system, in *Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, Utrecht, The Netherlands, July 25-29, 2005, pp. 449-454.

[115] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.

[116] L. B. Said, T. Bouron and A. Drogoul, Agent-based interaction analysis of consumer behavior, in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, Bologna, Italy, July 15-19, 2002, pp. 184-190.

[117] K. Selvarajah and D. Richards, The use of emotions to create believable agents in a virtual environment, in *Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, Utrecht, The Netherlands, July 25-29, 2005.

[118] M. Si, S. C. Marsella and D. V. Pynadath, Thespian: using multi-agent fitting to craft interactive drama, in *Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, Utrecht, The Netherlands, July 25-29, 2005, pp. 21-28.

[119] R. Simmons, The Curvature-Velocity Method for Local Obstacle Avoidance, in *Proceedings of International Conference on Robotics and Automation*, April 1996, 1996.

[120] A. Sloman, *What sort of control system is able to have a personality?*, in R. Trappl and P. Petta, eds., *Creating Personalities for Synthetic Actors: Towards Autonomous Personality Agents*, Springer, 1996, pp. 16-208.

[121] A. Sloman, *What sort of architecture is required for a human-like agent?*, in M. Wooldridge and A. Rao, eds., *Foundations of Rational Agency*, Kluwer Academic Publishers, 1997.

[122] L. D. S. P. Smith, Emotionware, *Crossroads, Special issue on artificial intelligence*, **3**(1), (1996), pp. 5-6.

[123] C. Stachniss and W. Burgard, An Integrated Approach to Goal-directed Obstacle Avoidance under Dynamic Constraints for Dynamic Environments, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[124] R. Sun and C. Sessions, Multi-agent reinforcement learning with bidding for segmenting action sequences, in *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB'2000)*, Paris, France, 2000.

[125] R. S. Sutton, Reinforcement learning architectures, in *Proceedings of ISKIT'92 International Symposium on Neural Information Processing*, Fukuoka, Japan, 1992.

[126] K. Sycara, Intelligent Agents and the Information Revolution, in *Proceedings of the UNICOM Seminar on Intelligent Agents and their Business Applications*, London, November 8-9, 1995, pp. 143 -159.

[127] J. Thangarajah, L. Padgham and M. Winikoff, Detecting & exploiting positive goal interaction in intelligent agents, in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, Melbourne, Victoria, Australia, July 14-18, 2003, pp. 401-408.

[128] T. Thurston and H. Hu, Distributed Agent Architecture for Port Automation, in *Proceedings of the 26th Annual Int. Computer Software and Applications Conference*, Oxford, England, August 26-29, 2002.

[129] I. Ulrich and J. Borenstein, VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots, in *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, Leuven, Belgium, May 16-21, 1998, pp. 1572-1577.

[130] I. Ulrich and J. Borenstein, VFH*: Local Obstacle Avoidance with Look-ahead Verification, in *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 24-28, 2000, pp. 2505-2511.

[131] M. Verleysen and K. Hlavackova, An Optimized RBF Network for Approximation of Functions, in *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'94)*, Brussels, Belgium, April 20-22, 1994, pp. 175-180.

[132] D. L. Waltz, Eight principles for building an intelligent robot, in *Proceedings of the First International Conference on Simulation of Adaptive Behaviour: From Animals to Animats*, Paris, September 24-28, 1991.

[133] C. J. C. H. Watkins and P. Dayan, Q-learning, *Machine Learning*, **8**, (1992), pp. 279-292.

[134] L. K. Wickramasinghe and L. D. Alahakoon, A Novel Adaptive Decision Making Agent Architecture Inspired by Human Behavior and Brain Study Models, in *Proceedings of 4th International Conference on Hybrid Intelligent Systems (HIS 2004)*, Kitakyushu, Japan, December 5-8, 2004, pp. 142-147.

[135] M. Wooldridge and N. R. Jennings, *Agent Theories, Architectures and Languages: A Survey*, in M. Wooldridge and N. R. Jennings, eds., *Intelligent Agents - Theories, Architectures, and Languages I*, Springer-Verlag, 1995.

[136] M. Wooldridge, *Reasoning about rational agents*, The M. I. T. Presss, Cambridge, MA, 2000.

[137] M. Wooldridge, *An Introduction to Multiagent Systems*, John Wiley & Sons (Chichester, England), 2002.

[138] P. Xuan and V. Lesser, Multi-agent policies: from centralized ones to decentralized ones, in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, Bologna, Italy, July 15-19, 2002, pp. 1098-1105.

[139] T. Yamashita, K. Izumi, K. Kurumatani and H. Nakashima, Smooth traffic flow with a cooperative car navigation system, in *Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, Utrecht, The Netherlands, July 25-29, 2005, pp. 478-485.

[140] H. Zhang and S. Y. Huang, Dynamic Map for Obstacle Avoidance, in *Proceedings of the IEEE 6th International Conference On Intelligent Transportation Systems*, Shanghai, China, October 12-15, 2003, pp. 1152- 1157.

[141] H. Zhang and S. Y. Huang, BDIE architecture for rational agents, in *Proceedings of the international conference for Integration of Knowledge Intensive Multi-Agent Systems (KIMAS05)*, Waltham, Massachusetts, US, April 18-21, 2005, pp. 623-628.

Appendix

A. Complete behaviour records of the vessel captain agent

ID	Insert_Time	Type	Behaviour
492	4/8/2005 7:32:20 PM	Agent1	Start!
493	4/8/2005 7:32:20 PM	deliberation thread0	Created. Status: New Target
494	4/8/2005 7:32:20 PM	deliberation thread1	Created. Status: Idle
495	4/8/2005 7:32:21 PM	deliberation thread0	Start. Status: New Target
496	4/8/2005 7:32:21 PM	deliberation thread0	finished. Status: New Target
497	4/8/2005 7:32:21 PM	intention thread0	Created. Intention: 0/1.
498	4/8/2005 7:32:21 PM	deliberation thread1	Start. Status: Idle
499	4/8/2005 7:32:21 PM	intention thread0	Start. Intention: 0/1.
500	4/8/2005 7:32:22 PM	intention thread0	finished. Intention: 0/1.
501	4/8/2005 7:32:21 PM	intention thread1	Created. Intention: 10.
502	4/8/2005 7:32:21 PM	deliberation thread1	finished. Status: Idle
503	4/8/2005 7:32:23 PM	deliberation thread2	Created. Status: The global path updated
504	4/8/2005 7:32:23 PM	intention thread1	Start. Intention: 10.
505	4/8/2005 7:33:06 PM	path plan	finished 2
506	4/8/2005 7:33:09 PM	path plan	finished 3
507	4/8/2005 7:33:06 PM	deliberation thread2	Start. Status: The global path updated
508	4/8/2005 7:34:38 PM	deliberation thread2	finished. Status: The global path updated
509	4/8/2005 7:34:55 PM	intention thread2	Created. Intention: 2/3.
510	4/8/2005 7:34:56 PM	suspend intention1.0	2.2 Start.
511	4/8/2005 7:34:56 PM	suspend intention1.0	2.2 Suspended.
512	4/8/2005 7:34:57 PM	intention thread2	Start. Intention: 2/3.
513	4/8/2005 7:34:57 PM	Subgoal reached	(0,0)
514	4/8/2005 7:34:57 PM	intention thread2	finished. Intention: 2/3.
515	4/8/2005 7:35:05 PM	resume intention	intention1
516	4/8/2005 7:35:05 PM	deliberation thread3	Created. Status: Running
517	4/8/2005 7:35:09 PM	deliberation thread3	Start. Status: Running
518	4/8/2005 7:35:55 PM	intention thread3	Created. Intention: 4/5.
519	4/8/2005 7:35:55 PM	suspend intention1.0	3.3 Start.
520	4/8/2005 7:35:54 PM	deliberation thread3	finished. Status: Running
521	4/8/2005 7:35:55 PM	suspend intention3.3	3.3 Suspended.
522	4/8/2005 7:35:56 PM	intention thread3	Start. Intention: 4/5.
523	4/8/2005 7:35:57 PM	Action	Created. action: Accelerate to max speed.
524	4/8/2005 7:36:07 PM	resume intention	intention1
525	4/8/2005 7:35:57 PM	intention thread3	finished. Intention: 4/5.
526	4/8/2005 7:36:37 PM	deliberation thread4	Created. Status: Running
527	4/8/2005 7:36:37 PM	deliberation thread5	Created. Status: Subgoal reached
528	4/8/2005 7:36:56 PM	deliberation thread4	Start. Status: Running

ID	Insert_Time	Type	Behaviour
529	4/8/2005 7:37:14 PM	deliberation thread4	finished. Status: Running
530	4/8/2005 7:37:16 PM	intention thread4	Created. Intention: 4/5.
531	4/8/2005 7:37:33 PM	suspend intention1.0	4.3 Start.
532	4/8/2005 7:37:33 PM	deliberation thread5	Start. Status: Subgoal reached
533	4/8/2005 7:37:38 PM	suspend intention1.0	4.3 Suspended.
534	4/8/2005 7:37:40 PM	Remove obsolete intention	1
535	4/8/2005 7:37:40 PM	intention thread5	Created. Intention: 2/3.
536	4/8/2005 7:37:40 PM	deliberation thread6	Created. Status: Targets changed
537	4/8/2005 7:37:40 PM	deliberation thread7	Created. Status: New Target
538	4/8/2005 7:37:40 PM	deliberation thread8	Created. Status: Idle
539	4/8/2005 7:37:39 PM	intention thread4	Start. Intention: 4/5.
540	4/8/2005 7:37:40 PM	Action	Created. action: Decelerate to min speed.
541	4/8/2005 7:37:40 PM	intention thread4	finished. Intention: 4/5.
542	4/8/2005 7:37:39 PM	deliberation thread5	finished. Status: Subgoal reached
543	4/8/2005 7:37:41 PM	deliberation thread6	Start. Status: Targets changed
544	4/8/2005 7:37:41 PM	deliberation thread6	finished. Status: Targets changed
545	4/8/2005 7:37:42 PM	intention thread5	Start. Intention: 2/3.
546	4/8/2005 7:37:42 PM	deliberation thread7	Start. Status: New Target
547	4/8/2005 7:37:43 PM	Subgoal reached	(45,24)
548	4/8/2005 7:37:43 PM	intention thread5	finished. Intention: 2/3.
549	4/8/2005 7:37:42 PM	intention thread6	Created. Intention: 11.
550	4/8/2005 7:37:43 PM	intention thread7	Created. Intention: 0/1.
551	4/8/2005 7:37:42 PM	deliberation thread7	finished. Status: New Target
552	4/8/2005 7:37:43 PM	deliberation thread8	Start. Status: Idle
553	4/8/2005 7:37:43 PM	deliberation thread8	finished. Status: Idle
554	4/8/2005 7:37:44 PM	intention thread8	Created. Intention: 10.
555	4/8/2005 7:37:48 PM	intention thread6	Start. Intention: 11.
556	4/8/2005 7:37:44 PM	deliberation thread9	Created. Status: Running
557	4/8/2005 7:37:50 PM	intention thread6	finished. Intention: 11.
558	4/8/2005 7:37:51 PM	deliberation thread9	Start. Status: Running
559	4/8/2005 7:37:51 PM	deliberation thread9	finished. Status: Running
560	4/8/2005 7:37:51 PM	intention thread7	Start. Intention: 0/1.
561	4/8/2005 7:38:00 PM	intention thread9	Created. Intention: 4/5.
562	4/8/2005 7:38:03 PM	deliberation thread10	Created. Status: Running
563	4/8/2005 7:38:33 PM	deliberation thread10	Start. Status: Running
564	4/8/2005 7:38:48 PM	intention thread10	Created. Intention: 4/5.
565	4/8/2005 7:38:48 PM	deliberation thread10	finished. Status: Running
566	4/8/2005 7:40:13 PM	deliberation thread11	Created. Status: The global path updated
567	4/8/2005 7:40:12 PM	intention thread7	finished. Intention: 0/1.
568	4/8/2005 7:40:16 PM	deliberation thread11	Start. Status: The global path updated
569	4/8/2005 7:40:17 PM	deliberation thread11	finished. Status: The global path updated
570	4/8/2005 7:40:13 PM	intention thread9	Start. Intention: 4/5.

ID	Insert_Time	Type	Behaviour
571	4/8/2005 7:40:17 PM	intention thread11	Created. Intention: 2/3.
572	4/8/2005 7:40:17 PM	Action	Created. action: Decelerate to min speed.
573	4/8/2005 7:40:18 PM	intention thread9	finished. Intention: 4/5.
574	4/8/2005 7:40:18 PM	deliberation thread12	Created. Status: Subgoal reached
575	4/8/2005 7:40:19 PM	intention thread10	Start. Intention: 4/5.
576	4/8/2005 7:40:19 PM	deliberation thread12	Start. Status: Subgoal reached
577	4/8/2005 7:40:19 PM	Action	Created. action: Decelerate to min speed.
578	4/8/2005 7:40:21 PM	deliberation thread12	finished. Status: Subgoal reached
579	4/8/2005 7:40:21 PM	intention thread10	finished. Intention: 4/5.
580	4/8/2005 7:40:21 PM	intention thread12	Created. Intention: 2/3.
581	4/8/2005 7:40:24 PM	intention thread11	Start. Intention: 2/3.
582	4/8/2005 7:40:26 PM	intention thread12	Start. Intention: 2/3.
583	4/8/2005 7:40:26 PM	intention thread12	finished. Intention: 2/3.
584	4/8/2005 7:40:24 PM	intention thread11	finished. Intention: 2/3.
585	4/8/2005 7:40:26 PM	deliberation thread13	Created. Status: Running
586	4/8/2005 7:40:27 PM	intention thread8	Start. Intention: 10.
587	4/8/2005 7:40:29 PM	deliberation thread13	Start. Status: Running
588	4/8/2005 7:40:30 PM	deliberation thread13	finished. Status: Running
589	4/8/2005 7:40:29 PM	path plan	finished 2
590	4/8/2005 7:40:30 PM	intention thread8	finished. Intention: 10.
591	4/8/2005 7:40:30 PM	deliberation thread14	Created. Status: Running
592	4/8/2005 7:40:30 PM	deliberation thread15	Created. Status: Subgoal reached
593	4/8/2005 7:40:30 PM	intention thread13	Created. Intention: 4/5.
594	4/8/2005 7:40:31 PM	deliberation thread14	Start. Status: Running
595	4/8/2005 7:40:32 PM	intention thread14	Created. Intention: 4/5.
596	4/8/2005 7:40:31 PM	intention thread13	Start. Intention: 4/5.
597	4/8/2005 7:40:32 PM	Action	Created. action: Decelerate to min speed.
598	4/8/2005 7:40:32 PM	intention thread13	finished. Intention: 4/5.
599	4/8/2005 7:40:31 PM	deliberation thread14	finished. Status: Running
600	4/8/2005 7:40:32 PM	deliberation thread15	Start. Status: Subgoal reached
601	4/8/2005 7:40:33 PM	deliberation thread15	finished. Status: Subgoal reached
602	4/8/2005 7:40:33 PM	intention thread14	Start. Intention: 4/5.
603	4/8/2005 7:40:33 PM	Action	Created. action: Decelerate to min speed.
604	4/8/2005 7:40:37 PM	intention thread15	Created. Intention: 2/3.
605	4/8/2005 7:40:36 PM	intention thread14	finished. Intention: 4/5.
606	4/8/2005 7:40:40 PM	intention thread15	Start. Intention: 2/3.
607	4/8/2005 7:40:40 PM	Subgoal reached	(51,21)
608	4/8/2005 7:40:40 PM	intention thread15	finished. Intention: 2/3.
609	4/8/2005 7:40:41 PM	deliberation thread16	Created. Status: Running
610	4/8/2005 7:40:41 PM	deliberation thread16	Start. Status: Running
611	4/8/2005 7:40:42 PM	deliberation thread16	finished. Status: Running
612	4/8/2005 7:40:42 PM	intention thread16	Created. Intention: 4/5.
613	4/8/2005 7:40:43 PM	intention thread16	Start. Intention: 4/5.

ID	Insert_Time	Type	Behaviour
614	4/8/2005 7:40:43 PM	Action	Created. action: Accelerate to max speed.
615	4/8/2005 7:40:44 PM	intention thread16	finished. Intention: 4/5.
616	4/8/2005 7:41:00 PM	deliberation thread17	Created. Status: Subgoal reached
617	4/8/2005 7:41:01 PM	deliberation thread18	Created. Status: Running
618	4/8/2005 7:41:01 PM	deliberation thread17	Start. Status: Subgoal reached
619	4/8/2005 7:41:01 PM	suspend deliberation17.2	18.3
620	4/8/2005 7:41:03 PM	deliberation thread18	Start. Status: Running
621	4/8/2005 7:41:03 PM	intention thread17	Created. Intention: 4/5.
622	4/8/2005 7:41:03 PM	resume deliberation	deliberation17
623	4/8/2005 7:41:03 PM	deliberation thread17	finished. Status: Subgoal reached
624	4/8/2005 7:41:05 PM	intention thread17	Start. Intention: 4/5.
625	4/8/2005 7:41:06 PM	Action	Created. action: Decelerate to min speed.
626	4/8/2005 7:41:06 PM	intention thread17	finished. Intention: 4/5.
627	4/8/2005 7:41:03 PM	deliberation thread18	finished. Status: Running
628	4/8/2005 7:41:05 PM	intention thread18	Created. Intention: 2/3.
629	4/8/2005 7:41:07 PM	intention thread18	Start. Intention: 2/3.
630	4/8/2005 7:41:07 PM	Subgoal reached	(68,46)
631	4/8/2005 7:41:07 PM	intention thread18	finished. Intention: 2/3.
632	4/8/2005 7:41:08 PM	deliberation thread19	Created. Status: Running
633	4/8/2005 7:41:09 PM	deliberation thread19	Start. Status: Running
634	4/8/2005 7:41:09 PM	deliberation thread19	finished. Status: Running
635	4/8/2005 7:41:10 PM	intention thread19	Created. Intention: 4/5.
636	4/8/2005 7:41:13 PM	intention thread19	Start. Intention: 4/5.
637	4/8/2005 7:41:13 PM	Action	Created. action: Accelerate to max speed.
638	4/8/2005 7:41:14 PM	intention thread19	finished. Intention: 4/5.
639	4/8/2005 7:41:24 PM	deliberation thread20	Created. Status: Subgoal reached
640	4/8/2005 7:41:24 PM	deliberation thread21	Created. Status: Running
641	4/8/2005 7:41:24 PM	suspend deliberation20.2	21.3
642	4/8/2005 7:41:26 PM	deliberation thread21	Start. Status: Running
643	4/8/2005 7:41:28 PM	intention thread20	Created. Intention: 4/5.
644	4/8/2005 7:41:26 PM	resume deliberation	deliberation20
645	4/8/2005 7:41:26 PM	deliberation thread21	finished. Status: Running
646	4/8/2005 7:41:26 PM	deliberation thread20	Start. Status: Subgoal reached
647	4/8/2005 7:41:29 PM	deliberation thread20	finished. Status: Subgoal reached
648	4/8/2005 7:41:30 PM	intention thread21	Created. Intention: 2/3.
649	4/8/2005 7:41:30 PM	intention thread20	Start. Intention: 4/5.
650	4/8/2005 7:41:31 PM	Action	Created. action: Decelerate to min speed.
651	4/8/2005 7:41:32 PM	intention thread20	finished. Intention: 4/5.
652	4/8/2005 7:41:33 PM	intention thread21	Start. Intention: 2/3.
653	4/8/2005 7:41:33 PM	Target1 reached	(91,50)
654	4/8/2005 7:41:33 PM	deliberation thread22	Created. Status: Running
655	4/8/2005 7:41:33 PM	intention thread21	finished. Intention: 2/3.

ID	Insert_Time	Type	Behaviour
656	4/8/2005 7:41:37 PM	deliberation thread22	Start. Status: Running
657	4/8/2005 7:41:37 PM	deliberation thread22	finished. Status: Running
658	4/8/2005 7:41:38 PM	intention thread22	Created. Intention: 4/5.
659	4/8/2005 7:41:38 PM	intention thread22	Start. Intention: 4/5.
660	4/8/2005 7:41:38 PM	Action	Created. action: Accelerate to max speed.
661	4/8/2005 7:41:38 PM	intention thread22	finished. Intention: 4/5.
662	4/8/2005 7:41:48 PM	deliberation thread23	Created. Status: Running
663	4/8/2005 7:41:48 PM	deliberation thread24	Created. Status: Subgoal reached
664	4/8/2005 7:41:49 PM	deliberation thread23	Start. Status: Running
665	4/8/2005 7:41:49 PM	deliberation thread23	finished. Status: Running
666	4/8/2005 7:41:49 PM	intention thread23	Created. Intention: 4/5.
667	4/8/2005 7:41:50 PM	deliberation thread24	Start. Status: Subgoal reached
668	4/8/2005 7:41:53 PM	intention thread23	Start. Intention: 4/5.
669	4/8/2005 7:41:50 PM	deliberation thread24	finished. Status: Subgoal reached
670	4/8/2005 7:41:54 PM	intention thread24	Created. Intention: 2/3.
671	4/8/2005 7:41:53 PM	Action	Created. action: Decelerate to min speed.
672	4/8/2005 7:41:54 PM	intention thread23	finished. Intention: 4/5.
673	4/8/2005 7:41:55 PM	intention thread24	Start. Intention: 2/3.
674	4/8/2005 7:41:55 PM	Subgoal reached	(90,28)
675	4/8/2005 7:41:55 PM	intention thread24	finished. Intention: 2/3.
676	4/8/2005 7:41:56 PM	deliberation thread25	Created. Status: Running
677	4/8/2005 7:41:57 PM	deliberation thread25	Start. Status: Running
678	4/8/2005 7:41:57 PM	deliberation thread25	finished. Status: Running
679	4/8/2005 7:42:00 PM	intention thread25	Created. Intention: 4/5.
680	4/8/2005 7:42:02 PM	intention thread25	Start. Intention: 4/5.
681	4/8/2005 7:42:02 PM	Action	Created. action: Accelerate to max speed.
682	4/8/2005 7:42:03 PM	intention thread25	finished. Intention: 4/5.
683	4/8/2005 7:42:18 PM	deliberation thread26	Created. Status: Target reached
684	4/8/2005 7:42:18 PM	deliberation thread27	Created. Status: Running
685	4/8/2005 7:42:19 PM	deliberation thread28	Created. Status: Target reached
686	4/8/2005 7:42:19 PM	deliberation thread26	Start. Status: Target reached
687	4/8/2005 7:42:19 PM	deliberation thread26	finished. Status: Target reached
688	4/8/2005 7:42:20 PM	deliberation thread29	Created. Status: Target reached
689	4/8/2005 7:42:20 PM	intention thread26	Created. Intention: 9.
690	4/8/2005 7:42:20 PM	deliberation thread28	Start. Status: Target reached
691	4/8/2005 7:42:20 PM	deliberation thread28	finished. Status: Target reached
692	4/8/2005 7:42:22 PM	deliberation thread30	Created. Status: Target reached
693	4/8/2005 7:42:23 PM	intention thread26	Start. Intention: 9.
694	4/8/2005 7:42:23 PM	deliberation thread29	Start. Status: Target reached
695	4/8/2005 7:42:24 PM	deliberation thread29	finished. Status: Target reached
696	4/8/2005 7:42:24 PM	deliberation thread31	Created. Status: Target reached
697	4/8/2005 7:42:24 PM	deliberation thread32	Created. Status: Target reached
698	4/8/2005 7:42:25 PM	Agent1	Stop begins!

ID	Insert_Time	Type	Behaviour
699	4/8/2005 7:42:24 PM	intention thread27	Created. Intention: 9.
700	4/8/2005 7:42:24 PM	deliberation thread33	Created. Status: Target reached
701	4/8/2005 7:42:25 PM	intention thread28	Created. Intention: 9.
702	4/8/2005 7:42:26 PM	Agent1	Stop ends!
703	4/8/2005 7:42:28 PM	intention thread26	finished. Intention: 9.

B. Publication list

[1] H. Zhang and S. Y. Huang, Dynamic Map for Obstacle Avoidance, in Proceedings of *the IEEE 6th International Conference On Intelligent Transportation Systems*, Shanghai, China, October 12-15, 2003, pp. 1152- 1157.

[2] H. Zhang and S. Y. Huang, BDIE architecture for rational agents, in Proceedings of *the international conference for Integration of Knowledge Intensive Multi-Agent Systems (KIMAS05)*, Waltham, Massachusetts, US, April 18-21, 2005, pp. 623-628.

[3] H. Zhang and S. Y. Huang, A Parallel BDI Agent Architecture, in Proceedings of *the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2005)*, Compiègne University of Technology, France, September 19-22, 2005, pp. 157-160.

[4] H. Zhang and S. Y. Huang, Are Parallel BDI Agents Really Better?, in Proceedings of *the 17th European Conference on Artificial Intelligence*, Riva del Garda, Italy, August 28-September 1, 2006.

[5] H. Zhang and S. Y. Huang, Dynamic Control of Intention Priorities of Human-like Agents, in Proceedings of *the 17th European Conference on Artificial Intelligence*, Riva del Garda, Italy, August 28-September 1, 2006.

[6] H. Zhang and S. Y. Huang, A General Framework for Parallel BDI Agents, in Proceedings of *the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2006)*, Hong Kong, China, December 18-22, 2006. (Nominated for journal publication)

[7] H. Zhang and S. Y. Huang, Realistic Simulation of Vessel Navigation Behaviour, in Proceedings of *2nd International Maritime-Port Technology and Development Conference (MTEC 2007)*, Singapore, September 26-28, 2007.

[8] H. Zhang and S. Y. Huang, A General Framework for Real Time Performance in BDI Agents (to be published), *Web Intelligence and Agent Systems* (2008).

[9] H. Zhang and S. Y. Huang, An Agent's Activities Are Controlled by His Priorities, The 2nd KES International Symposium on Agent and Multi-Agent Systems : Technologies and Applications (KES AMSTA 2008), Inha University, Korea, March 26-28, 2008. (Nominated for consideration for best paper)