1-1-2006

Rational hierarchical planning and coordination in multi-agent systems.

Dong Liang
University of Windsor

NOTE TO USERS

RATIONAL HIERARCHICAL PLANNING AND COORDINATION IN MULTI-AGENT SYSTEMS

by

Dong Liang

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

Your file Votre référence
ISBN: 978-0-494-35930-3
Our file Notre référence
ISBN: 978-0-494-35930-3

ABSTRACT

Multi-agent planning involves agents planning to achieve a set of common goals. In dynamic environments, agents are required to make autonomous decisions at execution time. Coordination is necessary to manage the interdependencies between activities performed by all agents and to achieve intended goals. This thesis uses a framework that incorporates notions from game theory and Hierarchical Task Network planning to achieve a degree of coordination and ensure that the agents exhibit some form of rational behavior. The utility-based approach incorporates spatiotemporal factors in assessing the utility of various desired goals, and in selecting the appropriate intentions.

To test the proposed approach, a multi-agent urban disaster simulation environment is used where rescue agents cooperate to extinguish fires, save injured agents, and clear blocked roads.

These tests show that in many cases, the proposed approach has outperformed other approaches that rely on different heuristics in planning and coordination.

ACKNOWLEDGEMENTS

I would like to give my thanks to Dr. A. Tawfik, my supervisor, for giving me such a great chance to participate in Robocup 2004 (Lisbon), and his guidance and suggestions.

I am very grateful to the contributions of other ARK Team members: Zina Ibrahim, Robert Price, Li Qin and Zhiwen Wu.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1 Introduction to MAS

The study of multi-agent systems (MAS) is a subfield of Artificial Intelligence that aims at providing both principles for construction of complex systems involving multiple agents, and mechanisms for coordinating the behaviour of independent agents. In MAS, an agent has incomplete information or capabilities for solving a problem. Usually, there is no centralized control, and computation in MAS is asynchronous (Sycara, 1998).

Since the 1980s, MAS have become more popular. MAS have been used in a variety of application domains, ranging from industry to space, and military. Industrial applications include manufacturing process control, air traffic control, multi-armed robots and mobile robots, which were the initial motivators of MAS research. Web agents and grid computing agents are examples of information management agents, composing queries or providing services, and controlling workflows on computational grids. Simulation environments examples are goal-directed agents for training or electronic games. Managing crisis situations examples include the cleanup of toxic waste, nuclear power plant decommissioning, fire fighting, search and rescue missions, security, and surveillance.

1.1 MAS Motivations

While single-agent systems are more intuitive than multi-agent ones as single-agent systems are more efficient than multi-agent systems when they deal with sequential tasks. A single-agent communicates only for two reasons: to gather problem information or to present results. Therefore, in single agent systems communication requirements are simpler than in MAS. However, when dealing with complex tasks, multi-agent systems

have more advantages over single-agent systems. Multi-agent systems are more efficient due to the parallel execution of asynchronous computations; and more robust because the functionality of the whole system does not rely on a single agent.

In a single-agent system as illustrated in Figure 1-1, there are three kinds of constituents: the agent, the environment, and their interactions. The agent is an independent entity with its goals, actions, and knowledge that is situated in the environment. In a single-agent system, no other such entities are recognized by the agent. Therefore, even if there are indeed other agents in the world, they are not modeled as having goals, actions, and knowledge. They are just considered part of the environment.

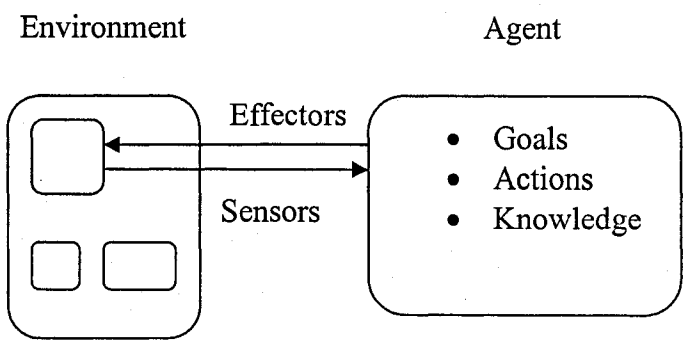

Figure 1-1 A Single Agent System Framework

There are many different kinds of multi-agent systems, some have competing objectives (e.g. self-interest agents), some have distributed control, some have communication constraints (e.g. delay, privacy), and some have computation constraints (e.g. processing power, concurrency). In many situations, there are different heterogeneous agents (robots) or teams with different goals, capabilities and knowledge,

that have to interact as part of a MAS. Despite their diversity, MAS share some common properties.

The difference between Multi-agent systems and single-agent systems is that several agents exist in the MAS model and each agent has a set of goals and actions. As illustrated in Figure 1-2, in the general multi-agent scenario, there may be direct interactions among agents (communication). From an individual agent's perspective, the environment can be affected by other agents. All multi-agent systems can be viewed as having dynamic environments and each agent is both part of the environment and modeled as a separate entity.

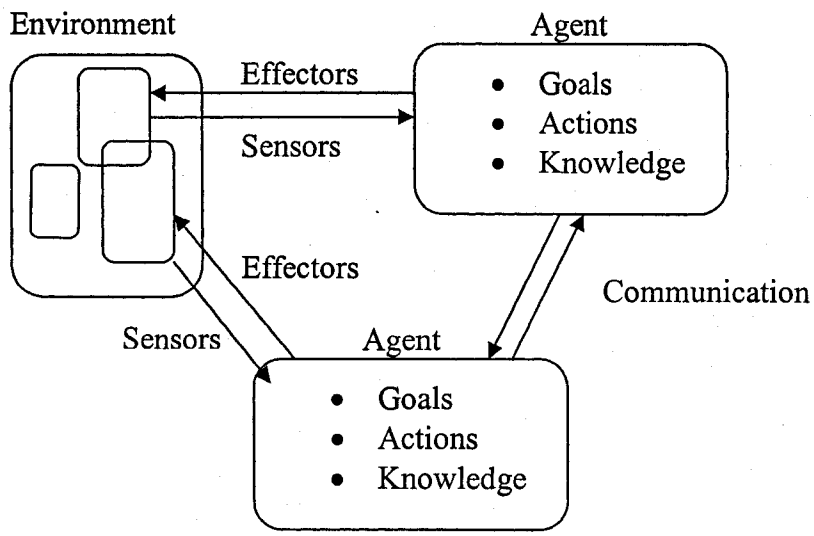

Figure 1-2 A General Multi-agent Framework

The motivations for the increasing interest in MAS include (Sycara 1998):

(1) Resource constraints. Some problems are too large for a single-agent to handle because of resource limitations. Moreover, the risk of having one centralized system could lead to a performance bottleneck or critical failures.

(2) Business needs. MAS can interconnect and interoperate with multiple existing legacy systems. To keep pace with business needs, legacy systems must periodically be updated. It is too expensive to completely rewrite such software. Incorporating legacy systems into an agent society can be done, for instance, by wrapping an agent around the software to enable it to interoperate with other systems. Therefore, in the short to medium term, the best way that such legacy systems can remain useful is to incorporate them into a wider cooperating agent community in which they can be exploited by other pieces of software.

(3) Society needs. MAS can provide solutions to problems that can naturally be regarded as a society of autonomous interacting agents. For example, in meeting scheduling, a scheduling agent that manages the calendar of its user can be regarded as autonomous when it interacts with other similar agents that manage calendars of different users. Such agents also can be customized to reflect the preferences and constraints of their users. Other examples include air-traffic control and multi-agent bargaining for buying and selling goods on the Internet.

(4) Distributed information sources. MAS can provide solutions that efficiently use spatially distributed information sources. Examples of such domains include sensor networks, seismic monitoring and information gathering from the internet.

(5) Distributed expertise. MAS can provide solutions in situations where expertise is distributed. Examples of such problems include concurrent engineering, health care, and manufacturing.

1.2 What is an Agent?

As shown in Figure 1.2, agents can sense, perceive and affect the environment; and they interact with other agents within their environment. Then, what is an agent in MAS? "An **agent** is anything that can perceive its environment through sensors, and can act upon that environment through effectors" (Russell and Norvig, 2003). An agent could be an entity, such as a robot, with goals, actions, and domain knowledge, situated in an environment. There are many different kinds of agents. This thesis deals mostly with autonomous agents. An autonomous agent is an agent whose decision-making relies on its perceptions as well as on prior knowledge (Vlassis, 2003). An autonomous agent is able to rationally balance proactive and reactive behaviors. For example, humans, robots, or software agents can be considered autonomous when they independently make their own decisions. A reactive agent's decision making does not rely on reasoning. It chooses the actions based on some prior knowledge, the decision making goes from Percepts to Events, and then Actions. A purely reactive agent has no representation of their environment, no reasoning based on perceptions. Decision making is implemented in some form of direct mapping from events to actions, in other words, it is a stimulus-response pattern of behavior. Unlike a reactive agent, a reasoning agent has beliefs, goals, and plans.

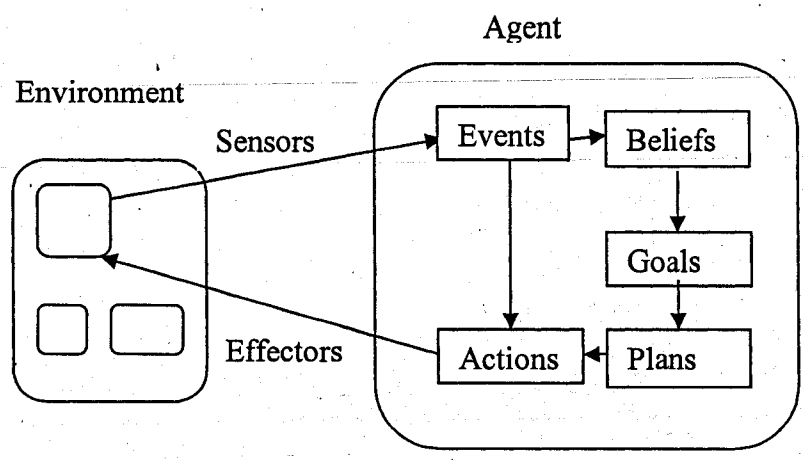

Figure 1-3 A General Autonomous Agent

As illustrated in Figure 1-3, agents are situated in an environment. **Actions** and **percepts** form the interface between the agent and its environment. In order to interact with environment, an agent must recognize significant things when they happen. These significant things are termed **events**. **Beliefs** represent the information generated from the perception of the environment. **Goals** are what agent strives for. In general goals yield autonomy and proactiveness, and also used to measure progress and detect errors. **Plans** are defined as the way of achieving goals. **Decisions** are modeled in a range of ways including explicit commitment. At the last step of the interaction with the environment, the agent has to take some **action**(s).

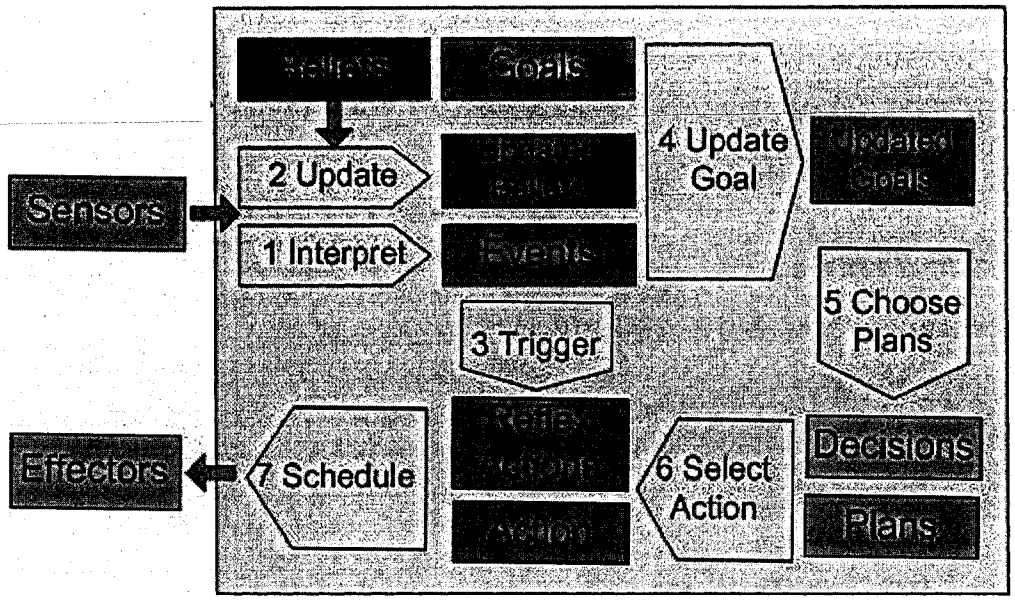

Figure 1-4 Flowchart of reasoning from sensors to effectors

In order to illustrate the agent and environment interaction more clearly, consider Figure 1-4 (Winikoff et al, 2001) as a flow chart:

1. Percepts by Sensors are interpreted to events.
2. Beliefs are updated from percepts and current belief.
3. Events and updated beliefs can yield reflexive actions.
4. Goals can be updated with current goals, beliefs and events.
5. Plans can be generated with goals, current state. This is decision making.
6. Plans yield actions.
7. Actions are scheduled and performed.

Usually a reactive agent goes from 1, 3 to 7. A reasoning agent goes through the steps 1, 2, 4, 5, 6, and 7. A rational agent refers to an agent that always selects an action which optimizes an appropriate performance measure. An optimal decision making problem for

an agent is how the agent can choose the best possible action each time, given what it knows about the world around it. For example, a fire brigade agent must decide optimally when it chooses a fire to extinguish among a set of raging fires, it also has to decide optimally on how to use the current supply of water, when to refill, and which route to take. The performance measure is typically defined by the designer of the agent and reflects what the user expects from the agent with respect to the task at hand.

Homogeneous agents are designed in an identical way and have a priori the same capabilities. In contrary, **heterogeneous** agents have different designs that may involve different hardware, (for example, soccer robots based on different mechanical platforms) or software (for example, software agents running different operating systems) or the agents that are based on the same hardware and software but implement different behaviors (for example, in Robocup Rescue, ambulance agents, fire brigades agents and police force agents have different action sets). Agent **heterogeneity** in multi-agent systems can affect all functional aspects of the agent from perception to decision making, while in single-agent systems the issue is simply nonexistent.

From the communication perspective, agents can be divided into **communicative agents** and **non-communicative agents.** From the goal perspective, agent interactions can be either **cooperative,** meaning that the agents can share a common goal or **selfish** (self-interest), means agents can pursue their own interests.

1.3 Environment

The environment that agents interact with is either **static** or **dynamic.** Static environments (time invariant) are easier to handle and allow for a more rigorous mathematical treatment. For example, chess games are static while dynamic

environments (time variant) change with time as in RoboCup Rescue. The presence of multiple agents makes the environment appear dynamic. This can be a source of difficulties, since the environment changes while the agent reasons about it.

A dynamic real-time environment is one that changes with time even without the influence of agents. A real-time change in the environment is only relevant to a group of agents. This group has to respond to the change by adjusting their goals, plans, actions, or schedules.

The information that can be sensed by the agents in a dynamic MAS environment is typically **spatially distributed** (appear at different locations) and **temporal** (arrive at different times). This automatically makes the world state **partially observable** to each agent. Therefore, agents have to use some spatiotemporal reasoning.

1.4 MAS coordination

A bunch of agents work in a dynamic environment, to achieve their goals, it is essential for agents to decide on an appropriate course of actions, allocate their limited local resources, and finally execute their actions. Sometimes, it is necessary for agents to work together to solve a problem. If the agents work together, they can achieve their common goals faster, and some goals are even impossible to achieve without collaboration. However, in other circumstances, it is more productive for the agents to separate or work in small teams. For example extinguishing a larger fire may require several fire brigades to collaborate, while it is better for rescue agents looking for injured civilians to work individually. The problem of managing interdependencies between agents arises whenever coordination is necessary.

1.5 Overview

The remaining chapters are organized as follows: Chapter 2 introduces game theory as it applies to multi-agents, MAS coordination mechanisms, multi-agent learning and multi-agent planning (MAP). Chapter 3 presents the contribution to MAP based on hierarchical planning and spatiotemporal utilities. Chapter 4 demonstrates the viability of the proposed MAP approach using the RoboCup rescue simulation as a test bed and evaluates these results. The last chapter presents some conclusions and future research.

Chapter 2 Multi-agent Coordination

Multi-agent coordination has been defined as "the act of managing interdependencies between activities performed to achieve a goal" (Malone and Crowston, 1990). Solutions to the coordination problem can be divided into three general classes: solutions based on communications, others based on conventions, and a third class based on learning (Boutlier, 1996). This Chapter starts from the basics of Game Theory, and then shows what the Optimal Joint Action is. Different approaches used in MAS coordination are also presented.

2.1 Game Theory

Game theory is the systematic study of rational players who interact with each other and make choices based on utilities associated with different choices. In multi-agent systems, game theory is used for formal study of conflicts and cooperation between agents as well as in making that potentially affect the interests of the other agents. Game theory can be applied wherever the actions of several agents are interdependent. While, this thesis treats game theory as it applies to autonomous intelligent agents, the theory applies to other situation where the agents may be individuals, groups, firms, or any combination of these. The concepts of game theory provide a language to formulate, structure, analyze, and understand strategic scenarios.

The notion of equilibrium points in n-player games as introduced by Nash (1950) complements the mathematical foundations of game theory as defined by von Neumann and Morgenstern (1944). Game theory has been used to model interactions in political science, economics, and multi-agent coordination. Games can be roughly divided into

two broad areas: non-cooperative (or strategic) games and co-operative (or coalitional) games.

2.1.1 Cooperative and Non-Cooperative

Strategy is a complete sequence of actions for a player. A **strategic game,** also known as a game in normal form, is a game in which players simultaneously choose their strategies. The **utility** in game theory is a measure of the happiness or satisfaction to the goods and services gained. The **payoff** in game theory represents the utility of individual players. There are three constituents comprising a strategic game: a list of participants, a list of strategies for each player, and a utility function (or a list of payoffs) for each player in each combination of strategies.

In a strategic game, each agent chooses a single action and then receives a payoff that depends on the selected joint action. The resulting payoffs are presented in a table with a cell for each strategy combination. The **joint action** is called the outcome of the game. The important point to note is that, although the payoff functions of the agents are common knowledge, an agent does not know in advance the action choices of the other agents. The best an agent can do is to try to predict the actions of others. A solution to a game is a prediction of the outcome of the game using the assumption that all agents are rational and strategic. Pure strategy means the plan of action is completely determined. A specific action is prescribed for each situation. While mixed strategy has a probability distribution over the player's pure strategies.

A classic example is that two gunners face the choice to flee from the advancing enemy or stay and operate their gun. They are stationed at a strategically important, but dangerous, pass. If they both stay, they can assure the enemy's advance will be halted.

However, they each face a significant risk of injury. If they both flee, they will lose the strategic pass, and their capture by the enemy is assured. If just one stays, that soldier will die in battle, while the cowardly soldier escapes unharmed. If each soldier's goal is to survive the attack, preferably unharmed, then each soldier has reason to flee. (Verbeek et al, 2004)

		Gunner 2	
		Stay	Flee
Gunner 1	Stay	2,2	0,3
	Flee	3,0	1,1

Figure 2-1 The Gunners Dilemma (Cooperative Game)

The above diagram in Figure 2-1 is called **payoff matrix**. This diagram shows that the payoff a gunner expects depends on the choices made by both agents. Each gunner has the choice between fleeing and staying. This choice is represented in the rows for gunner 1 and the columns for Gunner 2. Each cell has a pair of payoff numbers, the number on the left side is Gunner 1's payoff, and the number on the right side is Gunner 2's payoff. Games like the Gunner's dilemma, where the agents may or may not collude are called cooperative games. In strictly cooperative games, the payoff matrices are identical.

Consider the case for Gunner 1. Suppose Gunner 2 decides to stay, Gunner 1's best choice is *Flee* and survive without getting hurt. In the formal representation of the matrix, this choice secures a higher payoff (3 rather than 2). Suppose Gunner 2 decides to *Flee*. Again, Gunner 1's best choice is Flee and survive the battle, although there is a risk of imprisonment for the duration of the war. If Gunner 1 were to stay, fight, and then die the

payoff would be lower (0 rather than 1). Gunner 2 is in the same position as Gunner 1, no matter what the other gunner does, the best choice is Fleeing. In short, each individual gunner would be better off fleeing, regardless of what the other does. However, it remains true that the joint payoff would be greater if both stood their ground.

		Player 2	
		Head	Tail
Player 1	Head	1,-1	-1,1
	Tail	-1,1	1,-1

Figure 2-2 Match Penny (Competitive Game)

If the agents (players) are strictly competitive (play against each other), the game is called **non-cooperative.** In non-cooperative game the payoffs are different for each player. In **Figure 2-2** player 1 wins if both agents choose the same coin face while player 2 wins if the two choices are different. Each cell of the payoff matrix corresponds to one player wining and the other loosing. It is a **zero-sum** or **strictly competitive game.**

		Car 2	
		Cross	Stop
Car 1	Cross	-1,-1	1,0
	Stop	0,1	0,0

Figure 2-3 Crossroad (Coordination Game)

Consider another coordination game example (Vlassis, 2003). The game in Figure2-3 represents two car drivers meeting at a crossroad; each agent wants to cross first and get a

payoff 1. However, if they both cross they will crash and both get -1 for payoff. Here, the agents have conflicting interests yet it is in the best interest of both agent to avert an infinite wait or a crash.

2.1.2 Optimal Decision Making

Contrary to single-agent systems, usually there is no central controlling agent that decides what each agent must do at each time; control in a MAS is typically distributed. The decision making of each agent lies to a large extent within the agent itself, this is called **distributed decision making.** Distributed decision making is advantageous for multi-agent systems as there is no need for a central entity that collects information from all agents and then decides what action each agent should take. Each agent is responsible for its decisions. The main advantages of such a decentralized approach over a centralized one are efficiency, and robustness. Efficiency is due to the asynchronous computation. As the functionality of the whole system does not rely on a central agent, robustness is improved (Vlassis, 2003).

Coordination ensures that the individual decisions of the agents result in good joint decisions for the group. The world changes when an agent takes an action. A transition model specifies how the world changes after an action is taken. There are two types of transition models, one is based on a deterministic world model, and the other considers the world as a stochastic system.

Goal is a desired state of the world. **Planning** is the process of searching through the state space for an optimal path to the goal. Some games with no states are called **matrix games.** When the world is deterministic, agent planning comes down to a graph search problem where a variety of methods exist and the game is called Markov game. If each

agent has a deterministic action choice, the game is also called a **pure strategy game.** When the world is stochastic, planning cannot be done by simple graph search because transitions between states are nondeterministic. The agent must now take the uncertainty of the transitions into account when planning. If an agent has non-deterministic action choices, the game is called **mixed strategy game.**

In mixed strategy games, there is a probability distribution over the available strategies of each individual. For example, the gunners could decide to flee with a probability of, say, 1/3 and stay and fight with a probability of 2/3. Whereas before the numbers in the matrix (0, 1, 2 and 3) only signified a *ranking* of the outcomes, here it is assumed that the numbers provide some information to assess an expected payoff. For example, the utility of "2" of the cooperative outcome means that the agent is indifferent between this outcomes and a gamble which offers a "0" payoff with probability 1/3 and "3" with probability 2/3.

Let's use the Gunners' dilemma example, suppose that the gunners have a pair of dice. Now they can realize cooperative distributions other than 2 each. If they agree to throw both dice and if a total of 6 or less comes up Gunner 1 will flee (thus realizing a utility value of 3). However, if the total of both dice is more than 6, Gunner 1 will stay and fight the enemy (realizing his worst outcome of 0). The expected utility of this deal for Gunner 1 is

$$3(^{5}/_{12}) + 0(^{7}/_{12}) = 1.25,$$

while Gunner 2 can expect 1.75 from this deal. In this way the gunners can realize a whole range of outcomes by varying the chances that improves on the non-cooperative

outcome. These outcomes form the bargaining area (see Figure 2-4)(Verbeek et al., 2004).

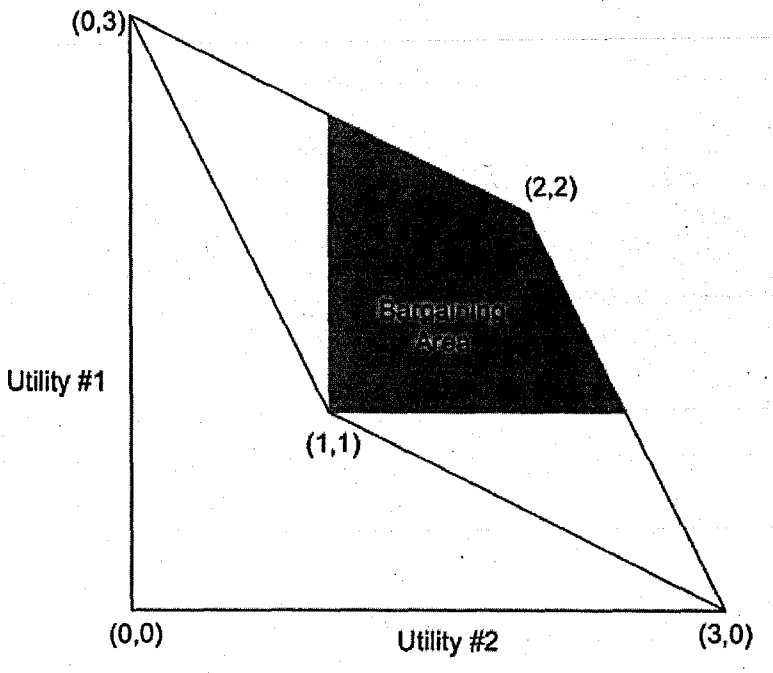

Figure 2-4 The outcomes form the bargaining area

To formalize the notion of state preferences for a specific agent, **utility** of state *s* *U(s)* is used for assigning to each state *s* a real number. If two states *s* and *s'* are possible and *U(s)* > *U(s')*, then the agent prefers state *s* to state *s'*, and if *U(s)* = *U(s')*, then there is no difference between *s* to state *s'*. The utility of a state expresses the desirability of that state for a specific agent. The larger the utility of the state, the better the state is for that agent. For instance, in Figure 2-5, an agent would prefer state c3 than state b2 or c2.

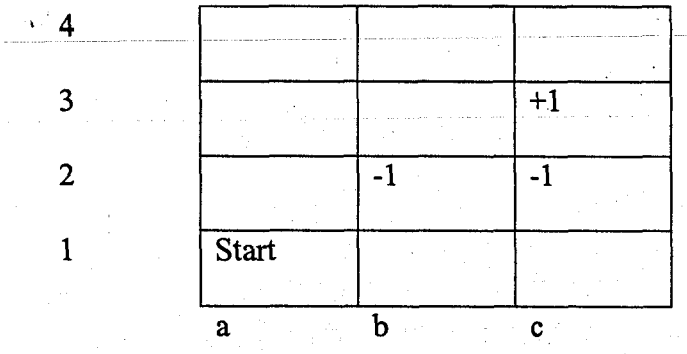

Figure 2-5 A world with one desired (+1) and two undesired (-1) states

In non-cooperative games, a state may be desirable to a specific agent at a specific time and undesirable to another agent. For example in soccer, scoring is rewarding to one team of agents, and obviously unpleasant to the opponent agents.

In a stochastic world, each action could result in one of a set of possible outcomes. Each possible outcome occurs with a certain probability and has an associated utility. To make a decision, the agent has to consider the transition model $P(s_{t+1}|s_t, a_t)$, where the state s_t is the agent's current state, a_t is an action, and s_{t+1} is a possible future state after the action. Let $P(s_{t+1}|s_t, a_t)$ be the probability of the transition and let $U(s)$ be the utility of state s for a specific agent. The **expected utility** (Russell and Norvig, 2003) is:

$$EU(a_t|s_t) = \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)U(s_{t+1}) \quad (2.1)$$

Utility-based **decision making** is based on the premise that the optimal action a_t^* of the agent should maximize expected utility. Therefore,

$$a_t^* = \arg\max_{a_t} \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)U(s_{t+1}) \quad (2.2)$$

where possible states $s_{t+1} \in S$ are the states the agent may transit to, given that the current state is s_t and the agent takes action a_t^*. To evaluate the optimal action a_t^*, the

agent has to multiply the utility of each possible resulting state s_{t+1} with the probability of actually reaching that state, and sum up the resulting terms. Then the agent chooses the action which gives the highest sum.

A policy specifies an action selection criterion such that by applying the policy, a complete mapping from states to actions is obtained. Therefore, choosing the action in each state that result in the largest possible utility value, (as specified in Equation 2.2) constitutes a greedy policy. This policy π maps states to actions in an optimal sense. By choosing the optimal action a_t^* every time, the agent expects a set of optimal utilities $U^*(s_t)$ with respect to a given task. This is also called the optimal policy for the agent (Vlassis, 2003).

The action value function for a state s_t and an action a_t, also known as the **Q-value** $Q(s_t,a_t)$ measures the goodness of action a_t in state s_t for that agent. The maximum Q-value for state s_t and action a_t for a specific agent is: $U^*(s_t) = \arg\max_{a_t} Q(s_t,a_t)$

Consequently, an optimal policy π^* maximizes the Q-values.

$$\pi^* = \arg \max_{a_t} Q(s_t,a_t) \qquad (2.3)$$

The advantage of using Q-values is that they do not require a transition model (Vlassis, 2003). It is enough to have a ranking of the actions in each state.

2.1.3 Iterated elimination of strictly dominated actions (IEADA)

As mentioned in Subsection 2.1.1, the strategic form is the normal type of game studied in game theory. A game in strategic form lists each player's strategies, and the outcomes that may result from each possible combination of choices. The outcome of combination of actions by all players is represented by a separate *payoff* for each player. In another

words, choosing a strategy means making a decision about what to do at each decision point in the game.

In a multi-agent context, it is important to associate an action with the agent taking the action. We use the notation $a_t(i)$ to indicate that agent i takes action a at time t. Furthermore, the notation $a_t(-i)$ refers to the actions taken by agents other than agent i. We say that $a_t(i)$ is **strictly dominated** by another action $a'_t(i)$ of agent i if

$$u_i\,(a_t(-i)\,,\, a'_t(i)) > u_i(a_t(-i),\, a_t(i)) \qquad (2.4)$$

In equation 2.4, s_t represents the state of the world including actions taken by other agents $a_t(-i)$. So, provided that the payoff that agent i receives in state s_t by taking action a'_t is always superior to that resulting for action a_t.

Iterated elimination of strictly dominated actions (**IESDA**) is an intuitive technique used in game theory, which removes strictly dominated actions iteratively for all agents, until no more actions are strictly dominated. It is based on the following two assumptions: 1.Every agent is rational and can predict the outcome using a mechanical procedure; 2. A rational agent would never take a strictly dominated action (Vlassis 2003).

In the Gunners dilemma, as Gunner 1, can see by examining the payoff matrix that payoffs in each cell of the bottom row are higher than the payoffs in each corresponding cell of the top row. Therefore, it can never be rational for the Gunner to play top row strategy (Stay), regardless of what the other does. Since the top row strategy will never be played, we can simply delete the top row from the matrix. Now it is obvious that Agent 2 will flee, since his payoff from fleeing in the two cells that remain is higher than his payoff from staying. So, once again, we can delete the one-cell column on the left from

the game. We now have only one cell remaining *(Flee, Flee)*. Since the reasoning that led us to delete all other possible outcomes depends at each step only on the premise that both players are economically rational, prefer higher payoffs to lower ones, there are strong grounds for viewing joint fleeing as the *solution* to the game, or the outcome on which its players *must* converge. The order in which strictly dominated rows and columns are deleted doesn't matter. Had we begun by deleting the right-hand column and then deleted the bottom row, we would have arrived at the same solution.

As an example, in the Gunners dilemma (Figure 2-1), *Stay* is a strictly dominated action for *Gunner 1*; no matter what *Gunner* 2 does, the action *Flee* always gives *Gunner* 1 a higher payoff than the action *Stay*. Similarly, *Stay* is also a strictly dominated action for *Gunner 2*.

When we apply IESDA to the Gunners dilemma, the action *Stay* is strictly dominated by the action *Flee* for both agents. Let us start from agent 1 by eliminating the action *Stay* from his action set. Then the game reduces to a single-row payoff matrix where the action of *Gunner 1* is fixed (*Flee*) and *Gunner 2* can choose between *Stay* and *Flee*. Since the latter gives higher payoff to *Gunner 2* (1 as opposed to 0 for *Stay*), *Gunner 2* will prefer *Flee* to *Stay*. Thus IESDA predicts that the outcome of the Gunners dilemma will be (*Flee, Flee*).

The agents do not need to maintain beliefs about the other agents' strategies in order to compute their optimal actions in the IESDA algorithm. The only thing that is required is the common knowledge assumption that each agent is rational. Moreover, it can be shown that the algorithm is insensitive to the speed and the elimination order; it will always give the same results no matter how many actions are eliminated in each step and

in which order. However, IESDA sometimes cannot make accurate predictions for the outcome of a game, if IESDA cannot eliminate an action from the action table (Vlassis 2003). For example:

		Player 2	
		C	D
Player 1	A	2,2	1,2
	B	1,2	1,1

Figure 2-6 IESDA not predict

In Figure 2-6 , IESDA cannot eliminate any action from the table.

2.1.4 Nash Equilibrium

Strategy tells player what action to take at each point of game. **Nash equilibrium** is a kind of optimal strategy for games involving two or more players, where no player has anything to gain by changing only one's own strategy. The theorem can be stated as: In the n-player normal form game $G=\{S_1,...S_n; u_1,...u_n\}$, if n is finite and S_i is finite for every i then there exists at least one Nash Equilibrium, possibly involving mixed strategies (Nash 1950). In other word, if there is a set of strategies with the property that no player can reward by changing his strategy while the other players keep their strategies unchanged, then that set of strategies and the corresponding payoffs constitute a Nash equilibrium. Equilibrium represents the mutual and joint action in a shared environment. Every player shares the strategy described in the Equilibrium. Pure strategy Nash equilibrium (NE) is the utility of the new joint action is bigger than or equivalent to any other joint action (2.5).

$$u_i(a^*_i(-i)\ ,\ a^*_i(i)) \geq u_i(a^*_i(-i),\ a_i(i)) \qquad (2.5)$$

Where $a^*_t(i)$ is the optimal action agent i will take at time t, $a^*_t(-i)$ is the optimal joint action taken by the rest of agents, $a_t(i)$ is the any other action agent i can take at time t.

$$u_i(a^*_t(-i), a^*_t(i)) > u_i(a^*_t(-i), a_t(i)) \qquad (2.6)$$

Equation 2.6 is called strict pure strategy NE. NE makes a joint action that no agent can individually improve his payoff, and therefore no agent has any reason to change the NE from an overall perspective. This is quite different to IESDA which is just an algorithm that agent just does take denominated action.

There is an alternative definition of a NE, called best response function. This has been defined as:

$$B_i[a_t(-i)] = \{ u_i[a_t(-i), a_t(i)] \geq u_i[a_t(-i), a_t(i)'] \qquad (2.7)$$

Where $a_t(i) \in A_t(i)$, $a_t(i)' \in A_t(i)$, $a_t(i)'$ is any other action $a(i)$ can take other than $a_t(i)$, at time t . $B_i[a_t(-i)]$ is the agent i's best response to joint action $a_t(-i)$ taken by the rest agents other than agent i at time t. $B_i[a_t(-i)]$ can be a set containing many actions. Using the definition of a best-response function, we can now formulate the Nash equilibrium as: a joint action a^* with the property that for every agent i holds (2.8)

$$a_i^* \in B_i[a_t(-i)^*] \qquad (2.8)$$

That is, at a NE, each agent's action is an optimal response to the other agents' actions. In the Gunner's dilemma, for instance, given that $B1(Flee) = Flee$, $B1(Stay) = Flee$, $B2(Flee) = Flee$ and $B2(Stay) = Flee$, we conclude that $(Flee, Flee)$ is a NE.

Both (2.8) and (2.6) algorithms enumerate all possible joint actions, and then verify which ones meet the condition. The two equilibriums 2.8 and 2.6 of a NE are equivalent. The complexity of both of the algorithms is exponential to the number of agents (Vlassis 2003).

The number of Nash equilibria in a strategic game could be a variant. For example, *(Flee, Flee)* is the only NE in the Gunner's dilemma. We can also easily find that the zero-sum game in Figure 2-2 does not have a NE, while the coordination game in Figure2-3 has two Nash equilibria *(Cross, Stop)* and *(Stop, Cross)*.

NE is a stronger solution concept than IESDA because it produces more accurate predictions of a game. If use IESDA algorithm eliminates all but a single joint action a_t, then a_t is the only NE of the game. A joint action a_t is **Pareto optimal,** if there is no other joint action a_t' for which $u_i(a_t') > u_i(a_t)$ for all agents. Pareto Optimal assumes that each agent i will choose his action deterministically from his action set A_i. However, this is not always true, since sometimes an agent i may choose actions $a_t(i)$ with some probability distribution $p_i[a_t(i)]$ which can be different for each agent.

2.2 Agent Coordination

Coordination is defined by (Vlassis 2003) as "the process in which a group of agents choose a single Pareto optimal Nash equilibrium in a strategic game", which manages the activities performed by agents to achieve a goal, and makes the agents capable of taking their own decisions in a distributed manner. A typical situation where coordination is needed is among cooperative agents that form a team, and through this team they make joint plans and pursue common goals. In other words, agents do not obstruct each other when taking coordinated actions.

2.2.1 Formal Coordination

A formal way to solve a coordination problem is to first model it as a strategic game and solve it according to some coordination mechanism. Then use Nash equilibrium serve as coordination mechanisms. Nash equilibrium is defined in terms of the conditions that hold at the equilibrium point as the payoff that dominates all other equilibria, and ignores

the issue of how the agents can actually reach this point. Therefore, coordination asks how the agents can actually agree on a single equilibrium in a strategic game that involves more than one such equilibrium. Theoretically, we can reduce a coordination problem to the problem of equilibrium selection in a strategic game using game theory.

In the example in Fig 2.3, two cars meet at a crossroad and the drivers have to decide what action to take. If they both cross they will crash, and it is not in their interest to stop. Only one driver is allowed to cross and the other driver must stop. Who is going to cross then? This is an example of a coordination game that involves two agents and two possible solutions: *(Cross, Stop)* and *(Stop, Cross)*. As we known, these two joint actions are Nash equilibriums of the game and they are both Pareto optimal.

In the case of fully cooperative agents, all n agents in the team share the same utility function $u_1(a_i) = ... = u_n(a_i) = u(a_i)$ in the corresponding coordination game.

For example, a coordination game is between two cooperative agents. Two agents wish to go out together to a concert. Each agent has a choice between two types of concerts either Bach or Stravinsky. (Figure 2-7)

	Bach	Stravinsky
Bach	1,1	0,0
Stravinsky	0,0	1,1

Figure 2-7 Bach or Stravinsky

Each agent has no prior information what concert the other agent will choose, and the agents choose independently and simultaneously. Choosing the same concert gives them payoff *1*, otherwise they get payoff *0*. In this game the agents have to coordinate their

actions in order to maximize their payoff. As in the previous example, the two joint actions where the agents choose the same musician are two Pareto optimal Nash equilibriums of the coordination game.

2.2.2 Agent Communication

Communication can be used in **coordination** among cooperative agents or **negotiation** among self-interested agents. Communication, in the sense of making something known or to exchange information with somebody, is a linguistic activity. Communication languages involve syntax, semantics, and must be carried out between parties that have the ability to transmit ideas. Communication between computerized agents should be carefully distinguished from communication in the human sense and restricted to using signals with fixed interpretation.

Computerized agents' communication can be viewed as an action that changes the knowledge state in a MAS. To better achieve the goals of the agents or of the system in which the agents exist, to maintain global coherence without explicit global control, to determine common goals and common tasks, to avoid conflicts, and to pool knowledge and evidence.

Communication methods could be active, passive or both. In other words, an agent could act as a master, a slave or a peer. Communication in a MAS is a two-way process. Message route could be binary (agent to agent), multicast (one agent to multiple agents) and broadcast (one agent to every agent). All agents are both senders and receivers of messages. There are two message types in communication: assertions and queries. All agents accept information by means of assertions. A passive agent accepts queries and

sends replies. An active agent issues queries and makes assertions. A peer agent can do all the above.

Communication is also closely related to network **protocols** which are used for the exchange of information safely and timely. Protocols define the language the agents must speak in order to understand each other and enable agents to have communication. For example, Agent *A1* proposes an action to agent *A2*, agent *A2* evaluates the proposal and sends back to agent *A1*: acceptance, counterproposal, or rejection. Communication protocols enable agents to exchange and understand messages including: propose, accept, reject, retract, disagree, and counter-propose.

There are three aspects to the formal study of communication: Syntax which means how the symbols of communication are structured, Semantics which means what the symbols denote, and Pragmatics which means how the symbols are interpreted. The structure of a protocol includes: Sender, receiver(s), language, encoding and decoding functions.

2.2.3 Social conventions

Social conventions achieve coordination in a large class of systems and are easy to implement. The conventions assume a unique ordering scheme of joint actions that is common knowledge among agents. As the agent designer, we can specify some rules for agents that will instruct the agents how to choose a single equilibrium in any game. A social convention (or social law) (Boutilier 1996) is a set of rules that places constraints on the possible action choices of agents. It can be regarded as a rule that dictates how the agents should choose their actions in a coordination game in order to reach an equilibrium.

In a particular game, each agent first computes all equilibriums of the game, and then selects the first equilibrium according to this ordering scheme. For instance, a lexicographic ordering scheme can be used. In the coordination game of Figure 2-6, for example, we can order the agents lexicographically by $A1 \succ A2$ ($\succ$ meaning that agent 1 has 'priority' over agent 2), or the actions can be ordered by a convention *Bach* $\succ$ *Stravinsky*. The first equilibrium in the resulting ordering of joint actions is *(Bach, Bach)*. Social conventions algorithm can be presented in Figure 2-8. Every agent can first computer all the equilibria in the game, and then choose the action with the biggest equilibrium.

```
For each agent i
{
        Compute all the equilibriums in the game.
        Sort these equilibriums based on a unique ordering scheme.
        Choose action a*(i) with biggest equilibrium a* = [a*(-i), a*(i)] in the
        ordered list.
}
```

Figure 2-8 Coordination by social conventions

There are some more elaborate ordering schemes for coordination, which can dramatically improve the speed in some complex world states. Consider one easy example, the traffic game of Figure 2-3. If traffic lights are available, the driver who sees the red light must stop, the driver who sees the green light will go. If there is no traffic light, but the state contains the relative orientation of the cars in the physical environment and the perception of the agents fully reveals the state, then a simple convention is first come first go, straight going driver has priority over left turn driver, the driver coming from the left gives right of way to the driver coming from the right.

Ordering the actions by *Cross* $\succ$ *Stop*, then coordinate by these social conventions implies that the driver from the right will cross the road first.

2.2.4 Role Play

Agents compute all equilibriums in a game, and then choose an action. However, computing equilibriums can be expensive when the action sets of the agents are large (Vlassis 2003), therefore one would like to reduce the size of the action sets first. Such a reduction can have computational advantages, and more importantly, it can simplify the equilibrium selection problem.

Assigning roles to the agents can reduce the action sets of the agents (Stone and Veloso, 1999) (Nair et al., 2003). Formally, a role can be regarded as a masking operator on the action set of an agent. For example in soccer game, if a player plays offender and possesses the ball, then he has action passing the ball, shooting the ball or making a header. If an agent is assigned a role at a particular state, then some of the agent's actions are deactivated at this state. Again in soccer games, an agent that is currently in the role of defender cannot attempt to do the same things as offender. The actions he can make are tackling the opponents, stealing the ball, etc. In this way, roles can facilitate the coordination game by reducing the action set and making it easier to find equilibriums.

```
For each agent
{
        I = [].
        For each role j = 1, ... , m
        {
                For each agent i = 1, ... , n with i ∉ I
                {
                        Compute the potential role value r_ij of agent i for role j.
                }
                Assign role j to agent i* = arg max_i{r_ij}.
                Add i* to I.
        }
}
```

Figure 2-9 Role Assignment

The algorithm shown Figure 2-9 (Vlassis 2003) assigns a role j to agent i. The agent can compute the role equilibrium, and select the role, then agent i is eliminated from the role assignment process. A new role is assigned to another agent, and so on so forth, until all agents have been assigned roles. After all roles have been assigned, the original coordination game is reduced to a subgame that can be further solved using coordination algorithms like, social conventions. For example, if *Agent 2* is assigned a role that forbids selecting the action *Bach*, then *Agent 1*, knowing the role of *Agent 2*, can safely choose *Stravinsky* resulting in coordination. Then there is only one equilibrium left in the subgame formed after removing the action Bach from the action set of Agent 2.

The algorithm (Figure 2-9) time complexity is polynomial in the number of agents and roles (Vlassis 2003). Furthermore, its precondition is full observability of the state providing that each agent can compute the potential role of other agents. After all roles have been assigned to agents, the original coordination game is reduced to a subgame that can be further solved using coordination by social conventions.

Moreover, the role assignment algorithm can be applied to the game where the state is continuous (Vlassis 2003). If in this case, the algorithm requires a function that computes potential roles, and such a function can have a continuous state space as domain. For example, suppose that in robot soccer we can assign a particular role j (e.g., attacker) to the robot that is closer to the ball than any other teammate.

2.2.5 Coordination graphs

Using coordination graphs (Guestrin et al. 2002) is another method which focuses on reducing the number of agents involved in a coordination game.

The basic idea of coordination graph is to decompose a coordination game into several smaller subgames that are easier to solve. In this framework, a subgame involves a small number of agents, and we can use simpler algorithms to solve it. In this decomposition framework, we assume that the global payoff function $u(a_t)$ can be written as a linear combination of k local payoff functions f_j, each involving only few agents. For example, suppose that there are $n = 4$ agents and $k = 4$ local payoff functions, each involving two agents (in 2.9):

$$u(a_t) = f_1[a_t(A1), a_t(A2)] + f_2[a_t(A1), a_t(A3)] + f_3[a_t(A3), a_t(A4)] + f_4[a_t(A2), a_t(A4)] \quad (2.9)$$

In this equation, for instance, $f_2[a_t(A1), a_t(A3)]$ involves only agents 1 and 3, with their joint action a_t. This decomposition of coordination can be graphically represented by a graph, where each node represents an agent and each edge corresponds to a local payoff function. The decomposition (2.9) can be represented by the graph of Figure 2-10.

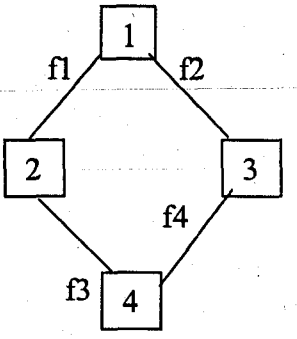

Figure 2-10 A coordination graph for a 4-agent problem

Coordination graphs use this kind of linear decomposition, and can apply utility maximization procedure (as shown in Section 2.1.4 Pareto Optimal Nash Equilibrium) iteratively in this way agents are eliminated one after the other. For example, to eliminate agent 1 in equation 2.9, we maximize all local payoff functions that involve agent *A1* (i.e. *f1* and *f2*.) After the inner maximum of $u(a_l)$ over the actions of agent *A1* equation 2.9 becomes:

$$\max_{a} u(a_l) = \max_{a(A2),a(A3),a(A4)} \{f_3[a_l(A3),a_l(A4)] + f_4[a_l(A2),a_l(A4)]\} +$$

$$\max_{a(A1)} \{f_1[a_l(A1),a_l(A2)] + f_2[a_l(A1),a_l(A3)]\} \quad (2.10)$$

In this equation agent *A1* choose an action that maximizes f_1+f_2, no matter what the combination actions of agents *A2, A3* and *A4* will be. This equation essentially involves computing the best response function $B1[a_l(A2), a_l(A3), a_l(A4)]$ of agent *A1* in the subgame formed by agents *A1, A2, A3* and *A4*, and the sum of payoffs $f_1 + f_2$. The function $B1[a_l(A2), a_l(A3), a_l(A4)]$ can be thought of as a conditional strategy for agent 1, given the actions of agents *A2, A3* and *A4*.

If we define a new payoff function $f_5[a_l(A2), a_l(A3)] = max_{a(A1)} \{f_1[a_l(A1), a_l(A2)] + f_2[a_l(A1), a_l(A3)]\}$ that is independent of $a_l(A1)$. Then, agent *A1* has been eliminated. The equation 2.10 becomes:

$$\max_{a} u(a_t) = \max_{a(A2),a(A3),a(A4)} \{f_3[a_t(A3),a_t(A4)] + f_4[a_t(A2),a_t(A4)]\} + f_5[a_t(A2),a_t(A3)]\} \quad (2.11)$$

Next we can eliminate agent *A2*, using the same technique as we did with agent *A1*. In (2.11), *f4* and *f5* involves a_t2, and maximization of f_4, f_5 over $a_t(A2)$ gives the best-response function $B2[a_t(A2),a_t(A4)]$ of agent *A2* which is a function of $a_t(A3)$, $a_t(A4)$. For this, we can define a new payoff function $f_6[a_t(A3),a_t(A4)]$, and agent *A2* is eliminated. Now we can write:

$$\max_{a4} u(a_t) = \max_{a(A3),a(A4)} \{f_3[a_t(A3),a_t(A4)]\} + f_6[a_t(A3),a_t(A4)] \quad (2.12)$$

Then, agent *A3* can simply choose the action $a_t(A3)$ that maximizes $\max_{a4} u(a_t)$. Agent *A3* is eliminated, resulting in $B3[a_t(A4)]$ and a new payoff function $f_7[a_t(A4)]$. Finally, $max_a\, u(a_t) = max_{a4}\, f_7[a_t(A4)]$, and since all other agents have been eliminated, agent *A4* can simply choose an action $a_t^*(A4)$ that maximizes f_7.The result at this point is a Nash Equilibrium, which is the desired maximum number over $a_t(A1)$, $a_t(A2)$, $a_t(A3)$, and $a_t(A4)$.

For each agent

{

$F = [f_1, \ldots, f_k]$.
For each agent $i = 1, 2, \ldots, n$
{
Find all $f_j[a_t(-i), a_t(i)] \in F$ that involve $a_t(i)$
Compute $Bi[a_t(-i)] = \arg max_{ai} \sum_j f_j[a_t(-j), a_t(j)]$
Compute $f_{k+i}[a_t(-i)] = max_{ai} \sum_j f_j[a_t(-i), a_t(i)]$.
Remove all $f_j[a(-)i, a_t(i)]$ from F and add $f_{k+j}(a_t(-i))$ in F.
}
For each agent $i = n, n-1, \ldots, 1$
{
Choose $a_t^*(i) \in Bi[a_t^*(-i)]$ based on a fixed ordering of actions.
}

Figure 2-11 Coordination graph algorithm

Figure 2-11 (Vlassis 2003) is the summarization of coordination graph algorithm. Where *-i* in $f_j[a_t(-i), a_t(i)]$ refers to all agents other than agent *i* that are involved in f_j, and it does not necessarily include all *n-1* agents. Similarly, in the best-response functions $Bi[a_t(-i)]$ the action set $a_t(i)$ may involve less than *n-1* agents. In this algorithm, an optimal action is computed for the last eliminated agent, and conditional strategies are computed for the other agents. Therefore, in the above example, plugging $a_t^*{}_4$ into $B3[a_t(A4)]$ gives the optimal action $a_t^*(A3)$ of agent 3. Similarly, we get $a_t^*(2)$ from $B2[a_t(A3)]$ and $a_t^*(1)$ from $B1[a_t^*(A2), a_t^*(A3)]$, and thus we have computed the joint optimal action $a_t^* = [a_t^*(A1), a_t^*(A2), a_t^*(A3), a_t^*(A4)]$. Note that one agent may have more than one best-response actions, in which case the first action can be chosen according to an a priori ordering of the actions of each agent that must be common knowledge.

It is assumed that all local payoff functions and an a priori ordering of the action sets of the agents are common knowledge among agents. The latter assumption is needed because each agent will finally compute the same joint action. This algorithm runs identically for each agent in parallel. The main advantage of this algorithm compared to social conventions is that we need to compute best-response functions in subgames involving only few agents, while computing all equilibria in social conventions requires computing best-response functions in the complete game which involves all *n* agents. When *n* is large, the computational complexity of coordination graph would be significantly less than that of social conventions.

By using this algorithm, we can choose a different agent elimination order, the resulting joint action should always be the same. However, the total runtime of the

algorithm will be different. In other words, different agent elimination orders produce different intermediate procedures and subgames. However, the time complexity of computing the coordination graph remains NP-complete (Vlassis 2003).

There are some other heuristics for multi-agent coordination. The common things among all heuristic algorithms are that they reduce the size of agent action set, or divide the game into a set of smaller subgames which could be more efficient to implement, so the agents can coordinate their actions with limited communication and each agent runs an identical algorithm.

2.3 Multi-agent Learning

Multi-agent learning, another popular technique for Multi-agent coordination, helps agents to identify environment information which map directly to coordinated actions. Machine Learning is concerned with computer acquiring new knowledge or updating existing knowledge. Multi-agent Learning (MAL) algorithms are based on Single-agent Learning (SAL) algorithms. The important distinction between MAL and SAL is that an agent in MAS can either learn knowledge from other agents or from the MAS environment.

Since MAL is based on SAL, MAL algorithms could extend SAL algorithms. There are several different approaches in MAL; some popular ones are Multi-agent neural networks, genetic algorithm (GA) and Multi-agent reinforcement learning (MARL).

The idea of neural networks (Hebb, 1949) has its origins in the biological neurons forming the human nervous system. An artificial neural network consists of a set of processing elements organized in layers and joined together by connections known as the synaptic junctions. Each connection has a weight associated with it and the neuron fire if

the weighted sum of it inputs exceeds a set threshold. Current NN models work well in classification, function approximation and pattern recognition. As regard to application of neural network in Multi-Agent systems, agents learn from the input generated from the environment and other agents, or learn from observing other agents. There is no central controller or direct interaction among them. Multi-agent NN applications include voice recognition systems, image recognition systems, industrial robotics, medical imaging, data mining, and aerospace.

The idea of genetic algorithms (GAs) comes from biology also (Holland 1975). People want programs that can evolve like biological entities using combination or mutation of chromosomes to breed new programs. GAs are a good search technique for optimization problems. They work well in many application domains, which have natural and rational encoding formats and fitness functions. GAs could be applicable to multi-agent coordination if the fitness function could be defined in multi-agent settings, thus, GA is a powerful method for artificial life and swarm systems.

MARL is an extension of single-agent reinforcement learning (RL) (Kaelbling et al., 1996). First RL researchers have studied animal behavior under the influence of external stimuli since 1980's. Therefore, reinforcement learning also has some relationship to biology. Reinforcement learning (RL) is a kind of machine learning technique which focuses on finding a policy that maximizes an agent's reward by interacting with the environment. RL emphasis is on how agents can improve their performance in a given task by perception and trial-and-error. The technique of single-agent RL is quite mature. However, the field of MARL is less mature. The main reason is that single-agent RL theoretical results cannot be directly applied in multi-agent systems. MARL can be

implemented in different ways. Some of MARL implementations use state action pairs representation, and some of MARL extend the scope of environment, so that an agent considers other agents' behavior as part of its environment. Some examples like (Hu and Wellman, 2003) present well-understood and practical results. Because of the popularity of MARL, additional information about is presented in the next subsection.

2.3.1 Single-agent reinforcement Learning

The basic reinforcement learning model formally has three parts: a set of environment states S; a set of actions A; and a set of scalar rewards. There are two basic reinforcement learning algorithms: value iteration and Q-learning.

A Markov Decision Process (MDP) uses Markov chains to find the transition matrix based on calculated action-reward pairs available to the agent at each time step. The reward that an agent receives is based on the action and the state. The goal of reinforcement learning is to find a function or a policy, which specifies which action to take in each state, so as to maximize the reward function. At each system time t, the agent perceives its state $s_t \in S$ and the set of possible actions $A(s_t)$. It chooses an action $a \in A(s_t)$ and reaches the new state s_{t+1} and gets a reward $R(s_{t+1})$ as a result. Based on many such attempts, the intelligent agent learns how to maximize the rewards by developing a policy $\pi: S \rightarrow A$. The reward the agent receives is also called reinforcement from the environment. The task of the agent to maximize its total discounted future reward and can be expressed as

$$R(s_t)+\gamma R(s_{t+1})+ \gamma^2 R(s_{t+2})+\ldots \gamma^{k-1} R(s_{t+k}),$$

where $0 \leq \gamma \leq 1$ is future reward discounting, t is current time, k is the time starting at t. If this expression is finite, then MDP has a terminal state. If this expression is infinite,

then MDP has no terminal states. The optimal utility is obtained by a policy that maximizes the expected reward.

$$U^*(s_t) = \max_{\pi} E\left[\sum_{x=t}^{t+k} \gamma^{x-t} R(s_x) \mid \pi, s_x = s_t\right] \qquad (2.13)$$

Where R is reward function, γ is future reward discounting factor, $E(.)$ is the expectation operator which calculate the average value of rewards and stochastic transition rewards, π is the policy, x is a variable of time, t is the current time, k is the time from the t, usually is the end time. An optimal policy $\pi^*(s_t)$ maximizes the utility of the above expression. From this discounted future reward function, we know that the sum will always be finite even with infinite sequences, and the sum will depend on the particular policy of the agent, because different policies result in different paths in the state space (Vlassis 2003).

We can combine (2.13) with (2.3), and get a recursive optimal utility. This is also called the Bellman equation:

$$U^*(s_t) = R(s_t) + \gamma \max_{a_t}\left[\sum_{S_{t+1}} P(s_{t+1} \mid s_t, a_t) U^*(s_t)\right] \qquad (2.14)$$

The solutions of this set of equations define the optimal utility of each state. Because in 2.1.2, we can use $Q^*(s_t,a_t)$ measure the goodness of action a in state s. We can use a recursive Q-value expression to replace the action values in 2.14:

$$Q^*(s_t, a_t) = R(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) \max_{a_t} Q^*(s_{t+1}, a_{t+1}) \qquad (2.15)$$

Where $Q^*(s_t, a_t)$ is optimal action value which is the maximum discounted future reward that the agent can receive after taking action a_t in state s_t. There can be many policies in a given task, but they all share a unique $U^*(s_t)$ and $Q^*(s_t,a_t)$.

What we have seen above is value iteration. The major disadvantages of value iteration is that it assumes that agents know the transition model $P(s_{t+1}|s_t, a_t)$. However, in many applications the transition model is unavailable and unobservable. We would like to have a Q-Learning method that does not require a transition model.

With Q-learning, the agent starts with random estimates $Q(s_t, a_t)$ for each state-action pair, and then begins exploring the environment by interacting with the environment repeatedly and tries to estimate the optimal $Q^*(s_t, a_t)$ by trial-and-error. This is a model-free method. During exploration, the agent forms tuples in the form (s_t, R, a_t, s_{t+1}) where s_t is the current state, R is the reward function, a_t is an action taken in state s_t, and s_{t+1} is the resulting state after executing a_t. Finally, the agent gets its final action value estimation as

$$Q^*(s_t, a_t) = (1-\lambda)Q(s_t, a_t) + \gamma[R + \gamma \max_{a_t} Q^*(s_{t+1}, a_{t+1})] \quad (2.16)$$

2.3.2 Multi-agent reinforcement learning

Applying Single-agent reinforcement learning to multi-agent systems raises is much complicated than Single-agent Reinforcement Learning. Because of the communication reason, they may not observe each other, so they may not model each other correctly. Because of the spatial-temporal factors, the agents may not receive the same rewards, and the agents may not know the payoffs of others.

For simplicity, in cooperative multi-agent systems, we can make several assumptions: Agents can receive the same reward in each time step. Each state is fully observable to all agents. There is a predefined payoffs and joint action table of the agents. Here, we only focus on cooperative systems. The true payoff of a joint action is assumed to be the same for all agents, reflecting average discounted future reward if this action is taken from the

particular state. The world states has a stochastic transition model $p(s_{t+1}|s_t,a_t)$, where s_t is the current state, a_t is the joint action of the agents, and s_{t+1} is the resulting state after a_t is executed. The transition model is unknown to the agents. Like in single-agent Reinforcement Learning, the task of the multi agents is to compute an optimal joint policy that maximizes discounted future reward in the specific environment $p(s_{t+1}|s_t,a_t)$. The payoffs are also unknown to agents, and this is what he must learn. The main task in multi-agent Reinforcement Learning is to guarantee that the individual optimal policies $\pi^*(s_t)$ are coordinated, they indeed define an optimal joint policy $\pi^*(s_t)$ (Vlassis 2003).

There are mainly two different approaches in Multi-agent reinforcement learning, one is Independently Learning and the other is Joint Action Learning.

Independently Learning is the simplest case of multi-agent reinforcement learning. The difficult part of multi-agent learning is that knowledge of $p(s_{t+1}|s_t, a_t)$ does not imply knowledge of $p(s_{t+1}|s_t, a_{ti})$, where a_i is the action of agent i. Therefore, it is reasonable for an agent to use Q-learning to compute its optimal policy $\pi^*(s_t)$, each agent can treat the other agents as part of the environment, and does not attempt to predict their actions. The world changes when agent i takes action a_i in state s. What the world will be changed depends on the actions of the other agents in s_t, and since agent i does not model the other agents there is no way for him to compute $p[s_{t+1}|s_t, a_t(i)]$. However, even Q-learning may not result in coordinated individual policies because its convergence relies on an underlying transition model that is stationary.

Joint Action Learner (JAL) was introduced by (Claus and Boutilier 1998). JAL has better results than independent learning, since the agents attempt to model each other, predict other agents' actions in JAL. Each agent maintains an action value function $Q^{(i)}(s_t,$

a_t) for all state and joint action pairs, which reflects the value of joint action a in state s as modeled by the particular agent. Every time a joint action a is taken in state s and a new state s' is observed, each agent i updates his $Q^{(i)}(s_t, a_t)$. In JAL, each agent has to observe the taken joint actions of whole system. JAL may make the convergence to a Nash equilibrium in multi-agent coordination. However, this equilibrium is not guaranteed be optimal also. The same problem exists in other equilibrium-selection approaches of the game theory.

2.4 Multi-agent Planning

The concept of multi-agent planning (MAP) covers the methodologies and formalizations involved in finding a sequence of actions that can transform some initial state into some state where a given goal is satisfied in the domain of multiple agents having to act together. It is an extension of single-agent planning (SAP), where reaching the goal state is the aim of a single agent, not a group.

The Multi-agent framework has been extended beyond the deterministic plan generation problem along many other dimensions, specifically those involving nondeterministic actions. The MAP domain involves agents planning for a common goal, an agent coordinating the plans of others, or agents refining their own plans while negotiating over tasks or resources, etc.

Multi-agent planning is often associated with another common term, which is multi-agent scheduling. It is important to clarify the difference between these two terms. While in planning agents choose an appropriate course of actions to achieve their goals, scheduling means that agents integrate actions in service of multiple goals and share their limited local resources and finally execute the actions. The tasks that need to be

performed in scheduling are already decided, and scheduling mainly focuses on algorithms for specific problem domains. Because of the overlap in the fields, we will not distinguish them and will use "planning" to refer to both planning and scheduling. Planning and scheduling can be incorporated to help agents improve their coordination behaviors.

Multi-agent planners make action policies for a set of agents that share tasks and results. The key aspect of MAP is that each agent aims to find a plan that has the highest payoff given the plans of the other agents, resulting coordination among the actions of the agents so the highest equilibrium can be achieved in a timely manner.

The above makes clear that the problem of multi-agent planning falls squarely within the setting of n-person cooperative game theory. From the perspective of game theory, the players have a shared or **joint utility function** in n-person games. Any outcome of the game has equal value for all players. Assuming the game is fully cooperative in this sense, it becomes more like a standard (one-player) decision problem where the collection of the n players can be viewed as a single player trying to optimize its behavior to obtain the largest equilibrium against environment.

This section aims to presenting an overview of the formalization of multi-agent planning. We start by presenting an overview of single-agent planning and its methodologies and will then shift to multi-agent planning and its formalization.

2.4.1 Single-agent Planning (SAP)

A plan is a sequence of actions that an agent should follow in order to achieve one or more goals (Bowling et al 2002). The process of generating a plan is called **planning** and is performed via an algorithm called a **planner.**

In order for a planner to come up with a complete plan, it requires as input the initial state of the world *(I)*, a set of possible actions the agent can make to change the world *(A)*, and a set of goals it wants to achieve *(G)*.

Given the above, the definition of **planning can** be restated as the process of finding a sequence of actions that can transform an initial state of the world to a goal state. As the planner goes through the space of plans, it searches for actions that makes goal true if executed.

When the planner searches all the possible situations between the initial state and the goal states, what is called **space searching** takes place. There are generally two approaches to space-search planning, they **progression** and **regression**. Progression (Forward chaining) planning is searching from the initial states to goal state. It starts with choosing action whose preconditions are satisfied, then searching a space of world states for the effect, and continues until goal state is reached. The main problem with this approach is the high branching factor and the thus the huge size of the search space.

As with regression (backward chaining) planning, searching is performed backwards, in other words, from the goal state to the initial state. It starts with choosing an action that has an effect that matches an unachieved sub-goal, then adding unachieved preconditions to the set of sub-goals, then continuing until set of unachieved sub-goals is empty.

An alternative to situation space search is to search through the space of plans rather than the space of situations. This approach, **called plan-searching**, usually starts with a simple incomplete plan called a partial plan. The partial plan is continuously expanded until a complete plan is reached where the problem is solved. The operators in this search

are operators on plans: adding a step, imposing an ordering, instantiating a previously unbound variable, and so on. Plan-space search includes partial-order planning, HTN and etc.

Classic **planners** consist of only primitive actions and keep their goals in a stack and performed an ordered depth-first search through the space of possible plans. Classic Planners are normally represented by STRIPS (Stanford Research Institute Problem solver), of which we give an overview at the end of the section.

Often it is convenient to group sequence of actions into **macro actions** to reduce the search space. In planning with macro actions both goals and action nodes can be expanded either into **macro actions** or **primitive actions**. This approach however, can be insufficient because practically, the designer is the one who plans not the planner. Also plans constructed by macro actions are less flexible, as the planner is simply repeating and combining only predefined sequences.

The planning process depends on the order in which goals are selected for achieving. This is due to the changes that previous actions might have made in the world state. Some improper ordering may cause searching to last longer or even to end up with no solution found. There are several solutions to these problems.

One obvious solution is to try to reorder the goals whenever a solution is not reached. Unfortunately this leads to big computational expenses so instead a hierarchy can be established on the literals of a domain. The most difficult literals for achieving are situated at the top while easily reproducible ones are at the bottom of the hierarchy. The goals in the planner stack are reordered so that higher level ones are achieved prior to satisfying lower level goals.

Goal protection may also cause problems and even prevent us from finding a solution for all possible goal orderings. The only solution in such situations will be to violate goals that are already achieved and then re-achieve them. Vere (Vere, 1992, p. 1168) has proposed a procedure called *plan splicing* that can be used for conflict resolution and as an alternative to permutation of goals upon failure as well. Splicing includes violation of protected goals and recursive deletion of parts of the generated plan. Vere has shown that splicing is efficient even at execution time as a reaction to unexpected events.

An agent perceives the environment and builds a model of the current world state. It can call a suitable planning algorithm to generate a set of actions to achieve a given goal. Single Agent Planning (SAP) explores the state-space of these actions by reasoning about actions. A plan exists if there is at least one ordered sequence of actions that satisfies the goal or goals. Moreover, all ordering constraints should be satisfied. The plan itself could consist of totally or partially ordered set of actions.

The basic elements in a plan are goals, states and actions. The action selection is a central issue in planning. A common approach relies on representing the planning problem as a constraint satisfaction problem (CSP) that can be solved using forward state-space search and/or backward state-space search. The traditional planning algorithm adopted in MAP is based on distributed constraint satisfaction which could be solved by using a search algorithm, like death first search, to find joint action.

However, the CSP approach has some serious drawbacks. First, the number of joint actions increases exponentially with the number of agents. Second, it fails to react in real time to dynamic changes in the environment.

STRIPS uses a representation that includes the initial world state I, goal state G, a set of deterministic actions Ω. Each action $a \in \Omega$ has a list of preconditions and an effect list, denoted by *Prec(a), Eff(a)* respectively. The planning problem involves finding a plan that when executed from the initial state I will achieve the goal G. According to this, a plan can be represented by three-tuple: $P=(A,O,L)$ where $A \subseteq \Omega$ is a set of actions $\{a_i\}$, O is a set of ordering constraints over A, like $\{a_i > a_j\}$, and L is a set of causal links over A. A causal link is a structure with three fields: a set of causal links over A, a set of open conditions, and a set of unsafe links. Finally, an open condition is of the form (p,a), where $p \in Prec(a)$ and $a \in A$.

A generally accepted method of action representation is the one using *preconditions* and *postconditions*, both of which are conjunctions of literals. *Preconditions* define the conditions that should be true in the current state of the world in order to perform an action. *Postconditions* represent the conditions that will be valid after the action is performed. If we need to model changes in the world that will occur when the action is performed, then this is not a primitive but a macro action that should be further decomposed.

The same model can be used to represent simple events. Their preconditions will contain the causes of the event, and postconditions will show its effects. Events can be inserted into plans using the same backward-chaining mechanism. There is only one difference between actions and events in this representation and it is that the event's rules must be allowed to chain forward whenever their preconditions are satisfied.

Search control, action representation, goal protection, time modeling and goals ordering are crucial issues to planning. To illustrate these concepts we will use the well known Blocks world model.

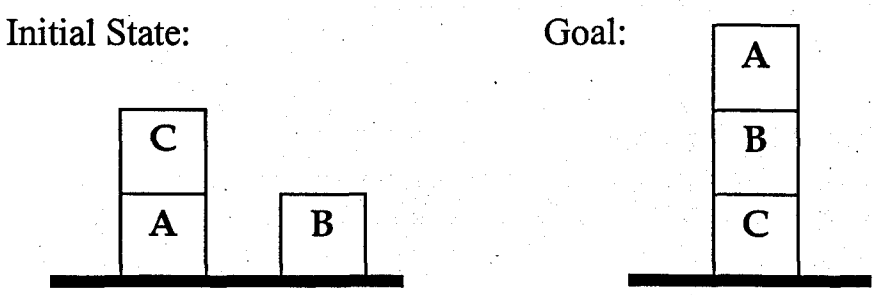

Figure 2-12 Block World Model

In Figure 2-12 above, three blocks A, B, and C are on table as initial state on the left. If putting the blocks one by one as the goal state, the resulting state is what is shown on the right of the figure. The STRIPS representation of the initial state of this example is given below.

Initial State: (on-table A) (on C A) (on-table B)

Preconditions: (clear B) (clear C)

Goal: (on-table C) (on A B) (on B C)

Actions: Pickup(x), Putdown(x, y)

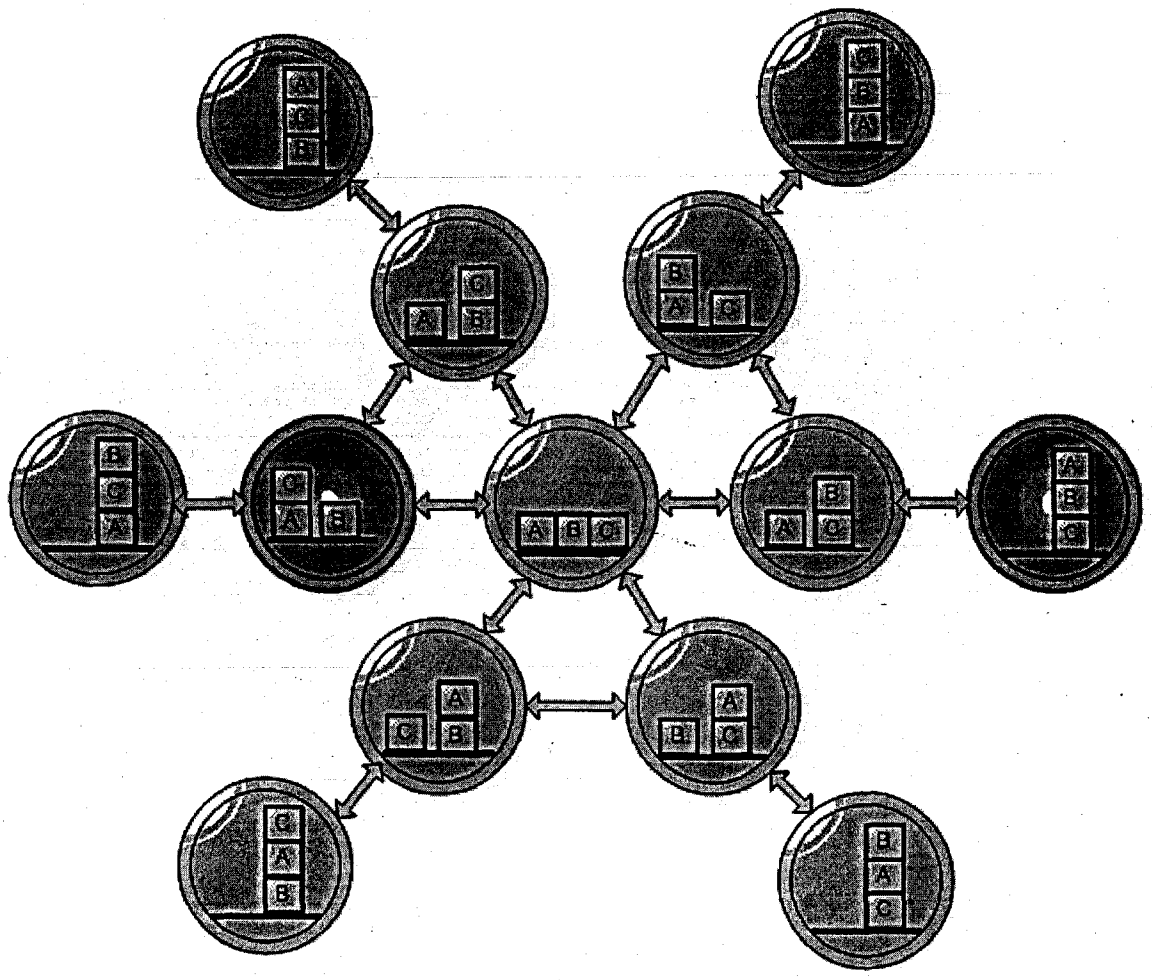

Figure 2-13 Planner representation

By looking at planning as a graph-search problem, planning the problem of going from the first state of Figure 2-12 to the second can be represented by a graph as in Figure 2-13. Nodes are used to represent the individual world states, arcs to represent actions, and the solution is the path from the initial state to the one that satisfies the goal (Russell and Norvig 2002).

2.4.2 Multi-agent Planning (MAP)

As apposed to SAP, Multi-agent planning distributes a global plan among several agents. Multi-agent plan has been generated for multiple executing agents, and the process of its

construction is called multi-agent planning. The plan itself may be built up by one or more agents. MAP generally has two approaches. The first is where one agent creates the multi-agent plan and is called **centralized multi-agent planning**. The second on the other hand is when more than one agent creates the multi-agent plan, and is called **distributed multi-agent planning**.

Centralized MAP has several drawbacks due to its structure. First, the communication infrastructure can have a big impact on the allocation of the global plan. Second, the security and bandwidth of the communication limit the allocation of sub-plans. (E. Durfee 1999). In centralized MAP, global plan is decomposed into sub-plans, sub-plans are allocated to each agent, each agent executes the tasks, and then synthesize global plan. Since the availability of the agents for the sub-plans is not easy to determine without first having devised the sub-plans, allocating the global plan to any current context is not certain. Hence, if there is no agent with a global view of the group activities, each agent should generate the sub-plans alone. Due to all of these issues, we focus on distributed MAP in this thesis.

Because MAP involves more than just the agents planning for a common goal, the tasks include the problems of coordinating the plans and replanning. Consideration must also be given to the constraints placed by the agents and the fact that an individual plan should take concurrent actions into account.

Actions are selected via a specific action-selection mechanism (ASM), which is influenced by the agent's commitments to others, the activities of other agents that may change the environment and the hardly predictable evolution of the world. MAP algorithm has to ensure that the appropriate joint actions are executed at the same time or

in a specific sequence. In MAP, agents need to actually perform the planned actions in order to achieve their goals. Multi-agents coordination problem could be regarded as a planning problem, more specifically, action selection mechanism (ASM) for the multi-agents.

In what follows we present an overview of terms that are necessary to understand the concept of multi-agent planning and its entailments.

Multi-agent Planning Domain (Bowling et al 2002)

A multi-agent planning domain D is a tuple $<P, S, n, A_{i=1...n}, R>$ *where,*

1 *P is the finite set of propositions,*

2 $S \subseteq 2^P$ *is the set of valid states,*

3 *n is the number of agents,*

4 A_i *is agent i's finite set of actions, and*

5 $R \subseteq S*A*S$ *is a nondeterministic transition relation, where* $A = A_1*...*An$ *and must satisfy the following condition. If* $<s, a, s'> \in R$ *and,* $<s, b, s''> \in R$ *then,* $\forall$ *i there exists* $s''' \in S$, $<s, <a_1, \ldots, a_{i-1}, b_i, a_{i+1}, \ldots, a_n>, s'''> \in R$. *I.e., each agent's set of actions that can be executed from a state are independent.*

Multi-agent Planning Problem (Bowling et al 2002)

Let $D = <P, S, n, A_{i=1...n}, R>$ be a multi-agent planning domain. A multi-agent planning problem P for D is a tuple $<D, I, G_{i=1...n}>$, where $I \subseteq S$ is the set of possible initial states and $G_i \subseteq S$ is the set of goal states for agent i.

In MAP research domain, researcher aims at reducing the searching space. Some techniques examples are plan coordination, Hierarchical Task Network (HTN), plan

reconciliation and etc.

Plan coordination means synchronization of plans which may take place at several points, can be done during task decomposition, at the time of plan generation or after the plan is created. Conflicts may appear due to incompatible states, incompatible order of actions or incompatible use of resources during distributed planning. Repeating the multi-agent planning and coordination of individual plans activities can be quite ineffective in uncertain domains and can cause significant delays at execution time. Plan coordination is based on the assumption that some restrictions can be considered by agents before they start planning. Some researchers attempt to impose social conventions on agents, which are prohibited against particular choices of action in particular context (Durfee, 1999).

The Hierarchical Task Network (HTN) captures the possible decompositions of abstract plan steps into more detailed concrete plans. The hierarchical behavior-space search repeating coordination can be avoided by resolving conflicts on abstract (higher) levels of plan representation. Abstract plans can help coordinating with other agents and allows agents to replan in case of failure without affecting the multi-agent plan.

Technique of plan reconciliation is used to assign agent tasks after reasoning through the consequences of doing these tasks in particular orders. Then, align behavior of agents toward common goals, with explicit division of labor achieving greater coordination.

In MAS, planning is difficult, since execution failures and unexpected results of actions complicate the coordination task, besides uncertainty about effects of actions, world states and perception. The external factors can affect goal achievement. The complete multi-agent plan may be invalidated by some unexpected events or outcomes of actions. For internal factors, the number of joint actions increases exponentially with the

number of agents. There are also time and resource constraints as it is difficult react in real time to dynamic changes in the environment. What can also complicate matters further is the difference in preference among agents. For instance, the utility of an agent may give contradicting preference to that of another agent.

Collaboration makes agents cooperate with each other and willing to assist or join with others. There are mainly two mechanisms to make agents collaborate with each other: task sharing and result sharing. Primary task can be decomposed into a number of sub-tasks, called task sharing. Solution synthesis is called result sharing.

According to Boutlier (1996), MAP problem can be solved using simple extensions to SAP. MAP problem can take advantage of characteristics of the problem to make the search simpler, use some flexible search strategies, like ordering tasks in which plan is executed. MAP problem can also use Divide and Conquer strategy, like goal decomposition, if we can assume conjunctive goals achieved independently.

To solve a planning problem, there are some concepts need to be defined to formalize MAP. Some of the definitions are: the possibility of reaching the goal, the possibility of reaching the goal in a finite number of steps. These concepts and their formalization are inspired and highly related to Cimatti and colleagues' single-agent solution concepts (Cimatti *et al.* 2000), and are formalized by Bowling et al with STRENGTH concept, in 2002.

Multi-agent Planning Solutions(Bowling et al 2002)

Let D be a multi-agent planning domain and $P =< D, I, Gi=1...n>$ be a multi-agent planning problem. Let τ be a complete joint state-action table for D. Let $K = <Q, T>$ be

the execution structure induced by τ from I. The following is an ordered list of solution concepts increasing in STRENGTH:

1. τ is a weak solution for agent i if and only if for any state in I some state in G_i is reachable.

2. τ is a strong cyclic solution for agent i if and only if from any state in Q some state in Gi is reachable.

3. τ is a strong solution for agent i if and only if all execution paths, including infinite length paths, from a state in Q contain a state in G_i.

4. τ is a perfect solution for agent i if and only if for all execution paths *s0, s1, s2,* . . . from a state in Q there exists some $n \geq 0$ such that $\forall i \geq n$, $si \in Gi$.

Multi-agent Planning Equilibriums (Bowling et al 2002)

Let D be a multi-agent planning domain and $P = <D, I, Gi=1...n>$ be a multi-agent planning problem. Let τ be a complete joint state-action table for D. Let $K = <Q, T>$ be the execution structure induced by τ from I. τ is an equilibrium solution to P if and only if for all agents i and for any complete joint state-action table τ' such that $\tau'_{j \neq i} = \tau_j$, $STRENGTH(D,P, I, \tau) \geq STRENGTH(D,P, I, \tau')$.

I.e., each agent's state-action table attains the strongest solution concept possible given the state-action tables of the other agents.

2.5 Conclusion of MAS Coordination

In order for the individual decisions of the agents result in good joint decisions for the group, working together harmoniously, agents can use communication to achieve a joint plan. Also, agents may communicate in order to determine task allocation. Social laws and conventions place constraints on the possible action choices of the agents. Social

laws may be imposed by the system through designed rules that dictate how the agents should choose their actions in a coordination game. However, social conventions could not dynamically anew for each problem, thus can lead to inflexibility and breakdown, have limited effects in large dynamic environment (Russell and Norvig, 2003). Multi-agent learning algorithm like Multi-agent Reinforcement Learning (MARL), genetic algorithms, neural networks and etc. improves individual performance as each agent learns. However, the complexity becomes prohibitive in many cases.

Chapter 3 RoboCup Rescue Simulation System Introduction

RoboCup Rescue Simulation System (RCRSS) is designed as a testbed for multi-agent technologies. It simulates a city soon after a large earthquake as buildings burn and collapse. Many civilians are buried in the collapsed buildings. Roads are blocked in the disaster space which slows down ambulance teams and fire brigades as they try to reach injured civilians and buildings on fire. Communications are interrupted, and civilians cannot call for help. However, an emergency communications system allows rescue agents to communicate among themselves and with central stations. With time, fires spread and injured civilians get weaker and start to die. It is an example of a crisis situation management environment with real-time decision making in unpredictable dynamic world.

3.1 RCRSS Introduction

RoboCup Rescue Simulation was developed following the Hanshin-Awaji earthquake. At 5:47 AM of January 17, 1995, Hanshin-Awaji Earthquake hit Kobe, Japan. It registering 6.9 on the Richter scale, the earthquake destroyed buildings, trapped civilians, and started fires throughout the city. The roads became difficult to navigate as they were blocked by debris. Over six thousands people were killed, over one fifth of the cities 1.5 million houses were destroyed, and eighty percent of the city's buildings were unusable. The infrastructure damage exceeded 100 billion US dollars, and total property loss well exceeded 300 billion US dollars.

One of the main reasons for this kind of damage after a large earthquake is fire. Buildings and houses collapse and burn, and many people get buried in the collapsed buildings. As fires propagate, people who could not move are burned. Since roads are

blocked in the disaster space, rescue team such as fire brigade and ambulance can not move through the debris soon enough to carry out rescue missions.

RoboCup Rescue project aims at simulating rescue agents such as ambulance teams, police forces, and fire brigades as they act in the simulated disaster space. Many civilian agents are dead or injured. In order to minimize damage resulting from a disaster, rescue agents have to accomplish their rescue missions. This requires information management to effectively share information, reliable and robust coordination in the distribution of tasks between rescue systems, and immediate transition from ordinary operations to emergency measures (Morimoto et al 2001).

3.2 Structure of RCRSS

According to the RCRSS manual (Morimoto 1999), RCRSS is a real-time distributed simulation system that consists of several modules. Module communication uses sockets to allow modules to run on a set of hosts interconnected by a network (Figure 3-1). Each module can run on a different computer as a separate process. Sub-simulators are modules which simulate disaster phenomena such as road blockade, collapse of buildings and the spread of fires. Ambulance teams (AT), Police Forces (PF), and Fire Brigades (FB) are independent RCR agents. The geographical information system (GIS) communicates with viewers and provides initial conditions of the disaster space, and the viewer visualizes conditions of the disaster space. The kernel is the central controller which manages communications among the modules and integrates all modules into RCRSS (Morimoto 1999).

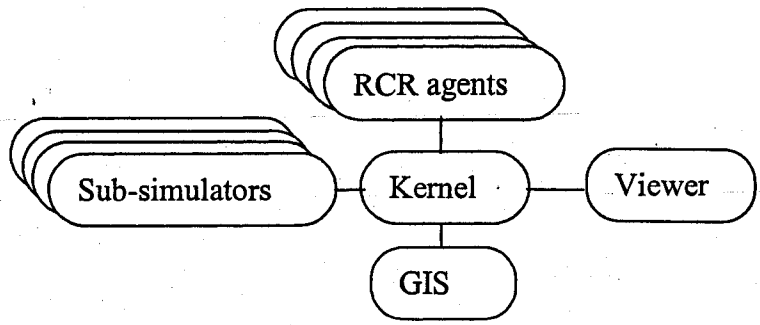

Figure 3-1 Structure of the RCRSS

After RCRSS starts and before the system cycles, first, the kernel connects to the GIS, and the GIS provide the kernel with the initial conditions of the disaster space. The initial conditions are loaded from a configuration file and a map of the disaster space is also specified. The map has a specific format that allows the representation of roads and buildings. Subsequently, sub-simulators and the viewer connect to the kernel, and the kernel sends them the initial condition; Third, RCR agents connect to the kernel with their agent type. The kernel assigns a unique id (9 or 10 digit number) to each rescue agent or civilian. Each agent can hear and see objects within a specified distance from its self. The visual and auditory ranges are initialized by the kernel at this stage.

After the initialization of the agents and the RCRSS is finished, the cycles start. Each cycle in the RCRSS is one second, but corresponds to one minute in the disaster space. There are 300 cycles in each RCRSS run, which simulate the first five hours after the earthquake. In each cycle, RCRSS performs the six steps in Figure 3-2 (except for the first cycle of the simulation where steps 1 and 2 are skipped).

1. The kernel sends individual vision information to each RCR agent.
2. Each RCR agent submits an action command to the kernel individually.
3. The kernel sends action commands of RCR agents to all sub-simulators.
4. Sub-simulators submit updated states of the disaster space to the kernel.
5. The kernel integrates the received states, and sends it to the viewer.
6. The kernel advances the simulation clock of the disaster space.

Figure 3.2 The Six Steps in Each Simulation Cycle

The kernel is designed to wait half a second at steps 2 and 4. However, the actual waiting time depends on the scale of simulation and the speed of the machine. Therefore, all RCR agents must select an action within half a second.

At the first 2 cycles of the simulation, the agents cannot make any action, because the modules of the RCRSS work on sub-simulations. In the 1st cycle, a collapse sub-simulator simulates building collapse, and a fire sub-simulator starts simulating fire spread. In the 2nd cycle: A blockade sub-simulator simulates road blockade based on the result of the collapse simulator, and a miscellaneous sub-simulator starts simulating humans who are buried and injured. Only then, RCR agents start acting.

Evaluation Rule:

To evaluate the performance of agents during a simulation run, the following performance function V is used. The function tries to capture all aspects of rescue agent performance such that higher the return value, the better rescue operation.

$$V=(P + S/Sint) * sqrt(B/Bint) \quad (3.1)$$

Where *P : number of living agents,*

S: remaing Health Point of all agents,

Sint: total Health Point of all agents at start,

B: area of houses that are not burnt,

Bint: total area at start.

3.3 Agents Development Tool-YabAPI

The YabAPI (Morimoto 2001) is an open java API for developing RCR agents. YabAPI consists of four packages: yab.io, yab.io.object, yab.agent, yab.agent.object described in Morimoto (2001) as follows:

> "The yab.io package provides functions for communication between an RCR agent and the kernel. The yab.io.object package provides classes of objects in the disaster space. The yab.agent.object package provides useful classes of objects in the disaster space for RCR agent developers. They wrap the yab.io.object package's classes. The yab.agent package provides the skeletons of RCR agents and utilities for concisely describing their intelligence."

The agent developed with YabAPI is called RCR agent, which controls act of an object in the disaster space. There are seven classes of object: the Civilian, AmbulanceTeam, FireBrigade, PoliceForce, AmbulanceCenter, FireStation, and PoliceOffice. An RCR agent controlling act of a Civilian object is called a civilian agent, an RCR agent controlling act of an AmbulanceTeam object is called an ambulanceTeam agent, and so on. In addition, the ambulance team, fire brigade, and police force agent are called a platoon agent, and the ambulance center, fire station, police office agent are also called a center agent .Both platoon and center agents are called rescue agents.

An agent always perceives the environment and makes an action. The activity of an RCR object consists of a repeating cognition of the surrounding circumstances followed by an action at each cycle. An RCR agent recognizes the surrounding circumstances based upon visual information and auditory information received from the kernel, selects an action, and submits the action command to the kernel. Moreover, an RCR agent communicates with other RCR agents asynchronously.

Different RCR agents have different capabilities for cognition and action, which are limited by the capability of this kind of agent. An RCR agent gets cognition information limited by its visual and auditory capabilities, and sends action commands such as move, rescue, load, unload, extinguish, and clear according to the agent's capabilities. The agent can utter natural voice (action: say) and speak via telecommunication (action: tell).

Civilian	Sense,Hear,Say, Move
Ambulance Team	Sense,Hear,Say,Tell,Move,Rescue,Load,Unload
Fire Brigade	Sense,Hear,Say,Tell,Move,Extinguish,Fill
Police Force	Sense,Hear,Say,Tell,Move,Clear
Ambulance Center	Sense,Hear,Say,Tell
Fire Station	Sense,Hear,Say,Tell
Police Office	Sense,Hear,Say,Tell

Figure 3-3 Actions available for each type of agents

After receiving sensory information (AK_SENSE block), an RCR agent submits an action command (AK_MOVE, AK_RESCUE ...etc.) at will in each cycle. The kernel

adopts only one action command every cycle per agent. If the agent submits more than one command, usually the last command is the one that kernel executes. As stated in Figure 3-3, in each cycle, an agent first receives a sensory block, and then submits an action command after half second. In some cases, agents cannot act, for example, delayed action commands are ignored; also a buried humanoid, whose buriedness is greater than zero, cannot act.

A humanoid can move in the disaster space by submitting an AK_MOVE command, which consists of the current position as the origin and a series of objects reaching the destination. When a humanoid is loaded by an ambulance, the origin is the ambulance's position. An ambulance team can progressively rescue buried humanoids under collapsed buildings by submitting an AK_RESCUE command. Rescuing a humanoid by an ambulance team in a cycle reduces the buriedness of the humanoid by 1 team cycle. If more ambulance teams work on rescuing a humanoid, the humanoid can be rescued in less time. The target humanoid must be at the same position as the ambulance team. Fire brigade agents extinguish fires by summit AK_EXTINGUISH command. However, single fire brigade agent can hardly extinguish a fire. FB agents need to cooperate with other FB agents. They also have to cooperate with different type agents, police forces will help clear the road to a destination. AT and PO can provide fire information.

Rescue agents need to communicate with each other in order to accomplish their missions efficiently. However the RCRSS only provide very limited communication. The maximum length of a message is 256 bytes (128 characters). For platoon agents, a maximum of 4 messages sent and 4 messages received per cycle is specified. For center agents, the maximum 2*n messages sent and 2*n messages received, where n is the

number of platoon agents of the same kind as center. Platoon agents can talk to their colleagues and their center agent. Center agent can talk to other center agents and his platoon agents. If several agents use their maximum allowed messages each cycle, other agents would miss most of these messages. For example, if ten FB agents send four messages in a cycle, the FS agent will only receive 20 messages; and the FB agents will only receive 4 of these messages. Therefore, agents are required to cooperate with minimum communication.

The abilities of each agent are complementary to those of other agents. Thus, an agent must cooperate with agents of different types. For example, it is crucial at the beginning the simulation that the fire brigades coordinate with the police force agents to clear blocked roads leading to fires. Ambulance team agents have the most priority after the fires are extinguished, but still need other agents to search for injured civilians.

Chapter 4 Multi-agent Hierarchical Planning

Having introduced game theory, utility-based agents, hierarchical task network (HTN) planning, the formalization of multi-agent planning, and the RCRSS project, we are now in the position to apply HTN and utility concepts to multi-agent planning in RCRSS. This chapter introduces techniques that use game theory and exploit efficient hierarchical planning.

4.1 Hierarchical Planning

Note that planning is effective in highly predictable environments. The basic purpose behind designing plans is that computing equilibriums could be expensive when the number of agents is large, or the action sets of the agents are large. Therefore we would like to reduce the size of the search space by using two techniques. The first technique classifies the actions into a smaller number of abstract actions; the second reduces the number of agents by grouping them into groups of abstract agents.

The global plan can be described in terms of B Beliefs, D desires and I intentions - BDI architecture (Rao and Georgeff 1991) (Wooldridge 2002). The Desires are Goals but can be inconsistent with one another. Goals are chosen from consistent desires. Formally, agent i's desire is to find itself in the local states: $q^1_i, q^2_i, \ldots, q^T_i$, such that

$$u_i(q^1_i, q^2 i, \ldots, q^T i) \geq \alpha_i$$

Where state q^t is for agent i at time t. u_i is the agent i's utility function, αi is the agent's minimum accepted utility value.

Beliefs B are modeled in each state s as the set of local states and the set of accessible worlds that the agent believes to be possible. These accessible worlds can be represented

with a conventional decision tree. The agent believes to be able to optionally achieve some goals.

Intentions are similarly represented by sets of worlds that the agent has committed to attempt to realize. Intentions lead to actions: if an agent has an intention, she believes that she has such actions which can fulfill her intention.

The compatibility of *B*, *G* and *I* has following relationship (Rao and Georgeff 1991):

1. GOAL(α) ⊃ BEL(α)

If the agent has the goal α, she also believes it.

2. INTEND(α) ⊃ GOAL(α)

If the agent intends α, she has α as a goal as well.

3. INTEND(does(α)) ⊃ does(α)

If an agent has an intention to do a particular primitive action, she will do that action.

4. INTEND(α) ⊃ BEL(INTEND(α))

If an agent has an intention, she believes that she has such an intention.

5. GOAL(α) ⊃ BEL(GOAL(α))

If the agent has a goal to achieve, the agent believes that she has such a goal.

6. INTEND(α) ⊃ GOAL(INTEND(α))

If an agent intends to achieve α, the agent must have the goal to intend α.

To achieve real-time performance, it is common to use a library of plans that contains methods for achieving certain goals. The plans in the library involve a trigger condition ϕ and a body π. The plan body π may be a sequence of actions, some of which are not directly specified but are a command to achieve some subgoal. A plan of this form is represented as $[\pi]\phi$. Where $\pi = a_1; \ldots; a_n$ is the sequence of action. If a plan also

involves a precondition ϕ', then its representation is of the form $\phi'[\pi]\phi$. For Example, *locked(door)[turn(key)]unlocked(door).*

Generally speaking, BDI agent-oriented system is flexible, responsive and well suited for real-time reasoning and control environment. However, "BDI frameworks rely entirely on context sensitive subgoal expansion". There is "a limitation of these systems is that they normally do no lookahead or planning in the traditional sense; execution is based on a user-provided plan library to achieve goals." (Sardina et al 2006).

Hierarchical Task Network (HTN) (Erol et al., 1994) is an approach to planning based on the decomposition of upper level compound tasks into lower level primitive task through a task network. HTN has an on-demand planning mechanism. Hierarchical planners use task hierarchies to search through all combinations of alternatives to select an action, and allocate resources and devises with a sequence of steps to achieve goals. Rather than building a plan from the beginning forward or end backward, hierarchical planners identify promising classes of long-term activities (abstract plans), and incrementally refine them to eventually converge on specific actions.

There are two types of HTN tasks. One is called compound task that cannot be executed directly. The other is called primitive task, an action that can be directly executed by the agent. A task network $d = [T, \phi]$ is a collection of tasks T that need to be accomplished and a Boolean formula of constraints ϕ. Constraints impose restrictions on the ordering of the tasks $(e \prec e')$, or/and on the variables $(x = c)$, or/and on what literals must be true before or after each task (l, e), (e, l), and (e, l, e'). A method (e,ψ,d) encodes a way of decomposing a high-level compound task e into lower-level tasks using task network d when ψ holds. Methods provide the procedural knowledge of the domain. An

HTN planning problem P is a triple $<d, B, D>$ where d is the task network to accomplish, B is the initial state (i.e., a set of all ground atoms that are true in B), and D is the plan domain, $D = (\Pi, \Lambda)$, which consists of a methods library Π and primitive tasks in STRIPS style Λ (Sardina et al 2006).

The key issues in this hierarchical planning are how to maintain the constraints and how to make the agent rational. In game theory, the agent's rationality is how they maximize the utility of the agent relative to its knowledge. Here, we make the agents choose the highest utility action compatible with the abstract action made at each level of the hierarchy (Ambroszkiewicz and Komar 1997).

Therefore, the RCRSS agents use a hierarchical planning architecture and a utility-based action selection approach. A 2-layered hierarchical planner is adopted as the architecture for multi-agent planning. The upper layer has a group of abstract agents; each represents a group of agents defined using abstract payoff and abstract actions. In each cycle, the top layer calculates the utility of abstract actions, determines the desired states, then passes the abstract action to the lower layer. The lower layer agents calculate the utility of lower level subgoals based the beliefs and the abstract action. Finally, teams of agents make joint actions involving groups of homogeneous agents. The top layer makes decisions for the underlying layer and may reset the execution plans developed at the lower layer. This is important to promptly respond to changes in the environment.

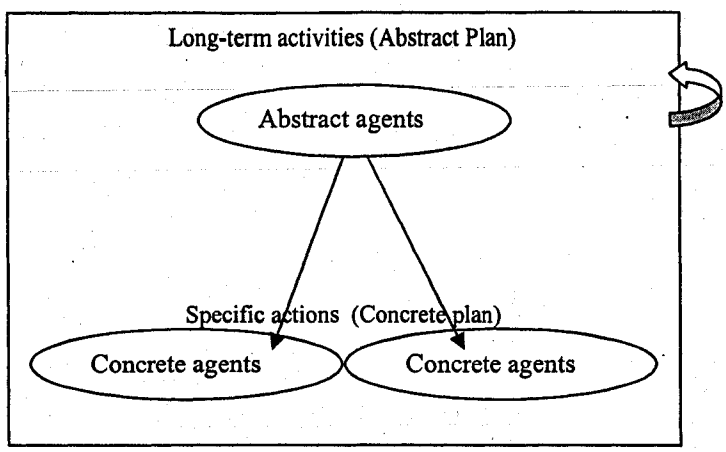

Figure 4-1 Plan Refinement

In this scenario (Figure 4-1), the upper level abstract agents have abstract activities and use upper level utility function. The upper level planning is responsible for defining the resources allocation and also defines the goals for the underneath plan (i.e. to extinguish limited fires or to separate the fired area and safe area). The upper level planning takes consideration of the overall spatiotemporal information (i.e., how many fires and how are them located) and available resources. Therefore, we use game theoretic agents to make the abstract agents. Abstract agents choose the policy by calculating the abstract utility that achieves a Nash Equilibrium at the abstract level.

After getting their abstract action, concrete agents in the lower level use lower level utility functions to calculate lower level utility and execute concrete actions. The lower level planning takes into consideration of the agent's environment and the current situation (for example, route selection may consider blockades on the possible routes), make the concrete action selection decision (choose the best route).

As we know, "the desire is represented as agent's goal to achieve a maximum level of its utility. The intentions are determined by some methods that realize this level of utility. These methods are called rational behaviors."(Ambroszkiewicz and Komar 1997)

Therefore, agent characteristics can represented as (*B, D, Rb*) where *Rb* is the rational behavior. Rational behavior is the reasoning process which conveys the knowledge from upper level into the lower level. The final level determines the final intention.

The upper level planner determines the behavior of the underlying planners, specific abilities implemented by the lower level may be reused for building different roles. In particular the layered approach is convenient for sharing low level abilities that all agents should possess.

The lower level agent maintains a set of task lists. A plan in the lower layer is generated from the list of unachieved tasks, which dynamically change with events in the environment. At each cycle, the interpreter of upper layer planner specifies an action in executable form to the lower layer. The lower layer will generate a plan from the task list. Upon finishing executing an action, the lower layer requests a new action specification from the upper layer, which provides another action afterward (Cisternino and Simi 2000). The architecture of the planner is shown in Figure 4-2 and Figure 4-3.

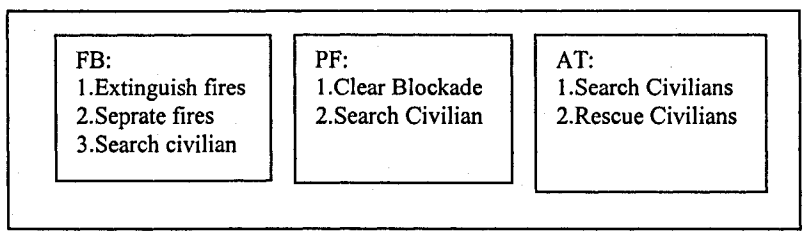

Figure 4-2 Abstract Actions Upper Layer

In Figure 4-2, the upper layer has a global plan for all different types of abstract agents. In the figure, FB stands for Fire Brigade, PF for Police Forces, and AT for Ambulance Team.

The FB abstract agent desires to extinguish or control the fires. The intents of the agents are the present commitments to particular sub-goals that lead to the desire. For a

fire brigade abstract agent, the intents would include specific high priority fires. As for the beliefs, they represent the cognitive interpretation that the agent has from combining available knowledge and current sensory inputs from the environment. The abstract agent may believe that a particular fire is controllable or uncontrollable, or that a fire is more important than another. The plan is made by first choosing the intention according to the FB abstract agent utility function, also based on some success fact.

The global plan would specify how to act based on current beliefs by specifying abstract actions that represent intentions in response to changing beliefs. For our abstract fire brigade, an abstract plan may specify that the agents should try to encircle (or separate) a fire believed to be uncontrollable and try to extinguish a smaller fire.

Similarly, an ambulance team (AT) abstract agent desires to rescue all the injured civilians, intents specify the injured civilians to commit to rescue or an area to target in searching for injured civilians depending on the situation, and the beliefs would include an assessment of the severity of injuries and short term prognosis for each injured agent according to what the sensor get from the environment. An abstract plan may specify that the agents should try to rescue the nearest civilians or those that are in more critical conditions. Figure 4-3 represents the mapping of abstract actions to concrete actions.

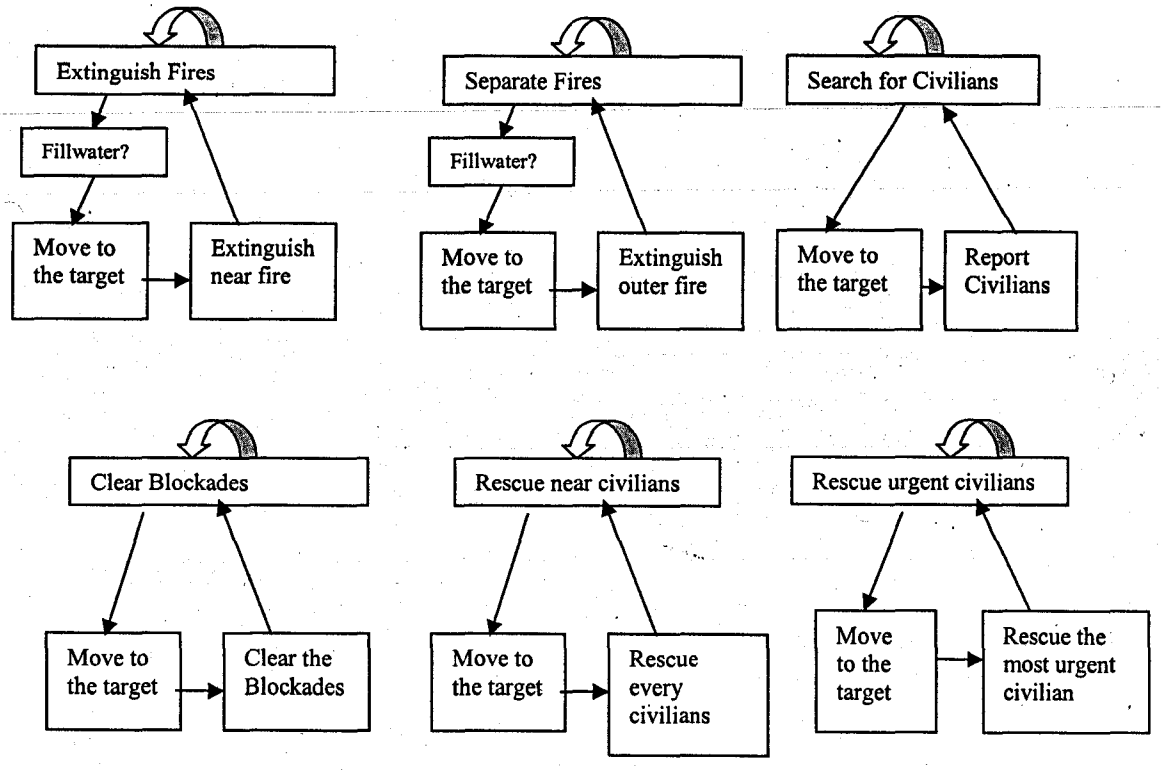

Figure 4-3 Example of mapping abstract actions to concrete actions

According to above, we construct agent in a tuple <*B,D,u,PL,C,Rb*>, where *B* represents Beliefs, *D* desires, *u* the utility function, *PL* the Plan Library, *C* is for constraints, and *Rb* the Rational Behaviour. The whole planning process could be expressed in following algorithm

```
Upper Level Agents:B = B0, D=D0
    For( every cycle) {
        get next percepts p
        B = brf(B,p)
        D = desire_update(D,B)
        Rb = C(B,D)
        If ( not (empty(Rb) or succeeded (D, B) or impossible(D, B)) {
            α = head(Rb)
            execute(α){
                Lower Level Agents In Same Cycle :
                {
                    B' = B, D'=α
                    D=desire _update(B', D')
                    Rd' = PL (B', D')
                    α' = head(Rd')
                    execute(α')
                }
            }
        }
    }
```

Figure 4.4 Hierachical Planning Algorithm

Where *brf* is belief revision, after perceiving the environment, concrete agents send messages to the abstract agents. Once the abstract agents receive all the information, they update their beliefs, and send combined message in "newspaper" format to concrete agents. Therefore all the concrete agents and abstract agents have the same consitent beliefs correspondingly. *Desire_update* take current beliefs and current desire, use utility function *u* and generate the set of Desires. *Rb* is deduced from constraints or selected from the Plan Library.

For example, the Fire Brigades collect the environment information. Then, the Fire brigades send the messages to fire center agent. The fire center agent also gets messages from the two other center agents. After fire center agent has all the information, the center agent updates its beliefs and desires. The Fire center agent makes the abstract plan,

execute the abstract action (specific high priority fires). Then send combined information (called Newspaper,' including upper layer agent intentions) to FB. All FB update their beliefs, determine various options to their goals, which are the current intentions. Finally use utilities to order the subgoals.

In order to achieve coordination, every agent has to take into consideration the actions of other agents. In addition to satisfying known constraints, it is imperative to appeal to the decision theoretic notion of utility due to the non-deterministic nature of hierarchical planning in an uncertain dynamic environment.

For example, the global goal for a fire brigade is to extinguish or control the fires, and help rescue all the civilians in an urban disaster situation, but this depends on the spatiotemporal conditions. The lower level is responsible for choosing a concrete action to execute. Spatiotemporal utilities can be used to guide this choice.

4.2 Spatiotemporal Utility

Utility theory (von Neumann and Morgenstern 1947) is an analytical method for action selection, given multiple criteria upon which the decision is based. A utility function is a measure of the desirability of expected outcomes resulting from actions. By weighing the utility of each outcome by the probability of the outcome, the theory is useful for decision making under uncertainty (White 1969). Therefore, the utility function guides the performance of the agent.

For example, two yogurt companies A and B compete in a particular market. Each company has fixed daily cost of $5,000, the price is $1 or $2 for each container. If the price is $2, the company can sell 5000 containers. At a price of $1, a company can sell 20,000. People in this market has the ability of pay $20,000 for yogurt. At a certain time,

both sell for $10,000 daily and earn $5000 as the two companies charge the same price ($1), and split market. If one of the companies lowers its price, the lower priced yogurts dominate the market. Payoff for a company is:

Payoff=sales reward - cost.

Where *sales reward* is the gross amount of money the company gets from the sales, and *cost* is the money spent on the whole process.

	Yogurt B strategy $1	Yogurt B strategy $2
Yogurt A Strategy$1	5000,5000	15000,-5000
Yogurt A Strategy $2	-5000,15000	5000,5000

Figure 4-5 The Yogurt company strategies

A payoff change example in Yogurt company: With the developing of the new markets (or people are realizing that yogurt is a very health food), the demand on yogurt may increase by 10% every year, and then the reward may also increase by 10% every year. So the next year payoff matrix may become:

	Yogurt B strategy $1	Yogurt B strategy $2
Yogurt A Strategy$1	6000,6000	17000,-5000
Yogurt A Strategy $2	-5000,17000	6000,6000

Figure 4-6 The Yogurt company strategies

In the above examples, we can conclude that both company A and company B have to strive to lower the yogurt price; otherwise, the higher priced product would lose the market. We can also conclude that changes in the environment could change the utilities, but the dominating strategy could remain the same.

In RoboCup Simulation System, different agents have different abilities and functions. We assign a different utility function to each type of agents such that the utility function captures a discounted future payoff for that agent's actions. For fire brigades, the reward is proportional to the area of the building that the agent extinguishes. The cost represents the area of the buildings lost due to fire, as well as the time that the agent spends to reach the fire and extinguish it. The payoff is the difference between the reward and the cost. Therefore, the payoff depends on the targets the agents are working on, their location with respect to the target, and the potential losses to fire during this period.

In chaotic unpredictable multi-agent system domains, the exact utility can be difficult to predict as it is dynamic and changes with time and space. For example, for a fire brigade working on extinguishing a fire, a function of time and space represents the utility:

$$Utility=u\ (t,\ loc) \qquad (4.1)$$

Where t is the time needed to reach and extinguish the fire, *loc* is the location of the building of the building on fire.

The dependence of utility on time and space can take many forms. For example, the utility may change monotonically with time: increasing or decreasing as time passes. In general, the change in utility over time and space can be quiet complex. However, we have identified the following change patterns as useful models in our application.

1. The utility is changing monotonically with time and space. *Utility=u (t, loc).* For example, a fire in the center of the city in more likely to spread than a fire on the perimeter of the city. Similarly, fires are more likely to spread as time passes. Therefore, the utility of extinguishing a particular fire has a monotonic

future reward term that decreases monotonically with time and increases monotonically with the distance from the city center.

2. Some utilities change with time only (i.e *Utility=u (t)*). Common cases of temporal utilities include utilities for meeting a deadline, utilities for meeting a temporal order constraint, and utilities that monotonically increase or decrease with time. In RCRSS, saving injured humanoids is rewarded if the ambulance teams get to the injured humanoid out alive (death is a deadline that has to be met). In addition, there is also another monotonic reward for saving humanoids early as the health point decreases with time, and the sum of the health points of all humanoids is part of the value function.
3. Some utilities do not depend on time but depend on space (i.e. *Utility=u(loc)*). For example, in RCRSS, for civilian agents, the utility of being in a refuge is that their health point remains constant and does not deteriorate with time.
4. Some utilities are converging to a constant (i.e. $Lim_{t\to\infty} u(t,loc)=c$, where c is a constant). In RCRSS, clearing road blockades initially depends on time and space as roads leading to fires or used to reach injured civilians are more important to clear. Eventually, these priority roads are clear, and the utility for clearing more blockades becomes constant.
5. Lastly, some utility do not depend on neither time nor space (i.e. *Utility =c* where *c* is a constant).

For the Fire Brigades the available actions are: Sense, Hear, Say, Tell, Move, Extinguish, and Fill. Every cycle the agent may sense the environment using the action: Sense, exchange messages using the communication actions: Hear, Tell, and

Say, as well as one physical action that affects states in the environment that can be either Move, Fill, or Extinguish. Note that the Q value of some action changes based on the conditions of the agent and those of the environment. For example, filling the tank with water has a high utility for a fire brigade if the tank is empty and there are buildings on fire.

According to the optimal policy (2.3) maximizes the Q-values. The Fire Brigades would choose a strategy F^*, which we call ARK_FB strategy.

$$F^* = \arg \max_A Q(t, loc, A) \qquad 4.2$$

Where t is the current time, *loc* is current position at time t, A is the list of targets *(a1, a2,...ai)* which agent would take as subgoals. Subgoals refer to the set of alternatives that a decision maker has to choose from.

Nash equilibrium is a unique solution to a game-theoretic problem for that no single player wants to deviate from his or her predicted strategies while the other players keep their strategies unchanged, everyone must satisfy Nash's mutual-best-response requirement. In our model, every player gives the best response to the other players. Between time t and time *300*, the utility is the difference between the total payoff and all possible losses. Combining those factors into one Q-value function, which becomes:

$$Q(t, loc, A) = p1(t1, a1) + p2(t2, \overline{A}) - l(t, \overline{A}) \qquad 4.3$$

Where *t1= distance(s,a1)/speed* ; *t2=t1+time necessary to extinguish the a1; a1* is the first target in the list of targets (subgoals) A; $\overline{A}$ is the rest of the subgoals after *a1; p1* is the profit function for the current target *a1*; *p2* is the profit function for the rest of the

target; The difference between *p1* and *p2* are: *p2* has lower probability than *p1*; *l* is the loss function for agent when agent decide to take the first target.

Obviously $300 \geq t2 \geq t1 \geq t$. Those sub function are all stochastic function, which means when we calculate the Q-value, we consider probability as a factor.

Utility functions could dramatically change agent behavior. For example, in the Gunners' Dilemma: If we design the utility in this way, then the Gunners will be brave enough to stay and fight instead of fleeing.

		Gunner 2	
		Stay	Flee
Gunner 1	Stay	3,3	2,2
	Flee	2,2	1,1

Figure 4-7 The Gunner's dilemma

How to design the utility function for agents is a crucial technique for developers. The utility of an action is usually some function of the cost, reward, risk, and other properties of the action. Every agent gives the best response to the whole game to maximize its utility.

Basically for fire brigades, the reward is the building area of not getting any fire with the probability. The cost is the area of the building would catch fire or burn down with probability. Every cycle, agent calculates the reward and cost for itself. He knows other agents positions and targets, he will calculate his the time to get to a target and time expected to extinguish a fire. Meanwhile, he would calculate the *cost* to do this, which is the area of the building, would catch fire and burn down. Therefore, individual decisions are made in a coordinated way so that a degree of harmony is achieved.

To choose a target among a set of alternatives, the decision maker would rather prefer an alternative that maximizes expected utility to other ones. The preferences refer to the ordering relationship among alternatives in the opinion of the decision maker. The ordering relationship in a planner is an ordered set of preferences over alternative actions *A*. A utility-based agent orders elements based on the respective expected utilities taking into account the desirability (or utility) of each course of action as well as its probability. By properly setting the utility function to favor achieving common goals, a degree of coordination can be achieved.

For the Fire Brigade, the payoff is the safe area in the city, according to the in evaluation function (3.1) *V=(P + S/Sint) * sqrt(B/Bint)* only *B/Bint* is directly related to Fire Brigades, which means more fires, less payoff.

When we design the utility function, spatial and temporal factors are considered in utility assessment. The fire agent assesses the utility of an action based on a set of factors including some spatial and temporal ones as illustrated in Figure 4-8.

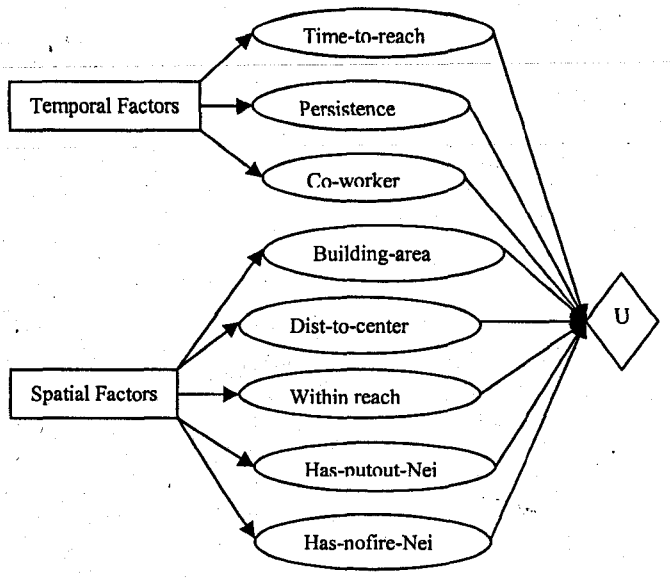

Figure 4-8 Utility Decision network for Fire Brigade agent

If a fire is put out, the building will not catch fire again. If the building is burn out, the building will not catch fire again. Based on these two facts, we can remove the extinguished buildings and burn-out from the task list, because their utilities are zeroes. If the fire brigades could cooperate to extinguish a fire building, the efficiency will improve significantly. If the agents could persistent on the job, they won't waste their effort. Quantitative and qualitative factor are considered in the utility function.

To ensure effective use of time, the estimated time to reach a building takes into account the status of the roads leading to the fire from the agent's current location. If the road is known to have been cleared, then the Euclidean distance is used in calculating the time. If the road is known to be blocked, then the time will be practically infinite. If the status of the road is unknown then the Euclidean distance is multiplied by a factor to account for potential delays. In the current implementation this factor is two. After

adding up the cost of all the possible routes, the agent chooses the fire with the maximum expected utility per unit time. Given a route, the time a Fire Brigade agent will spend to get to the fire location *Eb=Dis(loc,fire)/speed* where *Dis( )* is the weighted distance to the fire on the selected route and *speed* is the agent's speed. A lower *Eb* represents a fire that can be reached quickly.

In Figure 4-8, persistence represents the amount of time a fire agent has to spend extinguishing a particular building. If the agent's target is the same as the last cycle, this value is increased by one. The factor Co-workers represents the number of coworkers that will help the agent extinguish the fire. By parsing the messages from the central agent, fire brigade will find out how many co-workers are working on a specific building.

The larger buildings have more priority than other buildings. However, this utility must be discounted to take into account the current and future efforts need to put out larger building. Normally, larger buildings need more firefighters and take longer to extinguish than smaller buildings. Building ground area is the current reward. Neighboring area is the future reward and the probability of getting this reward is assumed to be 50%.

The distance from the target building to the center of city plays an important role in the utility assessment. First, the fires in buildings closer to the center of city can spread to more buildings than fires in corner or isolated regions. Second, central locations are usually more easily reachable by a larger number of agents than remote ones. Therefore, a building closer to the city center has more payoff than others. In fact, the distance of

the fire location to the city center R contribute a normalized value $Ea=1-R_{fire}/R$ to the utility such that the larger the Ea the higher the utility for extinguishing it becomes.

If the target building is within the fire-brigade's reach distance, it does not have to spend anytime on the road. Therefore, it makes sense to just put out the reachable fires first.

To control the spread of a fire, the fire agents would surround a cluster of buildings on fire with a rim of put-out buildings. Therefore, extinguishing a building that has put-out neighbors may help surrounding the fire cluster.

Besides time and space elements, other agents' subgoals should also be considered in utility function making. Within the multi-agent decision making framework, a decision making problem is either a single problem when only one alternative is allowed to be selected from the set of alternatives at any time, or multiple problems when several alternatives are allowed to be selected from the set of alternatives at a time. In our test domain RCRSS, the fire brigades decision making represents the second case.

In addition to the above three elements, agents' decision making is also based on whether the information is complete. Perfect information means every agent has complete information about all previous moves. While imperfect information means some or all agents have only partial information.

After deciding on subgoals, agents take the necessary actions. Every action has some preconditions *Prec(a)* and some effects *Eff(a)*. For example, FB agent's *prec(move_to(target))* include: not close enough to target, and *prec (extinguish(target))* include: close enough to the target and target is on fire. *prec(sendMsg)* is sense the environment or/and get message, etc.

For the Ambulance team, the payoff depends on the number of agents alive and their health point. The function used for ambulances in RCRSS uses $P + S/Sint$, which gives more weight to P the number of the civilians alive, over the ratio of the remaining health point total S to the initial health point $Sint$.

We define that Fire Brigades would take a strategy T^*, which we call ARK_AT strategy.

$$T^* = \arg \max_A Q(t, loc, A) \qquad 4.4$$

Where t is the current time, s is current position at time t, A is the list of targets $(a1, a2, ...ai)$ which agent would take as subgoals.

Similarly, every Ambulance Team gives the best response to the other players. Between time t and time 300, the utility is the difference between the total payoff and all possible losses. Combining those factors into one Q-value function, which becomes:

$$Q(t,s,A) = p1(t1, a1) + p2(t2, \overline{A}) - l(t, \overline{A}) \qquad 4.5$$

Where $t1$= *distance(s,a1)/speed* ; $t2$=$t1$+*time suppose to rescue next human in* $a1$; $a1$ is the first target in the list of subgoals A; $\overline{A}$ is the rest of the targets after $a1$; $p1$ is the profit function for the current target $a1$; $p2$ is the profit function for the rest of the target; The difference between $p1$ and $p2$ are: $p2$ has lower probability than $p1$; l is the loss function for agent when agent decide to take the first action.

Obviously, $300 \geq t2 \geq t1 \geq t$ and the payoff depends on the number of persons in that are rescued. Those sub function are all stochastic function, which means that when we calculate the Q-value, we consider probability as a factor.

After the AT agent makes the subgoal decision, then the AT agent follows the STRIPS algorithm to choose its action.

For the police force, the payoff is sum *blockades_rpt_FB+blockades_rpt_AM*. We define that Police Forces would take a strategy T^*, which we call ARK_PF strategy.

$$T^* = \arg\max_{A} Q(t, loc, A) \qquad 4.6$$

Where t is the current time, s is current position at time t, A is the list of reported blockades (targets) *(a1, a2,...ai)* which agents would take as subgoals.

We also let each PF agent give the best response to the other players. Between time t and time *300*, the utility is the difference between the total payoff and all possible losses. Combining those factors into one Q-value function, which becomes:

$$Q(t,s,A) = p1(t1,a1) + p2(t2,\overline{A}) - l(t,\overline{A}) \qquad 4.7$$

Where *t1= distance(s,a1)/speed* ; *t2=t1+time suppose to reach next blockade; a1* is the first target in the list of subgoals A; $\overline{A}$ is the rest of the targets after *a1; p1* is the profit function for the current target *a1*; *p2* is the profit function for the rest of the target; The difference between *p1* and *p2* are: *p2* has lower probability than *p1*; *l* is the loss function is the loss associated with other agents failing to use the road.

In summary, this chapter presents the algorithmic and conceptual elements of the multi-agent framework we propose. The RCRSS is used to exemplify how these concepts can be applied.

In a dynamic and uncertain environment, the states that justify the agents' plans may dynamically change, while the planning process is still going on or during the execution of the plan. An agent may not know the properties of the environment or other agents with certainty. The actions of an agent are non-deterministic and could span a range of possible outcomes. The outcome of an agent's performing an action might be influenced by other agents' behaviour. Their behaviour may be significantly different from what the agent may have anticipated. This chapter presents the results obtained from applying the theory developed in previous chapters to develop a multi-agent planning and coordination system for the RoboCup Rescue Simulation System (RCRSS).

5.1 Communicative actions

In order to obtain as much as possible information about the uncertain environment, the agents exchange as much information as possible, especially at the early stages of the RCRSS simulation session.

An agent may hold many desires that conflict with each other. Different agents may have conflicting goals as well and the outcome of an agent's action may be influenced by the actions of other agents. Thus an agent needs to know about other agents and decide how to collaborate with others, which makes the agent's planning process in a multi-agent system more complex than in a single agent environment.

In our approach, the communicative action is mostly an action within a collaborative multi-agent team. Each agent reports its current percepts to central agents. The central agents remove all redundancies and reports back to all agents. The agents report the

status of buildings on fire, the status of blocked roads and cries for help from injured civilians. Agents also report which tasks they have successfully accomplished. Fire agents send messages when they put-out a fire. Police agents announce when a blocked road has been cleared and ambulance agents report when they rescue an injured agent.

Agents reduce redundancy in their communications by remembering what they have previously reported. Agents also need to notify others of the action they are taking, so one of the messages they need to pass is the agent's action target. The format of the message exchanged between agents is as follows:

Current Task | *buildings on fire* |*extinguished buildings*| *blocked roads*| *cleared roads*| *building with trapped agents* | *injured agent information*

Every object in the RCRSS has a unique id. This id is 8 or 9 digits long, which takes a lot of message space. Because of the limit of the message length (256 bytes), we have to find an efficient way to represent the information. At the beginning of the simulation, we assign each object a unique short 3 digits id. A set of special characters act as the field separators. The center agent filters redundant information. Thus, the messages become shorter and easy to parse. Moreover, in some runs, messages longer than 256 bytes are generated. These messages are cut into 2 or 3 messages.

5.2 Implementation Results

To assess the performance of rational hierarchical utility-based agents, I compare with two greedy implementations. One implementation, the priority agents, is based on a set of rules for deciding on the next target building. These rules are designed to generate

priority class and each agent chooses the closest highest priority target (Tawfik et al 2004). The second implementation, the sample agents, uses a greedy selection rule to select a target from the task list. The reported results represent the average of 4 runs in each map. I used three maps Kobe, Foligno and Virtual City (VC). The use of these two heuristics for comparison is based the relatively good performance of these heuristics; the sample agents were the winner of the 2003 RoboCup competition, and the priority agents were semifinalists at the 2004 RoboCup competition. Moreover, it was possible to gain a deep understanding of the inner workings of these agents.

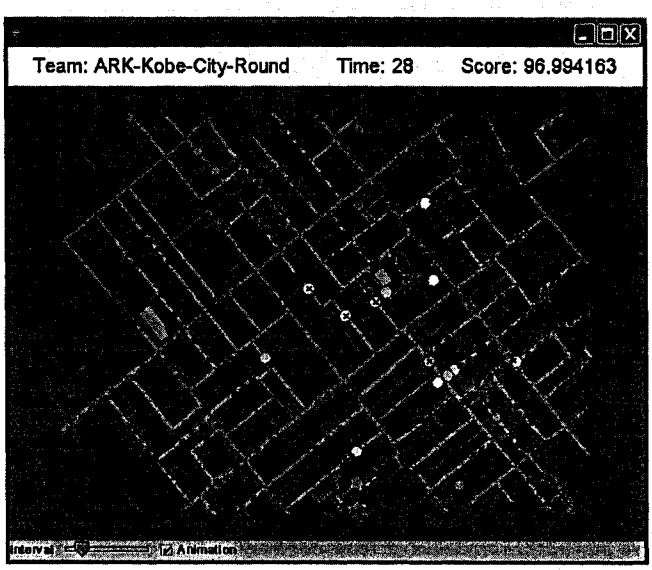

Figure 5-1 RCRSS viewer shows the Kobe map

Figure 5-1 shows the RCRSS Kobe city map. For the map of the city of Kobe, Three versions of the map are used. Map 2 and 3 has more fires, more blockades, and more injured civilians than Map 1. The results in Figures 5-3, 5-4, 5-5, 5-6, 5-7, and 5-8 show the average performance for the three variations of the Kobe map. Figure 5-3 shows the overall score as calculated using the evaluation function (3.1). From the figure, it appears

that the proposed approach outperforms the other two heuristics. Figure 5-4 gives a plausible justification of the improved performance. It is clear that the spatio-temporal agents are more successful that the other agents in extinguishing the fires in this map.

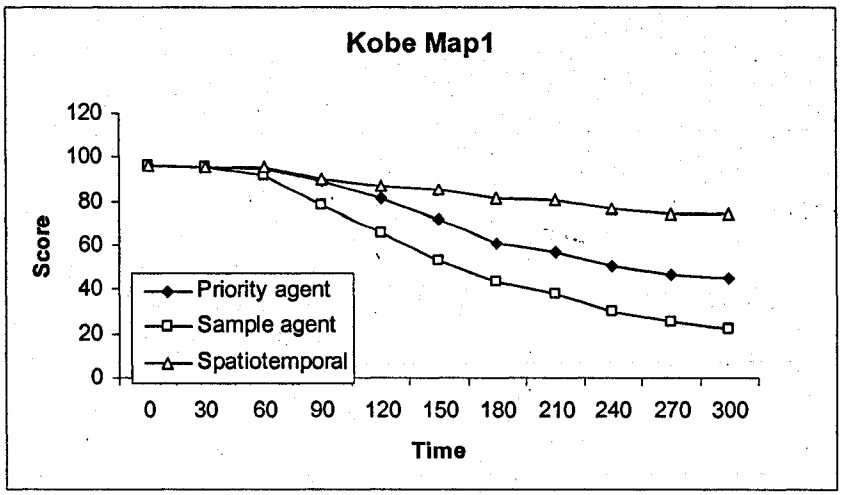

Figure 5-2 The scores of Kobe map round 1

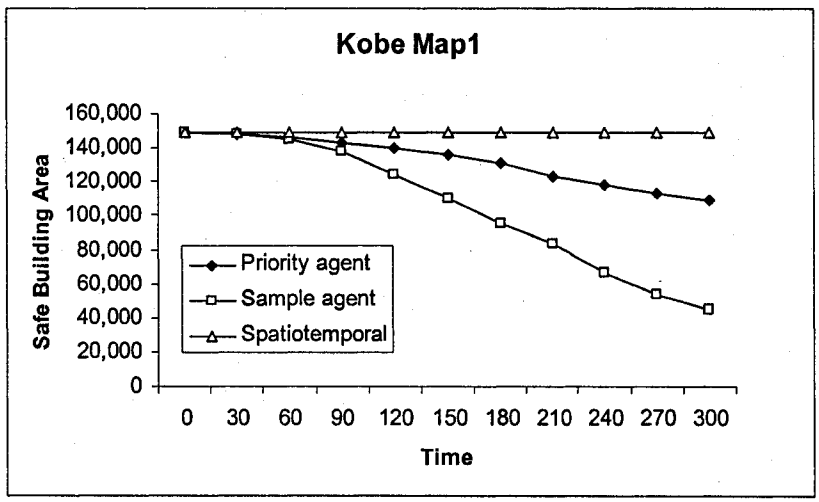

Figure 5-3 The safe building area of Kobe map round 1

The improved fire fighting performance of the spatio-temporal agents is also evident for Kobe Map2, according to Figure 5-6. However, Figure 5-5 shows that this performance improvement was not translated into a significantly better score. The reason for this

apparent discrepancy is that a higher number of death among injured agents resulted as Kobe Map 2 has more blocked roads and some traffic jams delayed ambulance agents trying to rescue injured civilians. In addition, some rescue agents were buried by the initial earthquake in Kobe Map 2. Figures 5-7 and 5-8 show improved performance in fire fighting and overall score for the spatio-temporal agents in Kobe Map 3.

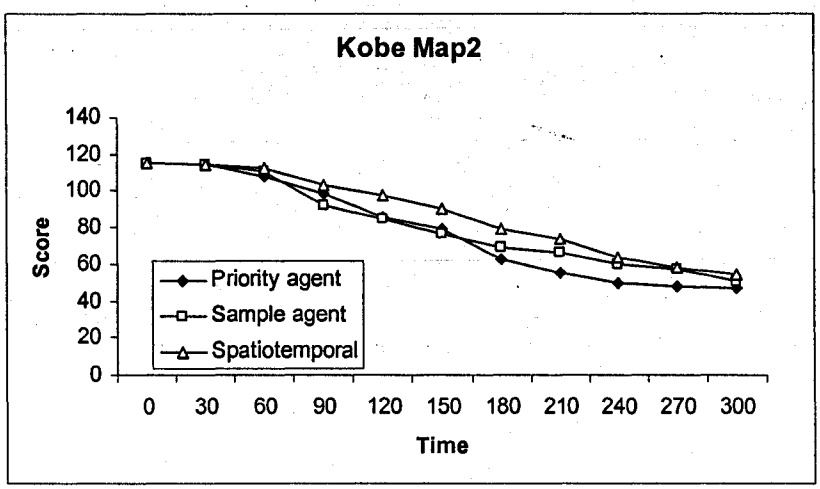

Figure 5-4 The scores of Kobe map round 2

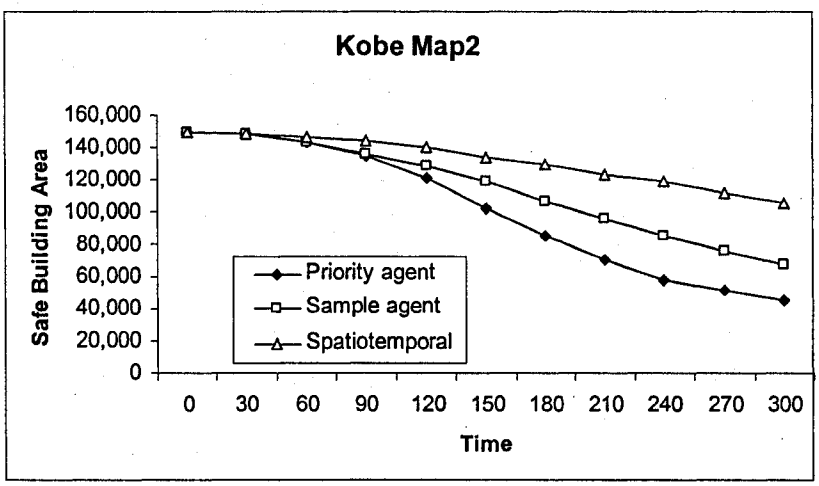

Figure 5-5 The safe building area of Kobe map round 2

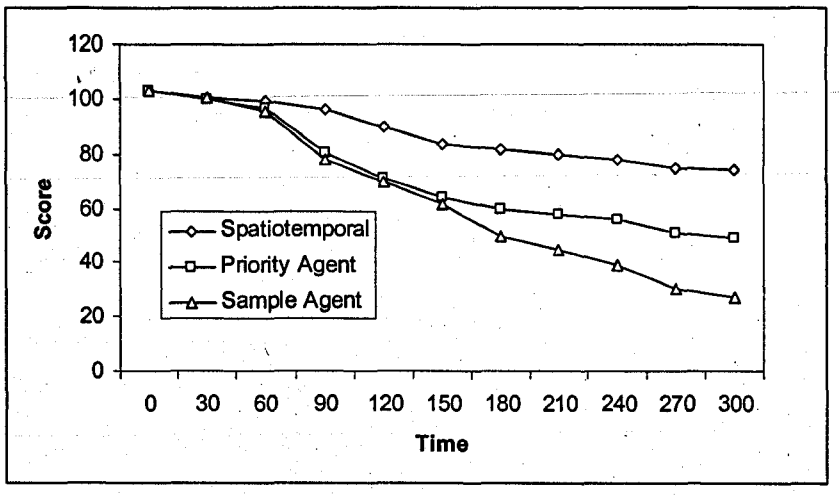

Figure 5-6 The scores of Kobe map round 3

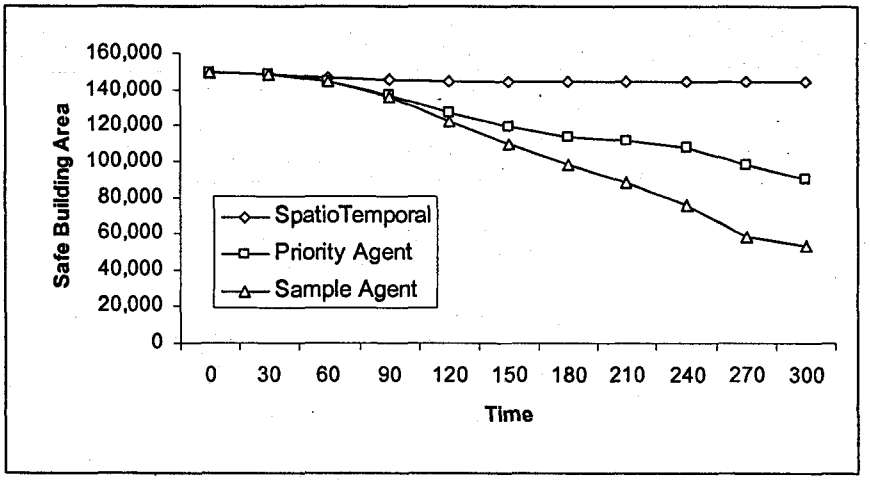

Figure 5-7 The safe building of Kobe map round 3

From performance of the agents on the three maps for Kobe, it is clear that the spatio-temporal fire brigades performance has been consistently better that the sample and the priority agents. However, it was necessary to test on other cities because some techniques may perform better on a specific map and perform poorly elsewhere.

Two additional maps are available for RCRSS: Foligno and Virtual City (VC). Foligno map have lots of blocks, the fires are difficult to extinguish in a short time. In the tests, the spatiotemporal agents did not extinguish the fires (like most implementations), but try to control the fire, and reduce the damage. For VC, there are more fires, but few blocks.

In the tests, the spatiotemporal agents have extinguished the fires in a relatively short time.

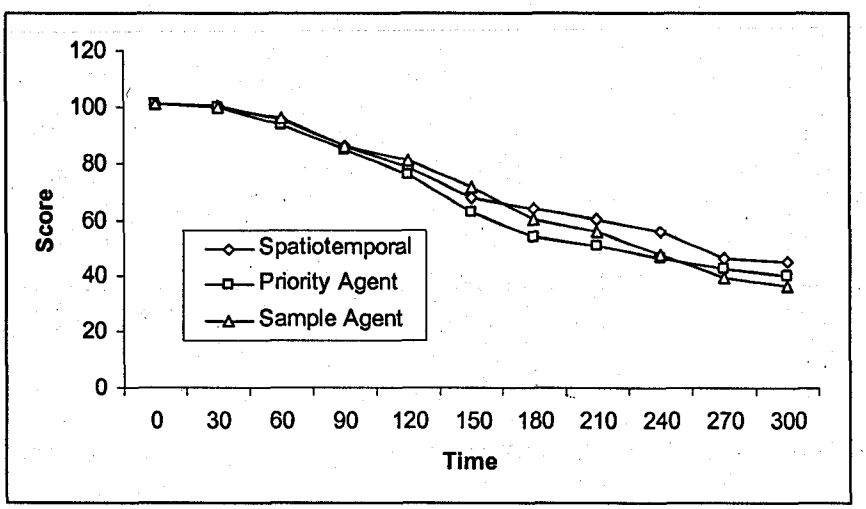

Figure 5-8 The scores of Foligno Map

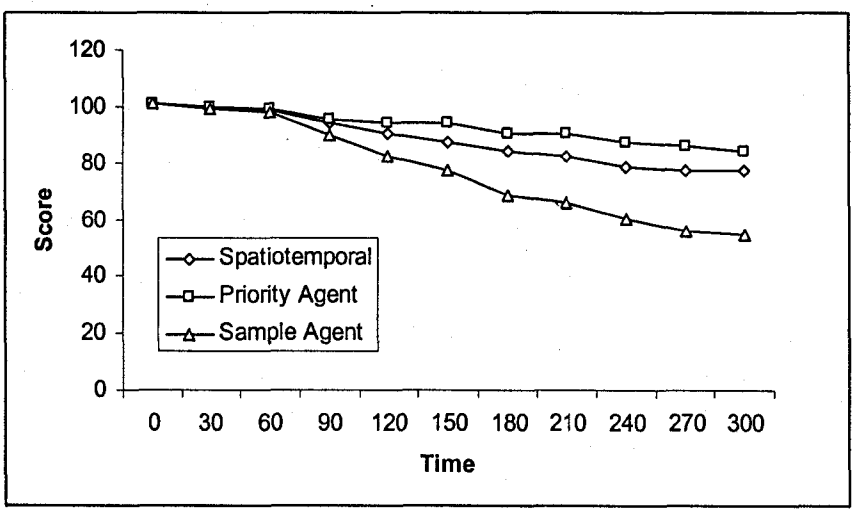

Figure 5-9 The scores of Virtual City Map

The kuwata viewer that provides continuous performance statistics during each run is not fully compatible with these two maps. The available viewer for Foligno and Virtual City only reports the overall score as shown in Figures 5-9 and 5-10. It appears that the spatio-temporal agents did slightly better in Foligno than the other two agents and did

slightly worse than the priority agents in Virtual City but still better that the sample agents.

In summary, the use of the hierarchical planning and spatio-temporal utilities proved to be beneficial and effective in multi-agent systems. In the RCRSS, it outperformed heuristics designed for this particular environment.

The approach presented in this thesis integrates elements from game theory, decision theory, BDI rational agent and Hierarchical Task Network planning. The combination uses utilities to guide action selection at each level of the Hierarchical Task Network planning and cope with changes in the environment. The main contributions of this thesis are:

- An approach for multi-agent hierarchical planning for heterogeneous and homogeneous agents.
- The use of spatio-temporal utility functions in action selection and Hierarchical Task Network (HTN) planning.
- Demonstrating a technique for mapping BDI desires and intentions between various levels in the HTN planner.
- Implementing the proposed techniques using a multi-agent simulation environment: RoboCup Rescue Simulation System. The test results are encouraging. Fire extinguishing is pretty fast and efficient. The performance of the police agents clearing road blockades has also improved. However, there has not been a significant improvement for ambulance teams. Further investigation is needed.

Our Rational Hierarchical Planning can be applied to a variety of complex dynamic multi-agent environments. For example, a group of military robots fighting against a group of enemy agents in a building have to assemble in nearby buildings, and coordinate their attack on the enemy.

6.1 Recommendations for Future Research

The framework presented here provides a basis for exploring the role of Game Theory and Decision Theory in multi-agent planning. Some of the open questions in this regards include how deal with dynamically changing payoffs in a planning context. Some special cases of dynamic payoffs have been analyzed. For example, Musacchio (2005) examines monotonically increasing payoff in the context of wireless session pricing and shows that there exists a perfect Bayesian equilibrium in this special case. However, a planning oriented formulation is necessary for multi-agent systems.

Second, in the hierarchical task network planning, the issue of commitment needed remains to be an issue worth investigating. Some types of commitment like blind commitment, single-minded commitment and open minded commitment have been proposed. However, our experience with RCRSS shows that a better solution for the commitment problem is still needed.

Bibliography

[Ambroszkiewicz and Komar 1997] S. Ambroszkiewicz, J. Komar, A Model of BDI–Agent in Game–Theoretic Framework. Model Age Workshop, 1997

[Bowling et al 2002] M. Bowling, R. Jensen, and M. Veloso, A formalization of equilibria for multi-agent planning Proceedings of the AAAI-2002 Workshop on Multi-agent Planning, August, 2002

[Cavedon and Rao 1996] L. Cavedon, A. Rao, Bringing About Rationality: Incorporating Plans Into a BDI Agent Architecture. PRICAI 1996: 601-612

[Cisternino and Simi 2000] A. Cisternino, M. Simi. Layered Reactive Planning in the IALP Teamin. in RoboCup-99: Robot Soccer World Cup III, Veloso, M., Pagello, E., Kitano, H. (Eds.), LNCS, Vol. 1856:263-273, 2000

[Claus and Boutilier 1998] C. Claus and C. Boutilier, The Dynamics of Reinforcement Learning in Cooperative Multi-agent Systems. AAAI, pp 746-752, 1998.

[Erol et al. 1994] K. Erol, J. Hendler, D. S. Nau, and R. Tsuneto, HTN Planning: Complexity and Expressivity. In Proc. of AAAI-94, pages 1123–1228, 1994.

[Erol et al. 1995] K. Erol, J. Hendler, D. S. Nau, and R. Tsuneto. A critical look at critics in htn planning. In IJCAI-95, 1995.

[Holland 1975] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975.

[Guestrin et al 2002] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In Proceedings of the 2002 AAAI Spring Symposium Series: Collaborative Learning Agents, Stanford, CA, March 2002.

[Kaelbling et al 1996] L. P. Kaelbling, M. L. Littman and A. W. Moore. Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, vol 4, pp. 237-285, 1996.

[Li and Soh 2004] X. Li and L. -K. Soh . Learning How to Plan and Instantiate a Plan in Multi-Agent Coalition Formation, to appear in Proceedings of the 2004 IEEE/WIC International Conference on Intelligent Agent Technology (IAT2004), Beijing, China, September 20-24, 133-139, 2004.

[Malone and Crowston 1990] T.W. Malone and K. Crowston. What is coordination theory and how can it help design cooperative work systems. Proceedings of the 1990 ACM conference on Computer-supported cooperative work, Los Angeles, California, United States, 1990.

[Morimoto 1999] How to Develop a RoboCup Rescue Agent for RoboCup Rescue Simulation System version 0.

[Morimoto et al 2001] T. Morimoto, K. Kono, and I. Takeuch. YabAI The first Rescue Simulation League Champion. RoboCup 2001, Team Description paper.

[Morimoto 2002] T. Morimoto. YabAPI: API to develop a RoboCup Rescue Agent in Java. http://ne.cs.uec.ac.jp/~morimoto/rescue/yabapi

[Musacchio 2005] J. Musacchio, Pricing and Flow Control in Communications Networks, PhD Dissertation, Dept. of EECS, UC Berkeley, January 2005.

[Nair et al 2003] R. Nair, M. Tambe and S. Marsella. Role and resource allocation in MAS: Role allocation and reallocation in multi-agent teams: towards a practical analysis. Proceedings of the second international joint conference on Autonomous agents and multi-agent systems, July 2003.

[Rao and Georgeff 1991] A. Rao, and M. Georgeff. Modeling rational agents within a BDI-architecture. In Second International Conference on the Principles of Knowledge Representation and Reasoning, 1991

[Russell and Norvig 2003] S.J. Russell and P. Norvig. Artificial Intelligence: a modern approach (2nd Edition). Prentice-Hall, 2003.

[Sen and Sekaran 1996] S. Sen and M. Sekaran. Multi-agent Coordination with Learning Classifier Systems. Proceedings of the IJCAI Workshop on Adaptation and Learning in Multi-Agent Systems, vol 1042, pp 218--233. editor, Gerhard Wei and Sandip Sen,1996.

[Stone and Veloso 1999] P. Stone and M. Veloso. Task Decomposition, Dynamic Role Assignment, and Low Bandwitdth Communication for Real Time Strategic Teamwork. Artificial Intelligence, 1999.

[Sardina et al 2006] S. Sardina, L. de Silva and L. Padgham. Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. Proceedings of the 5th Autonomous Agents and Multi-Agent Systems Conference (AAMAS-2006).

[Segerberg 1989] K. Segerberg. Bringing it about. Journal of Phil. Logic, 18, 1989

[Sycara 1998] K. Sycara. Multi-agent System. AI Magazine 19(2), 1998.

[Tan 1997] M. Tan. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Learning. Readings in Agents, Morgan Kaufmann, San Francisco, CA, USA, 487—494, 1997.

[Tawfik et al 2004] A. Y. Tawfik, Z. Ibrahim, D. Liang, R. Price, L. Qin, and Z. Wu. ARK Team Description: A Change-based Approach to Urban Rescue, Proceedings of Robocup: The 8th RoboCup International Symposium. Lisbon, Portugal, 2004.

[Verbeek et al, 2004] B. Verbeek and C. Morris. Game Theory and Ethics, The Stanford Encyclopedia of Philosophy (Winter 2004 Edition), Edward N. Zalta (ed.), URL = http://plato.stanford.edu/archives/win2004/entries/game-ethics/

[Vlassis 2003] N. Vlassis. A Concise Introduction to Multi-agent Systems and Distributed AI. Informatics Institute, University of Amsterdam, September 2003.

[von Neumann and Morgenstern 1947] J. von Neumann and O. Morgenstern. Theory of Games and Economic Behavior. Princeton University Press, 1947.

[White 1969] D. J. White Decision Theory. Chicago: Aldine Pub. Co. 1969.

[Winikoff et al 2001] M. Winikoff, L. Padgham, and J. Harland. The concepts are described in Simplifying the Development of Intelligent Agents. In proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AI'01), Adelaide, 2001.

[Wooldridge 2002] M. Wooldridge. Introduction to Multi-agent Systems. John Wiley and Sons, Chichester, England

VITAE AUCTORIS

Dong Liang joined the Temporal Inference Project (TIP) in April 2002. Since then he has been working on many aspects of Multi-agent planning. Along with his teammates in the research groups, he has participated in the Robocup Championship in 2004 in Lisbon, Portugal. He is now working in Chicago, U.S.A. after the completion of this thesis.