

國立交通大學

資訊工程學系
碩士論文

BDI 代理人的意圖排程

Intention Scheduling for BDI Agent Systems



研究生：林祖年

指導教授：王豐堅 教授

中華民國九十三年八月

BDI 代理人的意圖排程

Intention Scheduling for BDI Agent Systems

研究生：林祖年

Student：Zu-Nien Lin

指導教授：王豐堅 博士

Advisor：Dr. Feng-Jian Wang

國立交通大學

資訊工程學系



A Thesis

Submitted to Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

In Partial Fulfillment of the Requirements

For the Degree of Master

In

Computer Science and Information Engineering

August 2004

HsinChu, Taiwan, Republic of China

中華民國九十三年八月

BDI 代理人的意圖排程

研究生：林祖年

指導教授：王豐堅 博士

國立交通大學

資訊工程研究所

新竹市大學路 1001 號

摘要

在各種代理人架構中，BDI 是最成熟的模型。目前，關於 BDI 代理人的研究，大多數集中於代理人動態選擇計畫的能力，即從欲望到意圖的能力。而在這些被選擇的計畫應該被執行順序上，卻很少有討論。我們稱此問題為「意圖排程」。在開發具有移動力和多重代理人的系統時，我們發現這些代理人必須能適應動態的變化。在缺乏適當的意圖排程時，代理人可能導致重複不必要的工作，浪費有價值的資源或者甚至導致使用者的計劃失敗。因此，我們要研究一個有效的意圖排程方式，並且提出 BDI 推理過程裡一些有用的架構，使 BDI 代理人更有效率並且更適用於動態的環境。

Keywords: BDI 代理人, 意圖排程

Intention Scheduling for BDI Agent Systems

Student: Zu-Nien Lin

Advisor: Dr. Feng-Jian Wang

Institute of Computer Science and Information Engineering

National Chiao Tung University

1001 Ta Hsueh Road, Hsinchu, Taiwan, ROC

Abstract

Among various agent architectures, BDI is probably the most mature model. Currently, most of the researches concerning BDI agents are focused on the ability for agents to dynamically select plans in order to achieve some goals, i.e. from Desire to Intention. However, there are few discussions on the order in which these selected plans should be executed. The problem might be called as Intention Scheduling. After developing systems with multiple agents of mobility and intelligence, we have discovered that these agents must adapt to dynamical and unpredictable changes. Without a proper scheduling, agents may result in repeating unnecessary work, wasting valuable resource or even failing the users' expect altogether. Therefore, we are going to study an effective intention-scheduling scheme and some useful structures in BDI reasoning process, to make BDI agent more efficient and suitable for dynamic environment.

Keywords: BDI Agents, Intention Scheduling

誌謝

本篇論文的完成，首先要感謝我的指導教授王豐堅博士，感謝他在技術上的指導，讓我學到了豐富的代理人技術，同時也增進了許多實務的軟體設計經驗。更感謝他的耐心鼓勵，使我瞭解做學問應有的虛心態度。另外，也非常感謝我的畢業口試評審委員留忠賢博士以及葉義雄博士，提供許多寶貴的意見，指出了我的盲點，使這篇論文能更加完善。

其次，我要感謝實驗室的伙伴們。尤其感謝博士班嘉麟學長在學問及生活上的指導，懷中學長指導我做實驗的方法，已畢業大鈞學長的鼓勵和其餘幾位學長姐平時熱心地討論。也要感謝我們這屆畢業生吉正、瓊文及大立，在學業上的互相砥礪，生活上的互相幫助，使得大家在專業技術及做人處事上均能有所成長。還有薰任學弟，幫我實做系統的許多部份，讓論文得以及時完成。

最後，我要感謝我家的人支持，讓我得以走自己想要走的路。還有我最愛的女友，元鈺，感謝她在我最徬徨、無助的時候陪在我身邊。由衷地感謝在我身邊的每一個人，謝謝。

Table of Contents

Chapter 1. Introduction	1
Chapter 2. Background	3
2.1 BDI Agent Theory.....	3
2.2 BDI Agent Architecture.....	3
2.2.1 PRS (Procedural Reasoning System).....	3
2.2.2 JAM.....	4
2.3 AgentSpeak(XL), TÆMS and DTC scheduler	6
2.4 Dynamic Discovery of Goals Interaction	7
Chapter 3. Intention Scheduling Concept	9
3.1 Factors of Intention Scheduling.....	9
3.1.1 Utility	9
3.1.2 Time Constraint.....	10
3.1.3 Interaction	11
3.1.4 Degree of completeness	12
3.1.5 Fairness	13
3.2 Intention Tree	13
3.2.1 Undecided Plan Problem.....	13
3.2.2 Structure of Intention Tree	14
3.3 The Scheduling Process	16
Chapter 4. Design and Implementation	20
4.1 JAM Script constructs for Intention Scheduling.....	20
4.1.1 goal.....	20
4.1.2 plan.....	21
4.2 Construction of the Intention Tree	23
4.2.1 static structure	23
4.2.2 Building Intention Tree	23
4.3 Information Propagation in Intention Tree	24
4.3.1 Information needed	24
4.3.2 Initialization and Updating of the Intention Tree.....	25
4.3.3 Computation Schemes of Updating	25
4.4 The Execution Cycle.....	27
4.5 Intention Selection	28
4.5.1 Computation of the Base Priority	28
4.5.2 Applying the Time Constraints	28
4.5.3 Applying the Interactions.....	28
4.5.4 Choosing the Best Intention.....	29
Chapter 5. Evaluation.....	30

5.1 Our Simulation Approach	30
5.2 Simulation Results	31
5.2.1 Considering Deadline.....	31
5.2.2 Considering Interactions	32
5.2.3 Considering both Deadline and Interactions.....	33
Chapter 6. Conclusions & Future Work.....	35
Reference	36



List of Figures

Figure 2.1 PRS-like agent architecture	4
Figure 2.2 a JAM intention structure in the middle of execution	5
Figure 2.3 an example TÆMS task structure for tracking.....	7
Figure 3.1 Shopping Agent’s intention structure in the middle of execution	14
Figure 3.2 Intention Tree of the Buying Shoes Intention	15
Figure 3.3 Shopping Agent’s intention structure in the middle of execution	16
Figure 3.4 Agent Execution Activities	17
Figure 3.5 an example of applying interactions between Intention Three.....	19
Figure 4.1 Example of Deadline Utility Function	21
Figure 4.2 The Class Diagram of Intention Tree	23
Figure 5.1 Normal versus Time-limited – SU.....	32
Figure 5.2 Normal versus Time-limited–FU, DU.....	32
Figure 5.3 Normal versus Interaction – SU	33
Figure 5.4 SU of all schemes while Deadline Density changes	34
Figure 5.5 SU of all schemes while Interaction Density changes.....	34



List of Codes

List 4.1 New Goal Format	20
List 4.2 New Plan Fields	21
List 4.3 New Plan Body Fields	22
List 4.4 A PL Example	22
List 4.5 Pseudo-Code of Sequence Scheme.....	26
List 4.6 Pseudo-Code of Branch Scheme	26
List 4.7 Pseudo-Code of Loop Scheme.....	27
List 4.8 Pseudo-Code of Calculating the Base Priority	28
List 4.9 Pseudo-Code of Applying the Time Constraints	28
List 4.10 Pseudo-Code of Applying the Interactions	29
List 4.11 Pseudo-Code of Choosing the Best Intention.....	29



Chapter 1. Introduction

Agent systems are currently one of the most active research fields in the computer science community. Researchers generally agree that for a computer program to be called a rational agent, it must show the characteristics such as autonomy, social ability, reactivity and pro-activeness.[1] Among various agent architectures, BDI (**Belief-Desire-Intention**)[2], standing for Beliefs, Desires and Intentions, where each represents a mental state in practical reasoning process, is probably the most mature model and has been adopted by many academic and industrial applications.

Currently, most of the researches concerning BDI agents are focused on the ability for agents to dynamically select plans in order to achieve some goals, i.e. from Desire to Intention. There are few discussions on the order in which these selected plans should be executed. The problem might be called as **Intention Scheduling**. After developing systems with multiple agents of mobility and intelligence [3], we have discovered that these agents must adapt to dynamical and unpredictable changes. Without a proper scheduling, agents may result in repeating unnecessary work, wasting valuable resource or even failing the users' expect altogether.

Take a shopping agent as example; the agent is designed to buy a suit at shop A, buy a pair of shoes at shop B, and do some other tasks. If these two buying goals are equally important to the user, the agent might end up going back and forth between shop A and shop B without buying anything actually. In another case, the shop A and shop B turn out to be the same shop, but the agent does not consider this point. The agent might go to shop A to buy a suit, to shop B to do some other things and back to shop A again to buy the shoes, etc. Moving twice while doing these jobs wastes the

valuable network bandwidth. Yet another situation might be that suit-buying task is more important than shoe-buying task, but the shop B will be closed within five minutes. Without awareness of this constraint, the agent cannot buy the shoes in time, i.e., fail the user's expectance.

A proper intention-scheduling scheme might greatly improve BDI agents' performance by exploiting the positive interaction between the given tasks and avoiding possible conflicts. From the example above, with a scheduling scheme to avoid conflicts, the agent will not go back and forth between two shops, but finish one task before doing another one. With the ability to exploiting the positive interaction, the agent will do the common things once and save the system's resources. With the ability to take temporal constraints into concern, the agent will not miss the deadline.

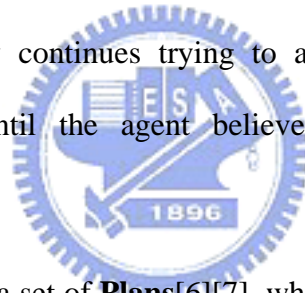
Therefore, we are going to study an effective intention-scheduling scheme and some useful structures in BDI reasoning process, to make BDI agent more efficient and suitable for dynamic environment.

The remainder of this thesis is organized as follows. Chapter 2 surveys the state of art BDI agent theories and architectures, as well as some related works on intention scheduling. Chapter 3 discusses the basic concept on intention scheduling. Chapter 4 presents the implementation aspects of our theories. Chapter 5 shows the effectiveness of our system. Chapter 6 concludes with our contribution and the future works.

Chapter 2. Background

2.1 BDI Agent Theory

The **BDI (Belief-Desire-Intention)** theory is a well-known model of rational agents, based on practical reasoning theory proposed by philosopher Michael Bratman[1]. **Belief** corresponds to the information that agent has about the world and itself. **Desire**, or **Goal**, represents the world state that the agent is trying to achieve. **Intention** is the desire that an agent has committed to achieve[4]. As Cohen and Levesque[5] said, an agent could have many desires, like human beings, but these desires may not all come true. Hence, an agent may choose some desires that seem achievable, and commit its resources to achieve them. Those chosen desires are called Intentions. An agent usually continues trying to achieve the intention, until the intention is achieved or until the agent believes that the intention becomes unachievable.



A typical BDI agent has a set of **Plans**[6][7], which defines sequences of actions to be performed to achieve a certain goal. So Intentions could also be seen as the plans an agent has chosen for eventual execution. Rao and Georgeff[8] have provided some logics for the BDI architecture. This model is believed to be effective, and has been used in a number of applications including air traffic control[9] and the handling of malfunctions on NASA's Space Shuttle[10]. Wooldridge and Jennings[11] have also done researches on intelligent agent theory.

2.2 BDI Agent Architecture

2.2.1 PRS (Procedural Reasoning System)

In this section, we are going to discuss a pioneer implementation of BDI theory, PRS[6], which is an ascendant of various BDI agent systems such as UMPRS[12],

ACT[13], etc. Figure 2.1 depicts the major parts of PRS System. There are four important components of a typical BDI agent, including Beliefs, Desires, Intentions and Plans.

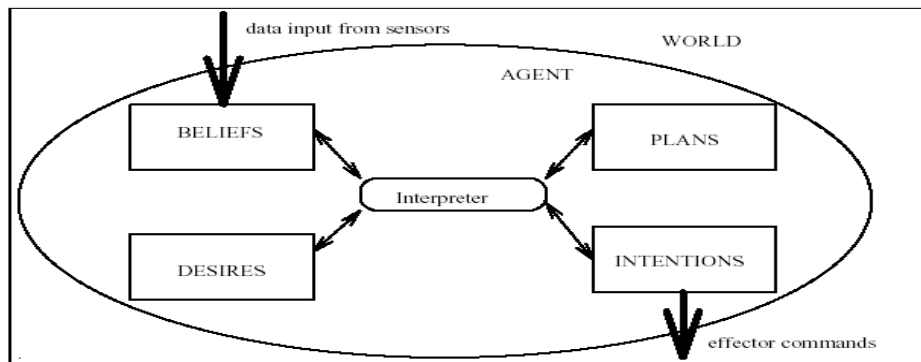


Figure 2.1 PRS-like agent architecture

Beliefs correspond to the information that an agent has about the world and itself. **Desires**, or Goals, represent the world state that the agent is trying to achieve. **Intentions** are the desires that an agent has committed to achieve. **Plans** define sequences of actions to be performed to achieve a certain goal or react to a specific situation.

The central part of this kind of agent systems is the **interpreter**. The interpreter runs in cycles. In every cycle, it first updates the Beliefs after observing the world. Next, it checks the Desires, finds those achievable ones that are not yet achieved, and chooses the most suitable plan from the Plans. Then, it associates the chosen goals (desires) and plans, and commits them into the Intentions. Finally, it executes the actions of the plans in the Intention structure. The interpreter starts the whole cycle over and over again until all the desires have been fulfilled.

2.2.2 JAM

JAM[7] is a modern BDI agent system derived from PRS. JAM supports rich

plan constructs, simple extension mechanisms, meta-level or utility-based reasoning over multiple concurrent goals, and goal-driven or data-driven behaviors. Because of these advance designs and Huber’s generosity of providing the source code of JAM for non-profit development[14], we decide to construct our system based on JAM, and further extend its capabilities.

Figure 2.2 shows an example of JAM’s Intention structure. JAM has two kinds of goals, **top-level goals** and **subgoals**. Top-level goals are persistent. That is, they are pursued until being satisfied. A JAM agent can have multiple top-level goals, and pursues these goals at the same time. Subgoals are the goals that the agent creates from the plans during plan execution. When a subgoal is issued, the current plan execution is halted, and the interpreter will try to find another plan that can achieve the subgoal. After the subgoal is achieved, the halted plan is resumed. JAM’s Intention structure is a set of intention threads, and every thread’s head is a top-level goal.

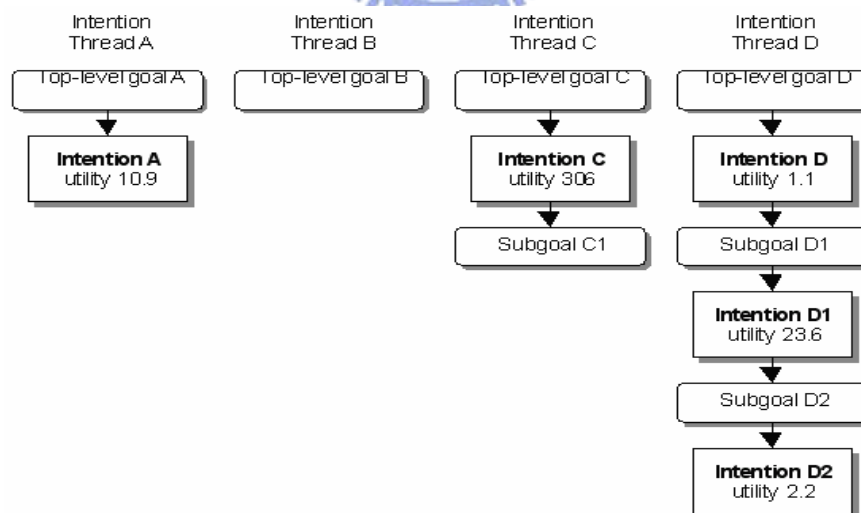


Figure 2.2 a JAM intention structure in the middle of execution

Because there may be many intentions being pursued, here comes the problem of intention scheduling. JAM uses a **Utility** function to decide which intention is more

important to the user for the execution order of intentions. However, the most important intention does not need to have the highest priority of execution. There are many things need to be considered further, which we are going to discuss in details in Chapter 3.

Note that although user can set all the considerations in utility functions and evaluated theme during runtime, doing this not only blurs the meaning of utility but also makes the agent more difficult to design.

2.3 AgentSpeak(XL), TÆMS and DTC scheduler

AgentSpeak(XL)[15] provides another kind of PRS-like BDI agents which deal with the problem of intention scheduling with **TÆMS**[16]and **DTC**[17].(see [18] for an overview of that approach to multi-agent systems.)

TÆMS (Task Analysis, Environment Modeling, and Simulation) is a task modeling framework which describes the characteristics of each task in an agent’s problem solving process, including quality, cost, time and probability. It uses a tree structure to model the relationships between tasks such as top-level task, subtask and quality accumulation function. Moreover, TÆMS also describes tasks’ temporal and resources requirements and interactions between tasks. (Tasks here can be viewed as instances of plans in PRS-like agent we’ve discussed above)

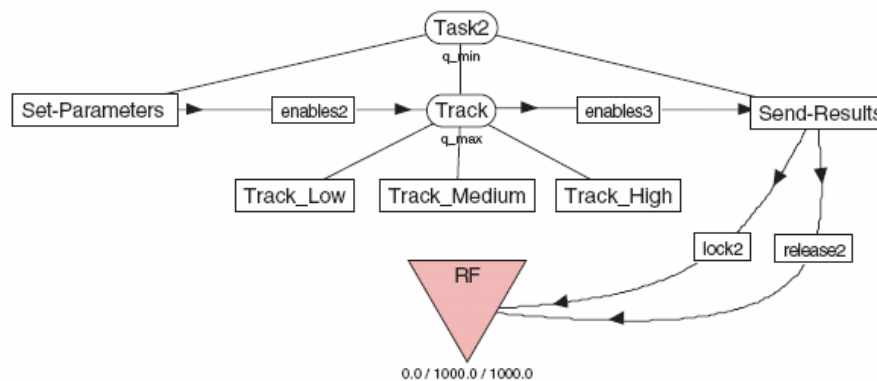


Figure 2.3 an example TÆMS task structure for tracking

DTC (Design to Criteria) scheduler uses the information provided by TÆMS to generate a proper course of actions, categorized into two parts. First are the tasks that must be performed to achieve the top-level goal. Second are the execution orders of the tasks chosen. Together, DTC and TÆMS accomplish a nice agent system that is able to deal with temporal and resource constraints.

Nevertheless, there are still some shortcomings with this kind of agents. First of all, it is not proper for these agents to be integrated into the current PRS-like agent architecture, because it duplicates the “from Desire to Intention” part of the reasoning. Second, DTC is a kind of “long term” scheduling. DTC considers too many things to do the scheduling in every execution cycles. Thus, the agent is more or less blunt to the environmental changes. Furthermore, TÆMS needs a lot of information long before the task is actually been executed. While sometimes an agent may acquire new kinds of tasks by importing new plans, AgentSpeak(XL) doesn't answer the question how the TÆMS is going to reflect these changes.

2.4 Dynamic Discovery of Goals Interaction

Padgham and Thangarajah have some great works discussed about dynamical reasoning for the goals in BDI agent, including representation and reasoning for goals[19], detecting resource conflicts[20], detecting similar goals[21], and detecting interference between goals[22]. These works utilize the information provided from “Goal-Plan-Tree” structure to detect positive or negative interactions between goals. Although the algorithms they proposed can effectively avoid conflicts or combine the similar plans, they do not take the importance of the goals and the time constrains into consideration. This may result in inefficient behaviors.

Since intentions are only committed goals in the thesis, we will use these techniques to discover interactions between intentions, and integrate them as factors in the whole scheduling process dynamically.



Chapter 3. Intention Scheduling Concept

In this chapter, we first discuss the major factors of Intention Scheduling, including utility, time constraints, interactions, degree of completeness and fairness. Then we introduce the Intention Tree, the structure to gather these factors. Finally, we propose our Intention Scheduling algorithm.

3.1 Factors of Intention Scheduling

3.1.1 Utility

Obviously, the most essential factor that affects the scheduling of intention is their importance. More important tasks should be given more time and higher priority to execute than less important ones.

JAM uses “Utility” to describe the importance of a goal and uses it as the only factor to select an intention. However, the word “Utility” in JAM has different meanings in different contexts. For goals, JAM’s Utility means how important the goal is for the user. For plans, JAM’s Utility means how good the plans will fulfill the goal. For intentions, JAM’s Utility means how urgent the intention is. As we stated in 2.2.2, this is too restrictive and confusing for the agent designer.

Here, we define **Utility** to be “the importance of the goal for the user.” It can be a real number or a simple function, whose value can be evaluated dynamically. **Applicability** is “the extents the plan can fulfill the goal.” This value represents the percentage of a goal’s Utility this plan will achieve. It can also be evaluated at runtime. **Priority** is “the urgency the intention is,” and is basically computed from the associated plan’s Applicability, goal’s Utility and some other factors we will stated later.

With these distinct definitions, agent designers do not have to think of some enigmatic functions to schedule the intentions. Instead, they can use simple values or functions to specify the importance of goals and usability of plans, and let the Agent System to compute the priority for intention automatically.

3.1.2 Time Constraint

Under a multi-agent environment, when agents cooperate with each other, they usually need to finish some tasks before deadlines. For example, a bidding-agent must place its bid before the auction-agent close the auction. To avoid missing the deadline, the task should be given a higher priority when its deadline is approaching. Therefore, the agent must know the deadline of the goal, how long the intention will cost on running, and how the priority will be affected when the deadline is approaching.

Specifying a deadline of an intention might be simplified as appending a value or function to a goal. The exact time will be computed when the goal is intended. Calculating the running time of an intention is not simple. There are two reasons at least. First, the associated plan may contain subgoals. We do not know the running time of a goal unless the plan is chosen to achieve it is selected. We called the problem “Undecided Plan Problem.” Second, beside the subgoals, each step might cost various amount of time, and the running time may depend on machine or network speed. Here, a agent designer given modifier **ERT** (Estimated Running Time) is chosen for describing how many logical time units a plan may need, excluding its subgoals execution. These time units will be transformed into the real time units during runtime with the running statistic of the agent platform. Together with the Intention Tree, ERT helps the agent to calculate the possible minimum and maximum running time of its intentions.

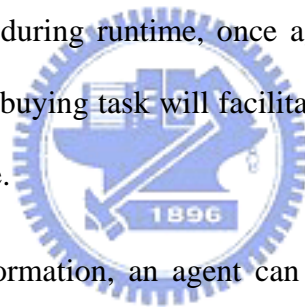
User can also specify a **DUF** (Deadline Utility Function), which formulates how

the priority of the intention will be affected when its deadline is approaching. The format of this function is defined in Chapter 4.

3.1.3 Interaction

Executing an intention may result in some effects that will change the Belief of the agent, and then affect other intentions. The relationship between two intentions that one intention helps the other is called **positive interaction**. The one that one intention hinders the other is called **negative interaction**.

Consider the shopping agent example in Chapter 1. When shops A and B are different, the suit-buying task of the agent will forbid the agent pursuing the shoe-buying task simultaneously because these two tasks must be done at different places. Another case is that, during runtime, once agent discovers that shop A and shop B are the same, the suit-buying task will facilitate the shoe-buying task because the agent need not move twice.



With the interaction information, an agent can schedule its intention to avoid conflicts or save system resources. In the above example, when the agent decides to execute the suit-buying task, it will defer the execution of the shoe-buying task because there is a negative interaction between them. When shop A and shop B are the same, the agent will try to execute these two tasks together even there is another task, say cleaning car, which is more important than the shoe-buying task.

Questions may arise regarding the importance of the tasks. For example, the shoe-buying task may be too important to be deferred; on the other hand, the car-cleaning task may be more critical so the agent should do it first. To solve this problem, the interaction cannot merely indicate the relationships of the tasks but also their magnitude of mutual influence. Obviously, if the interaction's magnitude is not

strong enough to affect the existing priorities of the intentions, the agent will take the original sequence of actions.

Here we present a tree type called Intention Tree. The interaction between the tasks can be modeled as links between Intention Trees' nodes. Each directed arc, between two nodes representing two tasks, represents the direction of the interaction. The value associated with the arc is the Utility of the affected intention.

The agent designer can specify interactions explicitly in the Plan. Still, some interactions can be automatically discovered without explicit help from the designer. For example, there should be a positive interaction between two intentions that achieving the same goal, and negative interactions between two intentions whose goals are incompatible. Besides achieving goals, plans may have side effects that may affect the Belief, and some pre-conditions or context must be kept before or during execution. These may also cause interactions between intentions. However, how to automatically discover interactions between intentions is beyond the scope of this thesis. More information on this topic can be found in the papers mentioned in 2.4.

3.1.4 Degree of completeness

When an intention is interrupted, it might lose the work it has done. For example, the cost to interrupt an intention of 80% completeness with utility of 90 is much more than the cost to interrupt an intention of 20% completeness with utility of 100. Therefore, the DoC (degree of completeness) of the intention should be considered when there are negative interactions between intentions.

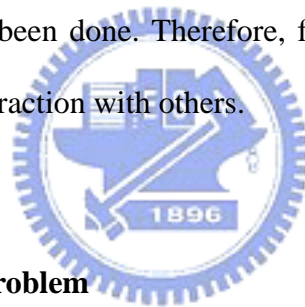
The simplest way to indicate the DoC of an intention is to count how many lines of code in a plan the intention has executed. However, each line of code in a plan has a distinct contribution. Thus, a progress label (**PL**) is provided for the designer to

show how much work has been accomplished when reaching a position of code. Note that the undecided plan problem still exists, so we need Intention Tree to compute the DoC.

3.1.5 Fairness

Besides efficiency, the fairness need be considered. The low-utility intentions may never get a chance to execute when there are high-utility ones keep executing. In order to avoid this “starvation” problem, the priority of an intention that has not executed for a long while should be raised.

However, if the low-utility intention has some negative interactions with other intentions, raising the priority of this intention might cause the other intentions to fail and waste the work that has been done. Therefore, fairness will only be considered when the intention has no interaction with others.



3.2 Intention Tree

3.2.1 Undecided Plan Problem

Intention Trees are mainly used to solve the “Undecided Plan Problem.” The problem results from the request of information providing by some execution, which does not occurs yet. Here we use an example to show the “Undecided Plan Problem.”

Figure 3.1 shows a JAM shopping agent’s intention structure during execution. The dotted-line box in Intention Thread B represents the activities not taken yet. When the agent is executing the “Moving to Site A” activities in intention-thread A, it doesn’t know that intention-thread B might have a “Moving to Site B” subgoal or not, because the Buying-shoes goal has not chosen which plan to execute yet. The Interaction cannot be discovered until the agent has decided to use “Buying Shoes’ Plan 1.” Currently, the “Moving to Site A” part of actions might be finished long

before. The situation gets nastier when the “Moving to Site B” is deep down in intention-thread B.

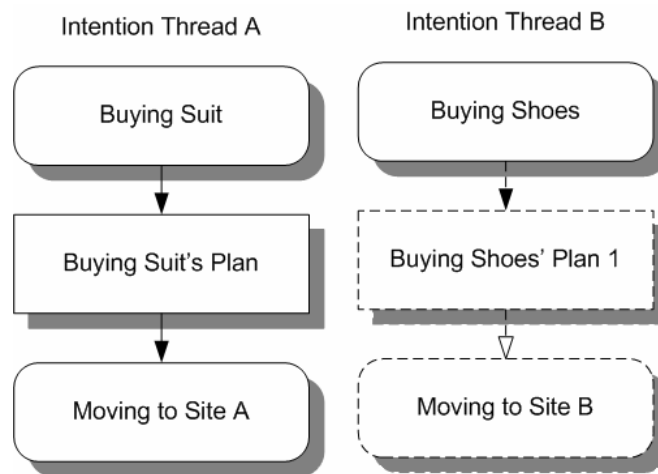


Figure 3.1 Shopping Agent’s intention structure in the middle of execution

“Undecided Plan Problem” not only obstructs the discovery of the Interaction, but also hinders other calculation such as the running time and the DoC of a plan. Choosing every plan for every goal before the execution really starts cannot solve this problem. The approach results in non-dynamic behavior because the environment change will not reflect to the plan chosen process. Our approach extends the concept of Intention Thread as Intention Tree to solve the “Undecided Plan Problem.”

3.2.2 Structure of Intention Tree

In the thesis, the **Intention Tree** is applied to model the possible behaviors of an agent. Its nodes record not only static structure of the agent but also the runtime information.

An Intention Tree mainly comprises two kinds of nodes, **goal-node** and **plan-node**. A goal-node contains the information about a goal. The information includes the goal’s type, name, utility, deadline, and etc. Plan-node contains the information about a plan. The information includes the plan’s body, applicability, estimated running time, and degree of completeness. The children of the goal-node

are those plans can achieve the goal, and the children of the plan-node are its subgoals. Figure 3.2 shows the Intention Tree of the Buying Shoes Intention (Intention Thread B in previous example.)

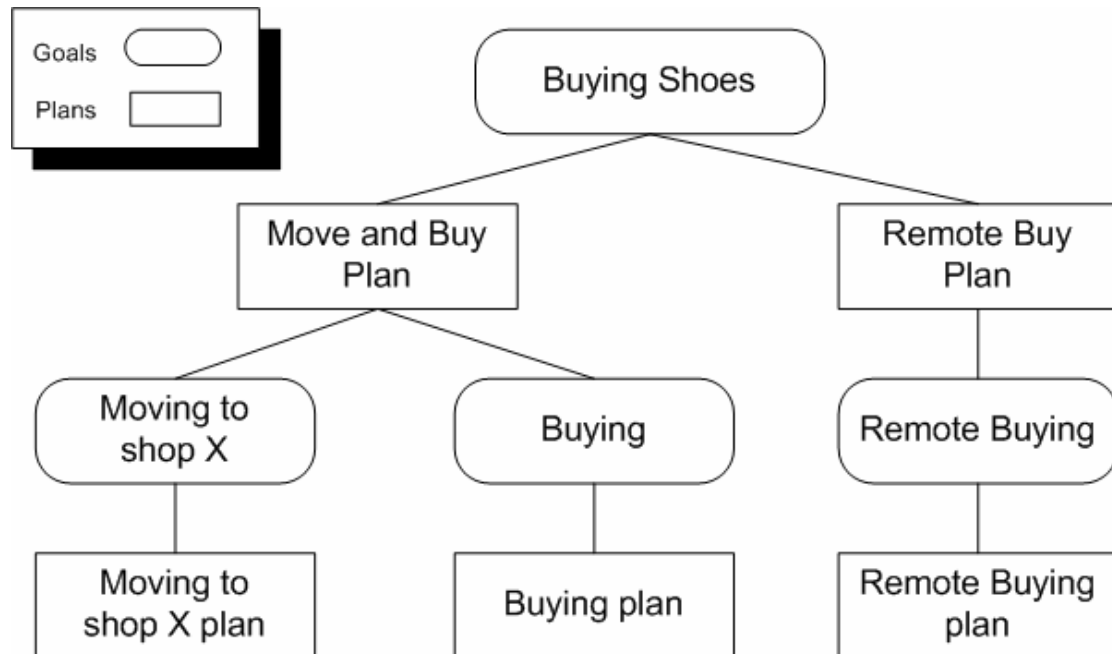


Figure 3.2 Intention Tree of the Buying Shoes Intention

The information recorded on each node is applied for recursive computation till the root of each intention tree, and kept update during runtime when actions are performed, goals are achieved, plans are failed, or new plans are added. Thus, we can “foresee” what the agent might do without the actual execution.

In figure 3.3, although the “Buying shoes” goal has not yet chosen which plan to execute, the agent still knows that there is a “Moving to Site B” subgoal in intention-tree B. Consequently, the agent can decide whether there is interaction or not. Note that the Intention structure in figure 3.3 dose not contain Intention Thread but the Intention Tree instead.

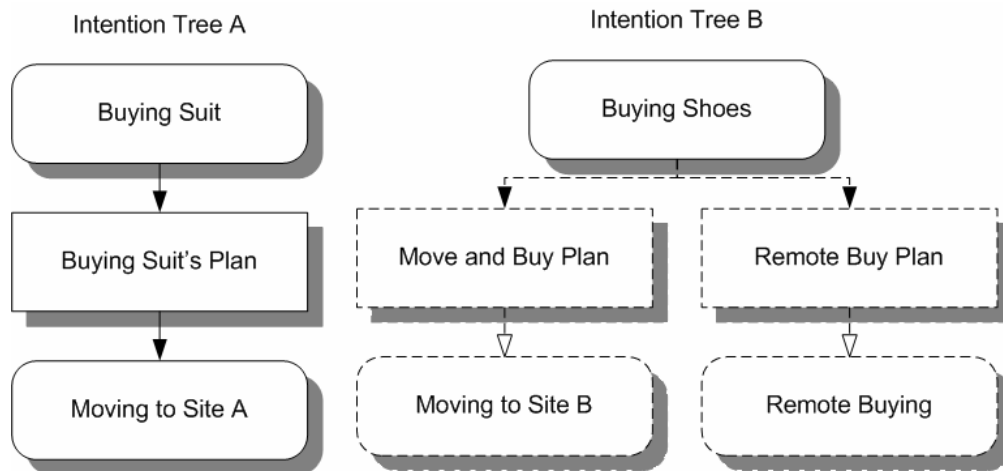
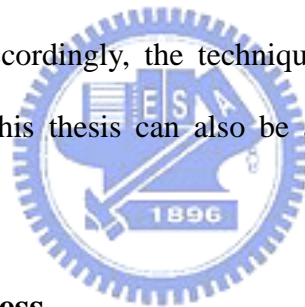


Figure 3.3 Shopping Agent's intention structure in the middle of execution

The ability to “foresee” what the agent might do could also benefit BDI's choosing plan process. For example, the agent could choose the plan that will not conflict with the already executing intention, or avoid the plan that definitely cannot meet the goal's deadline. Accordingly, the techniques to construct and update the Intention Tree described in this thesis can also be applied in other aspect of BDI agents.



3.3 The Scheduling Process

This section presents our scheduling process. Figure 3.3 shows the BDI agent's execution activities. The bolded rectangles are those new intention-scheduling related activities.

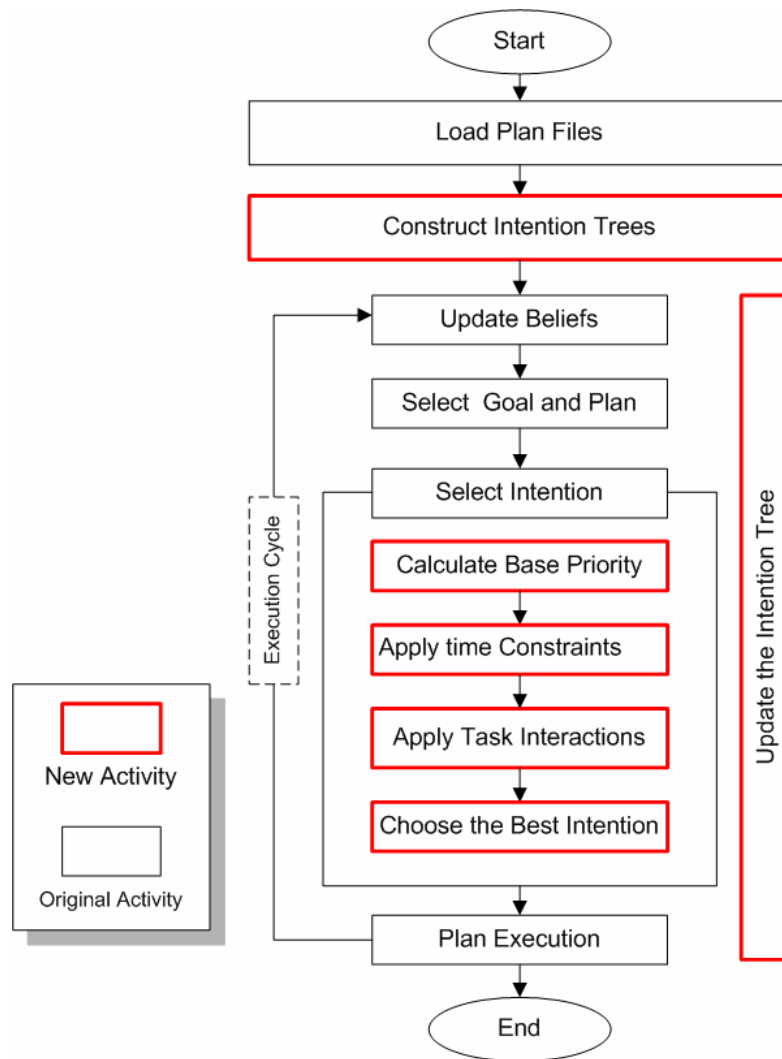


Figure 3.4 Agent Execution Activities

After loading the plans, an agent constructs its Intention Tree. The block “Update the Intention Tree” beside the “Execution Cycle” is not a separated execution thread. In the figure, the “Update the Intention Tree” box is added to indicate the Intention Tree is always kept updated and can be used in process other than Intention Selection.

The “Select Intention” is the process that chooses the intention to execute next. It contains four major steps.

First, the agent calculates each Intention Tree’s **Base Priority**. The value is defined as Utility of the top-level goal times Applicability of the top-level plan. (The top-level plan is the current plan applied to achieve the top-level goal.) Here, only the

top-level goal is considered because its importance dose not change whether a subgoal is added or deleted. For example, the importances of studying for midterm dose not decrease because its subgoal, say, flipping the page of textbook, is not so important. Subgoals' Utility and their plans' Applicability are considered when applying deadline and interaction.

Second, the agent checks if the intention is approaching its deadlines. An Intention Tree may have more than one deadline because its subgoal may have deadlines too. The agent adjusts the priorities of the intention tree according to each deadline's DUF.

Third, the agent takes the interaction between intentions into account. This step works with a help of graph. Each node in this graph is an Intention Tree of the agent. There are three kinds of links, **Base Links**, **Interactions Links** and **Final Links**. The links with value N from nodes A to nodes B means A has higher priority than B by the magnitude of N . A Base Link is associated with a value, which is the difference of the Base Priorities of both ends. Its direction is from the node of higher base priority to another one. **Interactions Links** represents the interactions between the intention trees, either specified by the user or automatically discovered. If there is no interaction, the agent adjusts the base links of the node according to the time it waits for execution. The agent consolidates these links between the nodes to produce the **Final Links**.

Finally, the agent chooses the node of with input Final Links. An Intention tree with no input Final Link has higher priority than others. When the tree does not exist, the agent recursively deletes the Final Link with the lowest value until there is a node with no input Final Links.

Figure 3.5 shows an example of activities 3 and 4. Although the Intention Tree C has the lowest base priority among all three, its strong interaction to Intention Tree A makes a Final Link from C to A. Because each node has at least one input Final Link, the agent deletes the lowest-value Final Link, which is the link between C and B. After that, there is no Final Links pointing to C. C becomes the next intention for execution then.

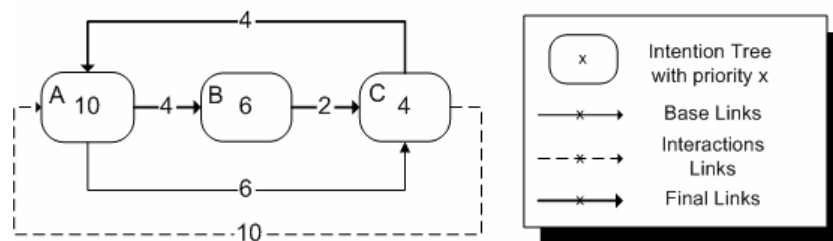


Figure 3.5 an example of applying interactions between Intention Three



Chapter 4. Design and Implementation

4.1 JAM Script constructs for Intention Scheduling

In this section we are going to explain those new constructs added for intention scheduling. The original Jam script specification can be found at[7].

4.1.1 goal

We modified JAM's goal format to add functionality such as deadline and timeout.

```
goal_type goal_name parameter1
(:UTILITY expression)?
(:DEADLINE expression
(:DEADLINE_UTIL_FUNC expression, expression)?
|:TIMEOUT)?;
```

List 4.1 New Goal Format

The **DEADLINE** field indicates that this goal must be finish before the time specified with the expression. Optional **DEADLINE_UTIL_FUNC** is the function, computing how much the time factor affects the intention's priority. A special variable **\$delta** can be used in the function to represent the difference between the current time and the deadline.

The field **DEADLINE_UTIL_FUNC** has two functions. Figure 4.1 illustrates an example. The **maxERT** and **minERT** showing in the figure indicate the time needed to finish the intention. The first function is applied when **delta** is beyond the **maxERT**. It should be set as a reverse function of **\$delta**, because the priority should rise when the time is approaching the deadline. The second function is applied when the deadline is between the **maxERT** and **minERT**. In this case, the agent cannot guarantee there is enough time to finish the job, so the user might define how the priority should

vary.

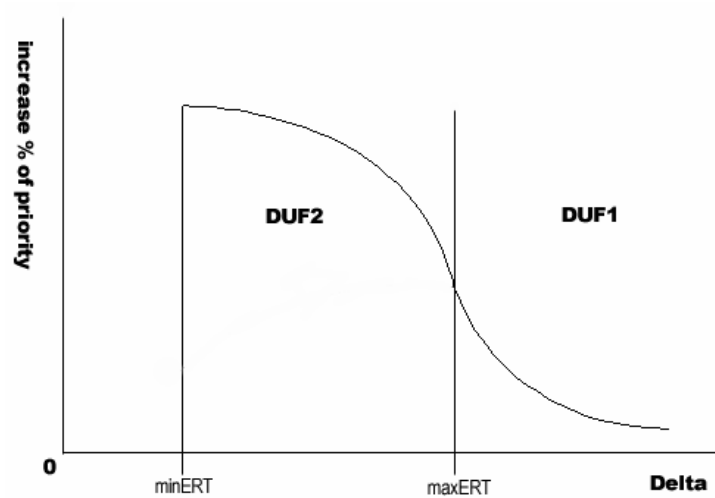


Figure 4.1 Example of Deadline Utility Function

When the `delta` is less than `minERT`, the Intention is timeout and considered fail. For those intentions waiting too long and having other applicable plans, they should abandon current suspended plan and choose another. Therefore, designer can append a **TIMEOUT** to the goal (Note that the second kind of timeout is actually an “Intention Reconsideration” problem, so we will not discuss it in details.)

4.1.2 plan

We add these new fields in to JAM’s plan:

```
APPLICABILITY: numeric expression
ERT: numeric expression
FAIRFACTOR: numeric expression
INTERACTION: (goal_spec|plan_name), [numeric expression]
```

List 4.2 New Plan Fields

The **APPLICABILITY** is how the plan can fulfill the goal. It is specified with a value ranging from 0.0 to 1.0. The original `utility` field in JAM no longer exists. The **ERT** field represents how many time units a plan may cost, excluding its subgoals execution time. Because of the “Undecided Plan Problem,” the subgoals execution

time should be gotten from the Intention Tree. The **FAIRFACTOR** field specifies the factor using in fairness calculation. The **INTERACTION** field describes the user-defined interaction between the plan and a kind of goals or plans. The magnitude of the interaction is given as the numeric expression.

A PL (progress label) and ELT (estimated loop time) are added to the plan body:

```
PL numeric expression:  
ELT numeric expression:
```

List 4.3 New Plan Body Fields

PL (Progress Label) shows how much work has been accomplished when reaching a position of the code. The field facilitates the calculation of DoC (Degree of Completeness) and the running time of the plan.

In the example shown in List4.4, the PL indicates that the plan will have 80% completeness when Action2 is finished. Also, together with ERT, it implies that Action1 and Action2 might take 40 time units and Action3 might take 20 time units to execute.

```
Plan: {  
    ERT: 100s  
    Body:  
        EXECUTE Action1;  
        EXECUTE Action2;  
    PL 80:  
        EXECUTE Action3;  
}
```

List 4.4 A PL Example

ELT are used when there are loop construct or possible recursive subgoaling, where a plan for a particular goal can subgoal to the same goal. It gives an estimate value of how many times the loop or recursive block are going to repeat, and therefore

the agent can calculate the running time and DoC accordingly.

4.2 Construction of the Intention Tree

4.2.1 static structure

Figure 4.2 shows the Class Diagram of Intention Tree. The agent's Interpreter access Intention Tree by its root, which is the `GoalNode` representing the top-level goal. `GoalNodes` have several `PlanNodes` as children. The `PlanNode` is a tree, which consist of one or more plan constructs. Various actions, including subgoaling actions, are done is these plan construct.

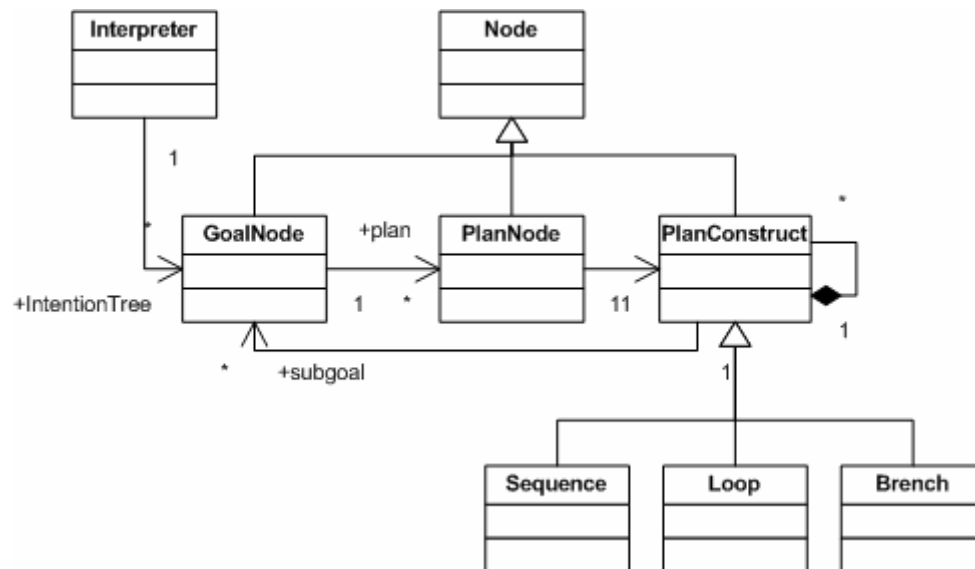


Figure 4.2 The Class Diagram of Intention Tree

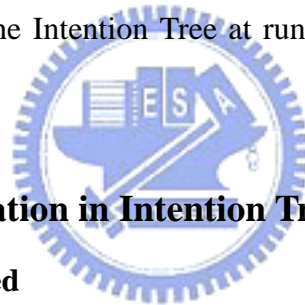
4.2.2 Building Intention Tree

Whenever a top-level goal is issued, the corresponding Intention Tree must be built. Interpreter will generate a new `GoalNode` to represent the top-level goal, and put it into its Intention Structure. For each plan that can fulfill the goal, a `PlanNode` is generated and appended under the `GoalNode`. The `PlanNode` consists of a tree of `PlanConstructs`, which are built according to the plan's body. The `PlanConstruct` with subgoaling action will generate `GoalNodes` and append them as its children. The

construction continues recursively until there is no subgoaling anymore.

The subgoaling can be recursive. The construction of the intention tree with recursive subgoaling will continue indefinitely. Therefore, when the interpreter detects recursive subgoaling, it marks the repeat part as **recursive block**, and generates this part only once.

Building the Intention Tree at runtime is a time consuming process. We can shorten the agent start-up time by compiling the agent before actual execution. The compilation produces an Intention Tree Template for every possible top-level goal, which can be used to generate Intention Tree rapidly in execution. For agents that can dynamically import plan, which may contain new top-level goals without precompiled template, we still can build the Intention Tree at runtime and cache it for the future use.



4.3 Information Propagation in Intention Tree

4.3.1 Information needed

The information needed for computation in Intention Tree are listed below:

1. Interactions Info: Both the user specified interactions and the information needed to discover the interactions automatically, such as possible effects, guarding conditions, etc.
2. Deadlines Info: Deadlines, DUF (Deadline Utility Functions), and Timeout
3. Runtime Info: maximum ERT, minimum ERT, DoC (Degree of Completeness), last execution time, and the **runtime state** of each node.

The runtime state of the node in the Intention Tree shows the current status of the agent. It can be the following values:

UNTRIED: the node is not yet intended

ACTION: the node is executing

SUCCESS: the node is completed

FAILURE: the node is failed

When propagating in the tree, Interactions Info and Deadlines Info are stored in lists, which contain links pointing back to the node that the interaction or the deadline is applied on. Thus, other information that might be used in priority calculation such as Utility or Applicability of the node can be easy access.

4.3.2 Initialization and Updating of the Intention Tree

After the Intention Tree is constructed, it will go through an initialization process, which is done by updating each leaf node of the tree, to gather the information needed for intention scheduling. The gathered information will be stored in the tree's root for future access. Note that the information is not stored in its final value, because the environment may change between the time the information is gathered and the time it is needed. For example, the **maxERT** will still in time units formats when stored in the root, and will be transformed to real value (milliseconds) when the agent applies it to the Deadline Utility Functions.

During the agent's execution, the Information in the Intention Tree must be updated whenever actions are performed, goals are achieved, plans are failed, or new plans are added.

4.3.3 Computation Schemes of Updating

Three kinds of computation schemes are used to update the Intention Tree. They are **Sequence**, **Branch** and **Loop**, each for different kinds of node.

Sequence is the scheme for the sequence plan construct. This kind of node will

succeed when all its children succeed but fail whenever any of its children fail.

```
if(every child node is in success state) state := success;
if(any child node is in failure state) state := failure;
(maxERT, minERT, interactions, deadlines):=
    sum of the (maxERT, minERT, interactions, deadlines)
    of every child in untried or action state;
DoC := the sum of (DoC of every child × the PL of child);
```

List 4.5 Pseudo-Code of Sequence Scheme

Branch is the scheme for the branch plan construct and the goal node. This kind of node will succeed whenever any of its children succeed but fail when all of its children fail.

```
if(any child node is in success state) state := success;
if(every child node is in failure state) state := failure;
maxERT := the highest maxERT of every child;
minERT := the lowest minERT of every child;
if(no child is in action state) //branch undecided
    (interactions, deadlines):= sum of (interactions, deadlines)
    of every child in untried or action state;
    DoC := 0;
else
    (interactions, deadlines):= (interactions, deadlines) of the
    child in action state;
    DoC := DoC of the child in action state;
```

List 4.6 Pseudo-Code of Branch Scheme

Loop is the scheme for the loop plan construct and the recursive block. These nodes have only one child, which will repeat the execution until it meets the boundary condition.

```
state := child's state
maxERT := maxERT of the child × estimated times of execution;
minERT := minERT of child;
(interactions, deadlines):=
```

```
(interactions, deadlines) of the child;  
DoC :=  
times of execution / estimated times of execution  
+ DoC of the child;
```

List 4.7 Pseudo-Code of Loop Scheme

4.4 The Execution Cycle

This section describes how the Intention Tree updating process fits into the execution cycle.

In every cycle, the interpreter first updates the Beliefs after observing the world. Next, the plans of the pending goals are checked against the current Beliefs. The pending goals are the goals that have no associated plan, but are waiting for submission to be achieved. All top-level goals are pending when the agent starts. The interpreter intends the best applicable plans for the pending goals, according to various selection schemes such as selection by applicability, or meta-level reasoning. The information stored in the Intention Tree can also be used in this step. For example, the interpreter could choose the plan that will not conflict with the executing intention. The intended plans are marked in **ACTION** state, and the information of these plans is recursively updated to the top-level goals.

Next, the interpreter chooses an intention to execute from the Intention Structure. The process is described in section 4.5. The interpreter traces the **ACTION**-state node in the chosen Intention Tree to find the next action to execute. After the execution, the Intention Tree must be updated accordingly. If the action is a subgoaling, the subgoal node will be marked pending, and a plan will be chosen at the next cycle. If the action is loading other plans, every Intention Tree with the goals which can be achieved by these new plans must be updated. The interpreter starts the whole cycle recursively until all the top-level goals have completed.

4.5 Intention Selection

4.5.1 Computation of the Base Priority

The agent produces a node for every Intention Tree that is not in the pending state. The node contains the information gathered from the Intention Tree as described in 4.2.2. Then it computes the base priority of each node. The **top-level plan** is the plan that currently applied to achieve the top-level goal.

```
$base_priority :=  
    top-level goal's Utility × top-level plan's Applicability;
```

List 4.8 Pseudo-Code of Calculating the Base Priority

4.5.2 Applying the Time Constraints

For each deadline a node has, agent adjusts the base priority according to its DUF. Deadline's Utility is the utility of the goal that the deadline applied on.

```
for (each node $n)  
    for (each deadline $d of $n)  
        $delta := $d.deadline - current time;  
        if($delta => $n.maxERT) $raise = $d.DFU1($delta);  
        if( $n.minERT <= $delta < $n.maxERT)$raise = $d.DFU2($delta);  
        if( $delta < $n.minERT) $raise = 0;  
        $n.base_priority += $d.Utility × $raise;
```

List 4.9 Pseudo-Code of Applying the Time Constraints

4.5.3 Applying the Interactions

After uses **interactions_discovery()** to automatically discover the interactions between the nodes, the agent can applied these interactions and produce the interaction links. The **fairness factor** is a user specified value used in fairness priory calculation.

```
interactions_discovery();  
for (each node $n)
```

```

for( each interactions $i of $n)
  if($i is a negative interaction)
    $interaction_links($n,$i.node) +=
      $i.value * (1 + $i.node.DoC);
  else
    $interaction_links($n,$i.node) += $i.value;
if ($n has no interaction with other node)
  $n.base_priority +=
    (current - last execution time) × FAIRFACTOR;
for (each node $on other than $n )
  $base_links($n,$on):= $n.base_priority - $on. base_priority;
  $final_links($n,$on):=
    $base_links($n,$on) + interaction_links($n,$on)

```

List 4.10 Pseudo-Code of Applying the Interactions

4.5.4 Choosing the Best Intention

Finally, the agent chooses the Best Intention, which is the node with no Final Links pointing to it. If there is no such node, the agent iteratively cancels out the Final Links with the value closest to 0.

```

for(each node $n)
  if(for (each node $on other than $n) $final_links($n,$on)=>0)
    choose $n to execute;
  while(no node is chosen)
    final_links_set $fl_set:=
      set of final_links which has value closest to 0;
    delete the final_links $fl
      which has highest deadline-rasie in $fl_set;
    $n = $fl;
  if(for (each node $on other than $n) $final_links($n,$on)>0)
    choose $n to execute;

```

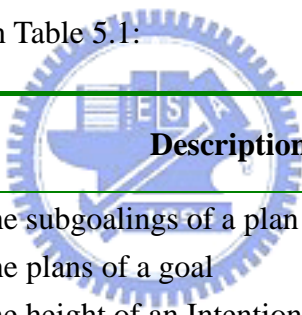
List 4.11 Pseudo-Code of Choosing the Best Intention

Chapter 5. Evaluation

In this chapter, a simple simulation is conducted in order to show the effectiveness of our approach.

5.1 Our Simulation Approach

Because the efficiency of the scheduling heavily depends on the tasks the agent will receive, to show the effectiveness, we use a **random task generator** to issue goals and plans to the agents. The generated goals have random utility, deadlines, and applicable plans, etc. The generated plans have random applicability, actions, ERT, level of subgoalings, and interactions, etc. The value ranges and descriptions of the random parameters is shown in Table 5.1:



Parameter	Description	Value range
SubGoalDegree	The subgoalings of a plan	0~3
PlanDegree	The plans of a goal	1~4
TreeLevel	The height of an Intention Tree	2,4,6,8
Step	The steps in a plan	1~12
DeadlineTightness	The times units before the goal timeout	10~100
Applicability	The applicability of a plan	0.7~1.0
Utility	The utility of a goal	10~100

Table 5.1 Parameters in the Random Task Generator

Two more controlling factors are applied to control the random process. The **Deadline Density** factor is applied to control how many goals will have deadline. The higher the factor, there will be more deadlines. The **Interaction Density** factor is applied to control how many plans will have interaction to each other. The higher the factor, the more interactions will be.

To focus on the scheduling algorithm, there are two kinds of interactions

considered only. First one is Forbid, which causes some other plans to fail. Second is Promote, which causes some other plans to success without execution.

Each agent is given 50 top-level goals, and uses different scheduling schemes:

Normal: consider only the utility of the top-level goals

Time-limited: consider the utility of the top-level goals and the deadline

Interaction: consider the utility of the top-level goals and the interactions
between the goals

Interaction + Time-limited: consider the utility of the top-level goals, the
deadline and the interactions between the goals

After the execution, the agent will report three values, which are:

SU : the sum of utility of the goals that succeed / total utility.

FU : the sum of utility of the goals that fail / total utility.

TU: the sum of utility of the goals that are timeout / the sum of utility of the
goals with deadline.

The results are showed in following sections.

5.2 Simulation Results

5.2.1 Considering Deadline

Figure 5.1 shows the results of the agents using Normal and Deadline scheduling schemes. The Interaction Density is held at 0.15, and the Deadline Density is changed from 0.1 to 0.3. The figure shows that when there are more goals have deadline, the performance of Normal scheme falls. On the other hand, the performance of Time-limited scheme stays rather unchanged.

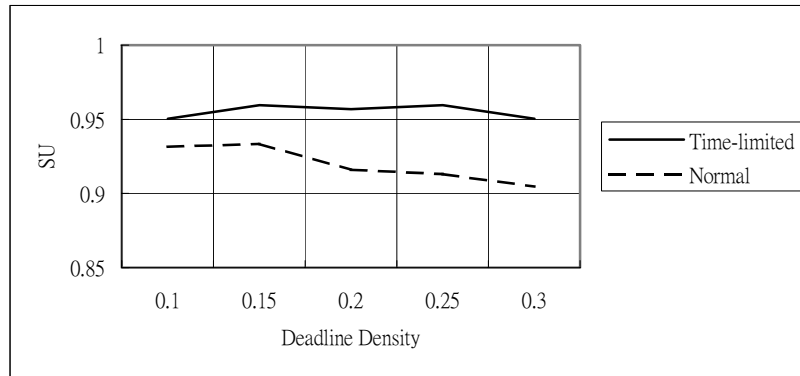


Figure 5.1 Normal versus Time-limited – SU

The results are also shown in figure 5.2. The Normal TU increases rapidly when the Deadline Density increases. However, the performance gain in the Time-limited has some side effect. In Figure 5.2, the Time-limited FU is slightly higher than the Normal FU. That is, Time-limited scheme will cause more fail because it doesn't consider about interaction.

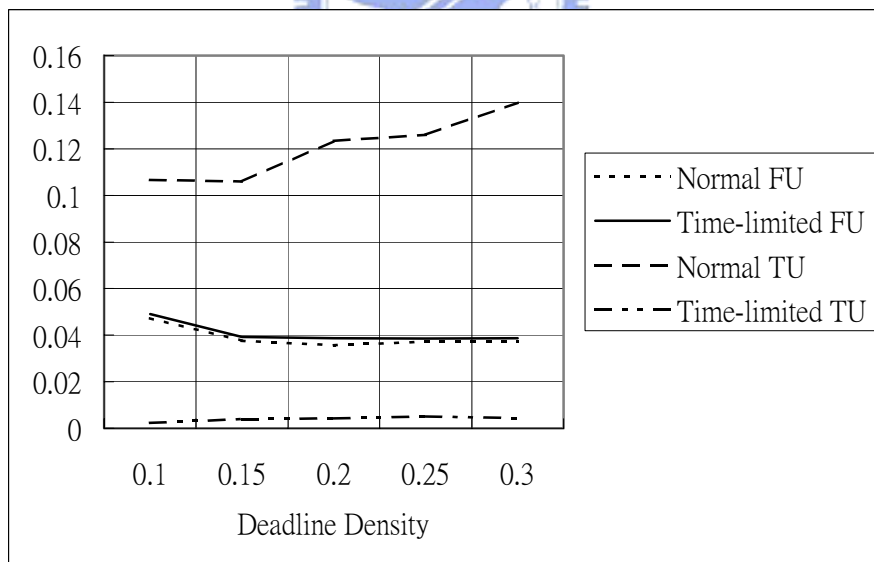


Figure 5.2 Normal versus Time-limited–FU, DU

5.2.2 Considering Interactions

Figure 5.3 shows the result of the agents using Normal and Interaction scheduling schemes. The Deadline Density is set to 0, that is, no goals will timeout.

The Interaction is changed from 0.1 to 0.3. The figure shows that when there are more interactions between plans, the Normal scheme will fall drastically.

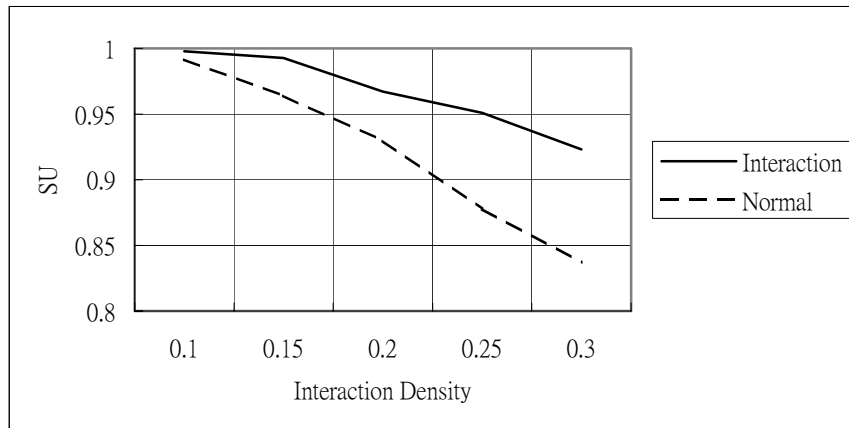


Figure 5.3 Normal versus Interaction – SU

5.2.3 Considering both Deadline and Interactions

Figure 5.4 shows the result of all four kinds of schemes when the Deadline Density is varied from 0.1 to 1 and the Interaction Density is set to 0.1. The SU of Deadline and Interaction+Time-limited schemes fall not quickly because they both consider deadline. The Interaction scheme has higher SU than Time-limited scheme when the Deadline Density is low. However, when deadline starts to contribute more fails than interaction, the SU of Time-limited scheme is better than the SU of Interaction scheme. SU of Interaction+ Time-limited scheme remains the highest among four until the Deadline Density is close to 0.9. Nevertheless, in the real application, Deadline Density seldom reaches so high. Thus, the Interaction+ Time-limited scheme is generally good in most situations.

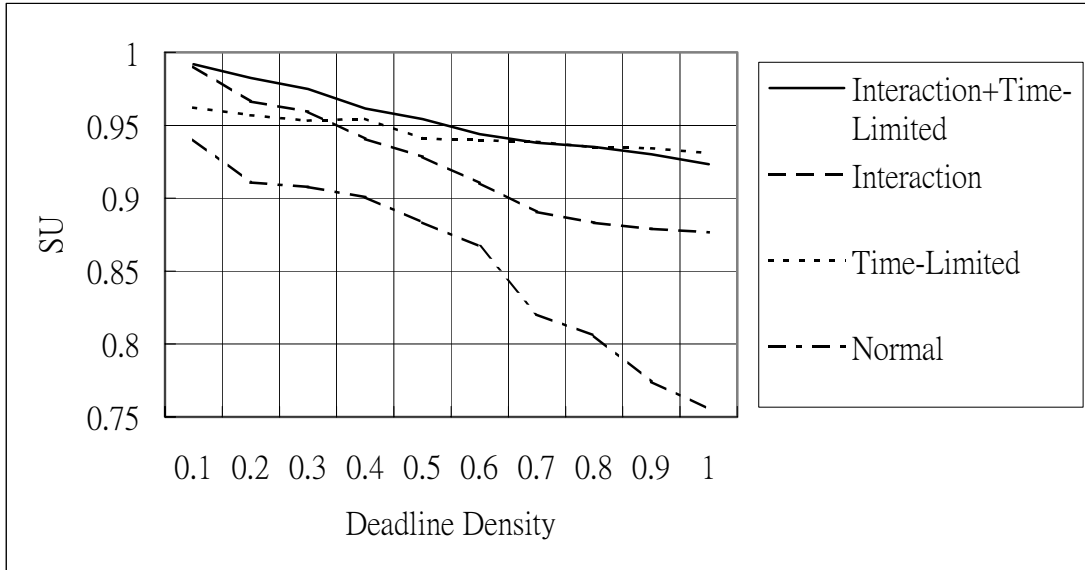


Figure 5.4 SU of all schemes while Deadline Density changes

Figure 5.4 shows the result of all four kinds of schemes when the Interaction Density varies from 0.1 to 0.5 and the Deadline Density is set to 0.3. The SU of Interaction+ Time-limited scheme remains the highest among the four.

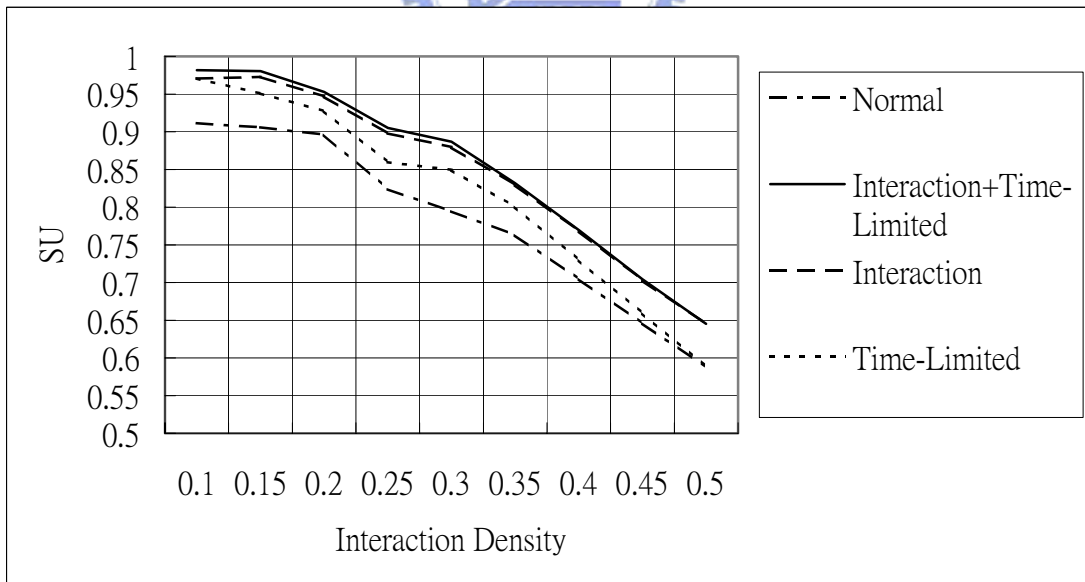


Figure 5.5 SU of all schemes while Interaction Density changes

Chapter 6. Conclusions & Future Work

Many BDI agent theories have been focused on agent's ability to choose different plans for different situations. However, the problem, how to choose among the intentions to execute, is seldom mentioned. In the systems with multiple agents of mobility and intelligence, a good intention-scheduling scheme will greatly improve BDI agents' performance.

In this thesis, we aim at developing an effective intention-scheduling scheme. We discuss the various factors affecting the intention scheduling. Moreover, we propose an Intention Tree model to describe the structural and behavioral aspect of the agent, and use this model to gather the information needed in the scheduling process. We also give an intention-scheduling algorithm, which consider the utility, time constraints and task interactions at the same time. Some implementation issues and major parts of the proposed algorithms are explicitly addressed in pseudo-code. Finally, we show the effectiveness of our approach in a simple simulation.

There are several problems not thoroughly discussed in this thesis, such as automatic interaction discovery, usages of Intention Tree structure in intending process, intention reconsideration, and other scheduling considerations in multi-agent environment. In the future, we plan to delve into these problems and integrate them to produce an efficient BDI agent system.

Reference

- [1] Paolo Busetta and Kotagiri Ramamohanarao. "An Architecture for mobile BDI Agents." Proceedings of the 1998 ACM symposium on Applied Computing, Pages 445-452,1998.
- [2] Bratman, M.E(1987) : "Intentions, Plans, And Practical Reason." Harvard University Press, Cambridge, MA, US, 1987. ISBN (Paperback): 1575861925
- [3] Chia-Lin Hsu, Hwai-Jung Hsu, Da-Ly Yang and Feng-Jian Wang. "Constructing a Multiple Mobile-BDI Agent System." The 14th Workshop on OOTA,2003.
- [4] Yoav Shoham. "Agent-Oriented Programming." Artificial Intelligence, 60(1):51-92, March1993. ISSN 0004-3702
- [5] Philip R. Cohen and Hector J. Levesque. "Intention is Choice with Commitment." Artificial Intelligence, 42(2.3):213-261, March 1990.
- [6] Michael P. Georgeff and Amy L. Lansky. "Reactive Reasoning and Planning." In Proceedings of the Sixth National Conference of the American Association for Artificial Intelligence (AAAI'87), volume 2, pages 677-682, Seattle, WA, July 1987. Morgan Kaufmann.
- [7] Marcus J. Huber. "JAM: A BDI-theoretic Mobile Agent Architecture." In Proceedings of the third annual conference on Autonomous Agents, pages 236-243, Seattle, Washington, United States, 1999. ACM Press. ISBN 1-58113-066-x.
- [8] Anand S. Rao and Michael P. Georgeff. "Modeling Rational Agents within a BDI Architecture." Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91), pages 473-484, San Mateo, CA, USA, 1991. Morgan Kaufmann. ISBN 1-55860-165-1.
- [9] Anand S. Rao and Michael P. Georgeff. "BDI-Agents: From Theory to Practice." In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95), San Francisco, USA, June 1995.
- [10] Michael P. Georgeff and François F. Ingrand. "Decision-Making in an Embedded Reasoning System." In Eleventh International Joint

Conference on Artificial Intelligence (IJCAI' 89), volume 2, pages 972-978. Morgan Kaufmann, August 1989.

- [11] Michael J. Wooldridge and Nicholas R. Jennings. *"Intelligent Agents: Theory and Practice."* Knowledge Engineering Review, 10(2):115-152, June 1995.
- [12] Jaeho Lee, Marcus J. Huber, Edmund H. Durfee, and Patrick G. Kenny. *"UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications."* In Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS'94), pages 842-849, Houston, Texas, March 1994.
- [13] K. L. Myers and D. E. Wilkins, *"The Act Formalism."* Version 2.2, SRI International Artificial Intelligence Center, Menlo Park, CA, September 1997.
- [14] URL: http://www.marcush.net/IRS/irs_downloads.html
- [15] Rafael H. Bordini, Ana L.C. Bazzan, Rafael de O. Jannone, Daniel M. Basso, Rosa M. Vicari. *"AgentSpeak(XL): Efficient Intention Selection in BDI Agents via Decision-Theoretic Task Scheduling."* (AAMAS'02): 1294-1302, Bologna, Italy.
- [16] K. S. Decker and V. R. Lesser. *"Quantitative modeling of complex environments."* International Journal of Intelligent Systems in Accounting, Finance and Management, 2(4): 215–234, 1993.
- [17] T. Wagner, A. Garvey, and V. Lesser. *"Criteria-directed heuristic task scheduling."* International Journal of Approximate Processing, Special Issue on Scheduling, 19(1–2):91–118, 1998.
- [18] V. R. Lesser. *"Reflections on the nature of multi-agent coordination and its implications for agent architecture."* Autonomous Agents and Multi-Agent Systems, 1(1): 89–111, 1998.
- [19] John Thangarajah, Lin Padgham and James Harland. *"Representation and Reasoning for Goals in BDI Agents,"* In Twenty-Fifth Australasian Computer Science Conference (ASCS2002), Melbourne, Australia.
- [20] John Thangarajah, Michael Winikoff, Lin Padgham and Klaus Fischer. *"Avoiding Resource Conflicts in Intelligent Agents."* Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002): 18-22,

Lyon, France.

- [21] John Thangarajah, Michael Winikoff and Lin Padgham. “*Detecting & Exploiting Positive Goal Interaction in Intelligent Agents.*” In Proceedings of the 2nd international joint conference on Autonomous agents and multi-agent systems (AAMAS '03): 401-408, Melbourne, Australia.
- [22] John Thangarajah, Lin Padgham, Michael Winikoff. “*Detecting and Avoiding Interference Between Goals in Intelligent Agents.*” In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003): 721-726, Acapulco, Mexico.

