The Pennsylvania State University

The Graduate School

College of Information Sciences and Technology

AGENT-BASED COLLABORATIVE PLAN ADAPTATION WITH RESOURCE CONSTRAINTS

A Thesis in

Information Sciences and Technology

by

Rui Wang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2006

The thesis of Rui Wang was reviewed and approved* by the following:

John Yen
University Professor of Information Sciences and Technology
Thesis Advisor
Chair of Committee

Lee Giles
David Reese Professor of Information Sciences and Technology

Peng Liu
Associate Professor of Information Sciences and Technology

Tracy Mullen
Assistant Professor of Information Sciences and Technology

Raj Acharya
Professor of Computer Science and Engineering

Joseph M. Lambert
Senior Associate Dean
Associate Professor of Information Sciences and Technology
Chair, Graduate Programs Advisory Committee
Head of the College of Information Sciences and Technology

*Signatures are on file in the Graduate School.

ABSTRACT

One important lesson learned from the crisis of Hurricane Katrina is that preplanned allocation of scarce resources (e.g., helicopters, vehicles, medical equipment, etc.) must be dynamically modified to adapt to the changing situation. Furthermore, multiple response teams, which own different resources, need to collaborate to determine tradeoffs among competing resource needs to reallocate such resources. The challenge of this collaborative resource reallocation is further complicated by the fact that each team may not have complete information about other teams' resources and the utility of their tasks. This limitation makes it difficult to assess tradeoffs among resource needs of different teams.

In order to solve such problems, this research developed team-based software intelligent agents to collaboratively adapt existing plans for resource use by consistently reallocating the limited resources among distributed tasks. Based on an innovative multi-agent teamwork model called R-CAST (RPD-enabled Collaborative Agents for Simulating Teamwork), this research developed a framework for collaborative plan adaptation under resource constraints. The framework addresses the challenges mentioned above in three ways. First, it extends R-CAST with explicit representation of resources and related reasoning algorithms about resources. Second, it uses a combinatorial auction mechanism to enable agents to exchange utility information regarding competing needs of bundled resources, so that agents can reason about resource tradeoffs for effective reallocation of scarce resources. Third, the framework implements an algorithm for an agent to assess the opportunity cost of offering a resource bundle that

has already been assigned to a task. The algorithm considers alternative ways to accomplish the task, the utility of such alternatives, and associated costs for obtaining required resources.

In summary, the work presented in this thesis facilitates distributed teams to reason about tradeoffs among competing requests for resource bundles by exchanging relevant information through combinatorial auctions. Experimental results have suggested that this research can significantly improve the utilization of limited resources in adapting plans to the dynamic situation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am deeply indebted to my thesis advisor, Dr. John Yen, for support, encouragement, and guidance in the past five years. It is my fortune to have Dr. Yen as my mentor and I could not have achieved this accomplishment without so many helps from him.

I am also grateful to members of my doctoral committee, Dr. Tracy Mullen, Dr. Lee Giles, Dr. Peng Liu and Dr. Raj Acharya, for their efforts and constructive suggestions on my doctoral studies.

My fellow colleagues in the intelligent agent lab provided inspiration through our discussions. I thank Dr. Xiaocong Fan for everything that I consulted him. I am also thankful to fellow graduate students: Shuang Sun, Viswannath Avasarala, Cong Chen, Kaivan Kamali, Guruprasad Airy, Shizhuo Zhu, Bingjun Sun, and Po-Chun Chen. I thank my friends whose names are not listed here as well, for their helps on my work and life.

Finally, the greatest debt I owe is to my parents and siblings. They support me at every moment in my life, no matter how far away I am from them.

Chapter 1

Introduction

1.1 Problem Statement

The crisis of meeting the emergency needs caused by Hurricane Katrina reveals the importance of consistently adapting preplanned tasks and handling emergencies with scarce resources under a dynamic, real-time situation. Due to the naturally evolving characteristics of such a situation, existing plans for completing rescue tasks are subject to unexpected changes during the execution process. Thus, those plans must be modified in an appropriate and timely way to keep them from failing. Also, certain changes during the response to such an occurrence as Katrina trigger additional tasks, which may generate new requests for resources.

Since the available resources are always limited in such emergency situations, tradeoffs have to be made judiciously in order to switch resources from a previously existing task to a new task and thus get a higher total resource utility.

Similar to the hurricane relief domain, many other real-world domains (e.g., emergency first response, military actions, and anti-terrorism, etc.) also require plan adaptation, especially the adaptation of resource allocation, in order to handle changes in a timely and effective way. Such adaptation is necessary because conflicts often arise among different tasks regarding their resource needs since the number of resources needed may exceed the number available. How to satisfactorily resolve conflicts when

different tasks compete for the same resources becomes the key problem that needs to be addressed. This problem is further complicated when scarce resources are owned by distributed teams, each of which needs to allocate resources among the tasks assigned to them, because each team has only limited information about the other teams' resources and situations.

There are two common characteristics in such domains that have prompted this research on plan adaptation. First, the situation is always changing, which makes it possible that either emerging tasks are triggering additional resource requests or some previously available resources have become unavailable. Second, there are only limited resources, which make it infeasible to satisfy every task's resource requirement and complete all those tasks. People usually have to make tradeoffs when selecting tasks so as to satisfy the resource requirement. In other words, those tasks should be selected in a way that optimizes the expected utility of the limited resources.

When multi-agent systems (MAS) applications are deployed in such uncertain, dynamic environments, those issues also need to be carefully considered. Although centralized mechanisms to solve the optimization problem can provide quality-guaranteed solutions, they are often not scalable and can create a "bottleneck" problem. Distributed solutions can be more flexible and more reliable, although these benefits are obtained through extra efforts in coordinating multiple agents. Due to the fact that each agent has only partial knowledge about others' resources and states, the coordination problem becomes more complicated. This thesis therefore focuses on designing and implementing a market-based mechanism to dynamically optimize the reallocation of

limited resources among distributed teams by making tradeoffs among competing resource needs based on exchanging utility-based price information.

1.2 Motivation

Problems with planning have had a long history in the field of Artificial Intelligence (AI). Traditional approaches to AI planning problems usually assume that the context is static and predictable in order to simplify the planning process [1]. Unfortunately, the real world is changing and unpredictable to a certain degree. If the planner neglects uncertainties in the real world, it will not be able to anticipate potential risks and to make preparation for handling those problems. As a result, the planning system will run into a dilemma when unexpected changes occur and fail to solve emerging problems to achieve the desired goal. Thus, recently there has been an increasing need for developing adaptive planning systems that work in a complex, dynamically changing environment, especially in crises of the proportion of Hurricane Katrina. The planning problem is no longer a one-time issue since static plans are not the ultimate goal of successful planning systems, and planners need to adapt current plans to the change online. Simply speaking, plan adaptation means modifying or repairing an old plan so it can solve a new problem by reusing part of the old plan. The approach of adapting previously successful plans is an attractive paradigm for the following two reasons. First, cognitive studies show that human experts depend on knowledge of past problems and solutions—which can be called experience—for good problem-solving performance. Second, arguments about computational complexity show that reasoning

from first principles requires time exponential to the size of the problem. Systems that reuse old solutions can potentially avoid this problem by solving a smaller problem: that of adapting a previous solution to the current task. Actually, because many new problem-solving situations closely resemble previous situations, there may be an advantage to using the principles of past successes to solve new problems [2].

Agent-based systems technology has been developed in AI as a new paradigm for conceptualizing, designing, and implementing software systems. There are numerous definitions for “intelligent agents” in AI communities. Here we adopt the definition provided by the Intelligence Software Agents Lab at Carnegie Mellon University to generally describe what an agent and a multi-agent system are in this research. Based on this definition, agents are sophisticated computer programs that act autonomously on behalf of their users, across open and distributed environments, to solve a growing number of complex problems. Increasingly, however, applications require multiple agents that can work together. A multi-agent system (MAS) is a loosely coupled network of software agents that interact to solve problems that are beyond the individual capacities or knowledge of each problem solver [3].

Many researchers have addressed the plan adaptation problem for a single agent [4, 5, 2]. However, large-scale adaptive planning problems in many real-world domains (e.g., hurricane relief, homeland security, military, etc.) usually require multiple agents including human planners to collaborate on a solution. For example, in the military domain, a mission plan may involve hundreds or thousands of units being deployed simultaneously. Since the real-world situation changes frequently, a huge amount of

information regarding the changes needs to be communicated among those units and each must process the exchanged information about the changes in a timely manner to adapt its own subplan correctly, and to communicate the result in order not to corrupt others' tasks. If this process is performed by humans, it is usually time consuming to produce the result and incurs huge cognitive loads. The key issue is that it distracts the human units from focusing on the big picture and divides their attention between the high-level tasks (at which humans are proficient) and the low-level information exchange and analysis (which are intelligent agents' advantages). Ideally, when an existing plan has to be adapted for responding to the changes, the expertise of different agents (both software agents and human planners) needs to be synthesized to enhance the overall performance. Usually, intelligent agents can gather and preprocess information needed in the plan adaptation process more quickly and accurately. Human experts may have a complete overview of the whole scenario from the information provided by intelligent agents and are able to generate better solutions based on their experiences and advanced reasoning abilities. Using such a collaborative method to solve the plan adaptation problem can speed up the process, reduce human labor, improve efficiency, and make it possible to handle information-overloaded tasks. Obviously, how intelligent agents can cooperate together effectively in this process is a challenging issue to address.

Resource is another important issue motivating this research. Usually a plan has its specific resource requirement in addition to preconditions. Only when both the resource requirement and preconditions are satisfied can the plan be executed successfully. However, under a situation in which the total available resources are limited, it may be not possible to satisfy the resource requirement for all tasks at the same time.

The scarce resources should then be allocated to selected tasks in a way that maximizes the expected utility. Also, the allocation of resources should not be fixed but dynamic in order to handle the case when there are emerging tasks or changes in the resource status of existing tasks. Consistently reallocating resources among distributed tasks is considered a significant part of the plan adaptation process, and is the focus of this research as well.

The scarcity of resources necessitates tradeoffs, and tradeoffs result in an *opportunity cost*. While the cost of a good or service often is thought of in monetary terms, the opportunity cost of a decision is based on what must be given up (the next best alternative) as a result of the decision. Any decision that involves a choice between two or more options has an opportunity cost. This research addresses the issue of opportunity cost when an agent is making a decision on whether to offer resources assigned to its own task by considering tradeoffs among alternative ways to accomplish the task.

From the encyclopedia of economics edited by David Henderson [87], in economics the term “opportunity cost” of a resource means the value of the next-highest-valued alternative use of that resource. If, for example, you spend time and money going to a movie, you cannot spend that time at home reading a book, and you cannot spend the money on something else. If your next-best alternative to seeing the movie is reading the book, then the opportunity cost of seeing the movie is the money spent plus the pleasure you forgo by not reading the book. The word *opportunity* in *opportunity cost* is actually redundant. The cost of using something is already the value of the highest-valued alternative use. But as contract lawyers and airplane pilots know, redundancy can be a

virtue. In this case, its virtue is to remind us that the cost of using a resource arises from the value of what it could be used for instead.

In recent years, developing effective multi-agent systems to solve complex problems in dynamic, real-world domains has drawn more and more AI researchers' attention. My research followed this tendency and was motivated to enable multiple software agents to collaboratively adapt existing plans by evaluating changes, sharing utility information, and optimizing the resource allocation. As a result, the allocation of limited resources is not just done at the beginning stage and not changed throughout the plan execution. Instead, resources are consistently being reallocated among tasks to adapt to changes arising from either the environment or the tasks themselves. The objective of dynamic resource reallocation is not only to satisfy some tasks' resource requirement but also to optimize the resource allocation and maximize the expected utility from a global perspective.

The following paragraphs give a high-level description of how multiple agents collaborate with each other to adapt existing plans online regarding the resource constraint. Whenever an agent detects a change, first it analyzes the impact of the change and decides whether to let its teammates know this information. Then, if the change is relevant, the agents exchange information about the impact of this change on each individual plan (from the local view). Based on more complete information of how the change affects the total plans (from the global view), finally the agents collaboratively generate a response to the change, trying to optimize the global performance.

Section **1.1** briefly describes the problem of adapting preplanned tasks with limited resources to changes in various dynamic, real-time domains. Here an overview of the collaborative plan adaptation in a hurricane relief scenario is presented to further illustrate the problem and how multiple agents can collaborate to solve it (Figure **1.1**). Three kinds of tasks are involved: 1) delivering foods to a large group people who have been isolated in a flooded area (*DeliverFood_Plan*); 2) transferring sand bags and other materials to a specific place in order to fix a broken levee (*FixLevee_Plan*); and 3) rescuing a few persons from a dangerous place that will soon be flooded (*RescuePeople_Plan*). Each task has a predefined plan template, which specifies the plan's goal, resource requirement, preconditions, and the process structure, etc. It is possible that multiple plan instances may be triggered from the same plan template by binding plan variables with different values. For example, the plan template *RescuePeople_Plan* has two instances. Each plan instance can be executed by a single agent or by a team of agents. In addition to agents who are executing those plan instances, each plan has a coordinator agent, who is responsible for gathering information about detected changes, analyzing the impact of changes, collaborating with other coordinators, and adapting the assigned plan instance. Since my research focuses on changes that impact a plan's resource requirement, the coordinator agent can be called a resource coordinator. And the collaboration studied here is the one between multiple resource coordinators rather than between agents who are executing a plan in a teamwork context[1]. A resource coordinator generates requests for missing resources in its designated plan,

[1] In the rest of this thesis, if not designated otherwise, an agent means an agent assuming the role of resource coordinator.

handles resource requests from other coordinators, evaluates tradeoffs between keeping requested resources being allocated to its own plan and transferring them to the requesters, and makes the allocation decision based on the principle of maximizing the total expected utility of all plan instances.

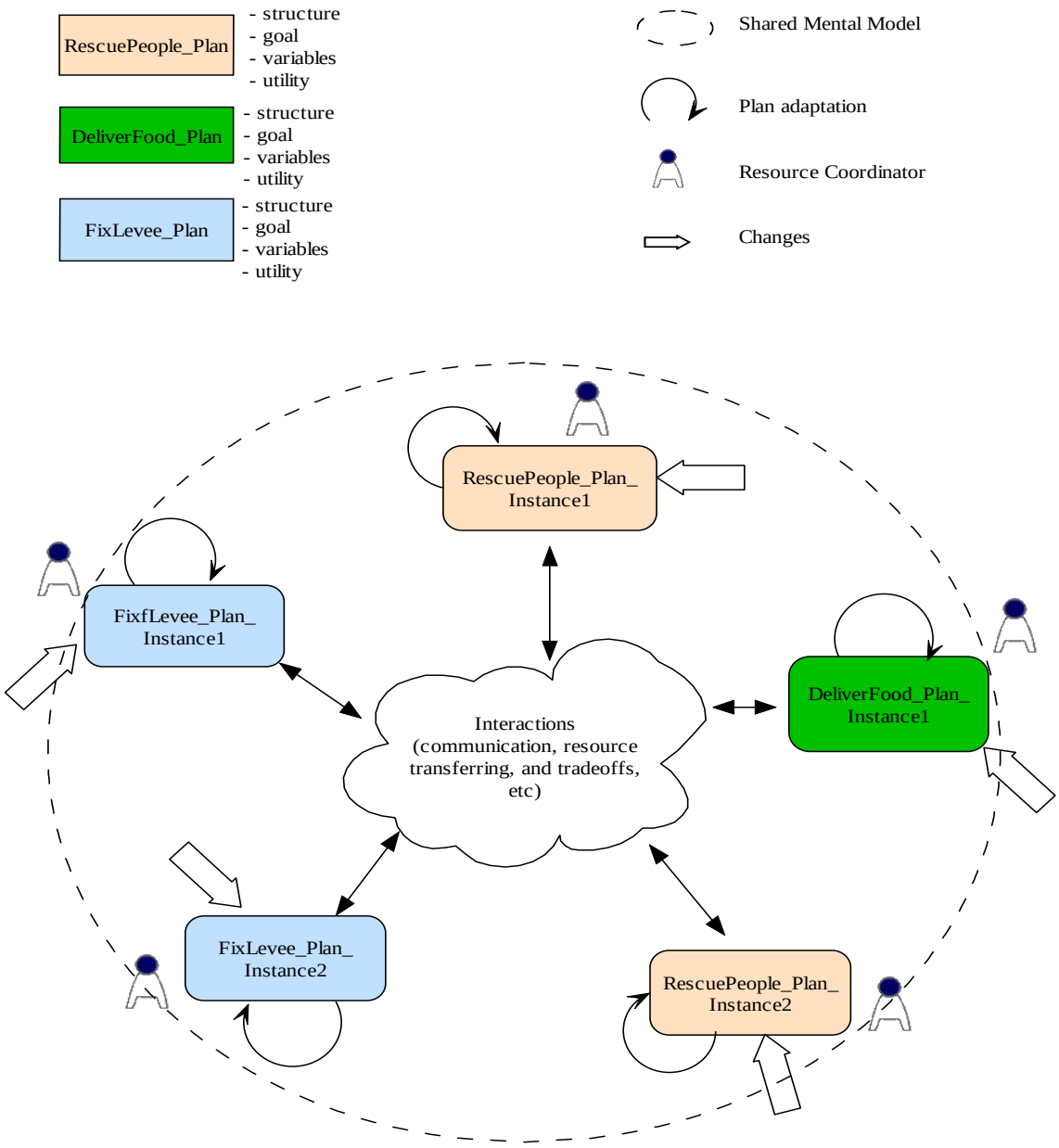

Figure **1.1**: The Overview of Collaborative Plan Adaptation under a Dynamic Environment

There are many interactions between resource coordinators involved in the whole process. Communication is certainly a key issue since different coordinators need to share relevant information such as detected changes, the impact of changes, resource availability, and utility, etc. When conflicts arise due to multiple tasks competing for limited resources at the same time, coordinators have to interact with each other to make tradeoffs in selecting which resource requests to satisfy. For example, if both a *RescuePeople_Plan* instance *P1* and a *DeliverFood_Plan* instance *P2* are competing for a helicopter as the needed resource, their resource coordinators should work together to make a tradeoff to solve the conflict. In this specific case, the tradeoff is made to allocate the helicopter to *P1* instead of *P2* since the expected utility of rescuing people is much greater than the expected utility of delivering foods so that the total utility is maximized. Usually after the decision of allocating resources is made, coordinators who currently own the requested resources start a process to transfer them to where the requesting agent or team can access those resources.

1.3 Research Scope

The scope of collaborative plan adaptation is vast. It involves both traditional AI planning techniques (e.g., adaptive planning, knowledge representation, and plan execution, etc.) and multi-agent system features (e.g., team structure, shared mental model, and coordination, etc.). My research focuses on creating distributed intelligent computational systems that adapt existing plans online to solve potential resource conflicts caused by the changes in the situation, whereas re-planning from first principles

is not part of this research. Existing plans are formed offline by either human experts or planning systems. Advanced techniques such as an agent-based teamwork model, market-based mechanism, and optimization algorithms are integrated into the proposed framework to solve the plan adaptation problem with resource constraints in a distributed manner. As mentioned before, coordination is a key issue to be addressed in the adaptation process. My research adopts the approach to solve the issue of multi-agent coordination by leveraging on a multi-agent architecture from the teamwork perspective.

In order to make an agent-based collaborative plan adaptation applicable in real complex situations, the following steps need to be covered. First, theories of collaboration or teamwork should be extended to include the plan manipulation. Different from many other collaborative works, which focused on behaviors or actions, the process of collaborative plan adaptation is much related to coordination among multiple cognitive processes, which are highly interdependent. In the human case, it is mainly a collection of mental activities from different people (often domain experts). And the result of the adaptive planning process will decide the following collaborative behaviors and actions. The plan adaptation also involves the issue of making the best decision under the current and projected circumstances. Thus, existing theories need to consider this aspect and new features or new theories may need to be developed to consolidate the theoretical background. Second, due to the complexity of the large-scale plan adaptation problem under the dynamic world, intelligent agents need to interact with the human to make a better decision among a list of candidate options, or to get more specific information that is out of their local knowledge base. Thus, the role of the human should be included in

the teamwork model in order to enhance the overall performance for solving a complex, realistic task. In other words, a user-friendly interface should be provided for human experts to interact with agents in a convenient and effective way in solving the adaptation problem. Third, after initial theories, models or designs, so called prototypes, are developed, we need to verify and modify them in an iterative procedure. Directly implementing them into the real world is risky, costly, and uncontrollable. Thus, we should test our ideas in a scaled world that simulates the real environment. Basically, we need to develop simulators in which we can easily test certain variables while constraining other irrelevant ones at much lower costs. In my research, the last two steps are included but the first step is not in the scope.

Below I list five assumptions under which this research is carried out. The proposed framework, algorithms and experimental results are valid if and only if those assumptions hold.

1. The total resources are limited, which means there are always more resource needs than available resources; otherwise there will not be conflicts in competing resources.
2. One task may have multiple methods to accomplish its goal while different methods have different utilities and different resource requirements.
3. Different tasks have reserved their own resources at the beginning stage and the knowledge about such private resources is not shared with other tasks.
4. Utility information for resources and tasks is distributed among different resource coordinators initially.

5. There are always communication costs when a coordinator needs to share certain information with other teammates.

1.4 Research Questions

The discussion above indicates the key issue in collaborative plan adaptation, which is how a team of agents can collaborate effectively in handling the changes. Interaction between agents occurs because agents solve subproblems that are interdependent, either through competing for resources or through relationships among the subproblems. Due to the complexity of solving problems in real-world domains, it is usually not possible to generate a perfect decomposition so that the computational requirements for effectively solving each subproblem and the location of information, expertise, processing, and communication resources in the agent network are totally independent [6]. This lack of a perfect fit often leads to a situation where there may be insufficient local information or resources for an agent to completely or accurately solve its assigned subproblems through its own processing. Plan adaptation in real-time requires each agent to handle the changes timely and correctly. Otherwise, even after a global plan has been developed, it may become inapplicable under the changed situation. Information sharing is the most useful way for multiple agents to coordinate their processes. However, communicating every piece of information is not an efficient solution because it involves extra costs when the information is not necessary. Resource constraint is another issue that cannot be neglected in the coordination process. Since the total resources are limited, conflicts usually arise due to different agents competing for

the same resource at the same time when they are adapting their own plans to the changes. Moreover, these conflicts should not be solved simply by choosing the winner randomly or based on rules (e.g., first in first serve, priority, etc). Instead resource conflicts should be resolved in a way whereby the global utility is maximized (i.e., the limited resources can be fully utilized).

The research presented in this thesis focuses on two research questions: 1) how an agent can efficiently share information about the change and its impact on other teammates in a timely way and 2) how a team of agents can effectively collaborate on an optimal solution for resolving resource conflicts in adapting existing plans.

1.5 The Accomplishment of This Research

In this research, I studied the problem of adapting distributed plans to changes online and provided a solution based on dynamic resource allocation to solve conflicts between competing needs for scarce resources in an efficient way. First, a theoretical framework for a collaborative plan adaptation with resource constraints was proposed in this research. This framework consists of different functional components, which are integrated to facilitate the whole process of adapting plans to resource-related changes. With the guidance of such a framework, I built the resource coordinator agents on top of a novel multi-agent architecture called R-CAST (RPD-enabled Collaborative Agents for Simulating Teamwork). In addition to existing modules (e.g., task manager, active knowledgebase, information manager, etc.) in R-CAST, I extended it by implementing a resource manager module, which is used to monitor resource status, generate resource

requests, and handle incoming resource requests. Last, I developed a market-based mechanism for dynamically optimizing the allocation of limited resources, and implemented an algorithm for a resource coordinator to select the right one from alternative methods in a task based on utility information and other tasks' resource needs.

In summary, this research makes four major contributions to the field of plan adaptation.

1. Proposed a theoretical framework for collaborative plan adaptation with resource constraint. Different functional components are integrated within this framework to support the adaptation process in a distributed way. With this framework, resource coordinators can share relevant information (such as utility, resource status, resource requirement, etc.), and collaborate with each other to reach a global agreement on how to allocate resources to achieve the maximum utility. The framework enables a team of agents to collaboratively resolve conflicts among competing resource needs to adapt plans to the changing environment, based on assessing tradeoffs among those competing resource needs: 1) It uses a combinatorial auction for the team of agents to exchange information about *opportunity cost*. 2) An agent assesses its *opportunity cost* for offering a resource assigned to a task T_i by considering the tradeoff between alternative ways to accomplish T_i.
2. Built resource coordinator agents on top of a novel multi-agent architecture R-CAST with extended functions. Each resource coordinator is responsible

for managing resources for its assigned task[2], and interacting with other resource coordinators in the adaptation process. R-CAST is extended with explicit representation of resources and related reasoning algorithms to generate resource needs and requests.

3. Developed an agent-based auction mechanism to enable resource coordinators to exchange utility information on their own tasks regarding competing resource needs. Thus, the allocation of limited resources is optimized and gives out the maximum global utility.
4. Designed and implemented an algorithm for a resource coordinator to evaluate alternative methods in a task and select the appropriate one based on utility information and resource needs of other coordinators. Usually, there are multiple methods to complete the same task, and different methods have different resource requirements. The algorithm enables a resource coordinator to assess tradeoffs between those alternative methods considering the impacts of selecting a method on both its own assigned task and other tasks.

Experimental results show that with the collaboration of resource coordinators, limited resources can be dynamically allocated to distributed tasks in adapting to resource-related changes. The market-based optimization mechanism makes it possible to fully utilize those limited resources and achieves a maximized global utility. And the

[2] A resource coordinator's assigned task is not the task that the coordinator needs to execute, which is the ordinary meaning of "assigned task." In this thesis, it actually means that the task is being managed by a resource coordinator. The resource coordinator is responsible for satisfying the task's resource needs in the adaptation process while the execution of the task is done by other parties, either a single agent or a team of agents.

method selection algorithm provides the flexibility and the optimality for completing a task from a broad view.

1.6 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 introduces the background knowledge of distributed planning, plan adaptation, resource constraint reasoning, agent-based teamwork, and market-based optimization, and reviews related works. Chapter 3 presents a framework for collaborative plan adaptation (CPA) under resource constraints and explains how it leverages on a novel multiagent architecture R-CAST. We elaborate the market-based approach to optimize resource reallocation during the adaptation process in Chapter 4. Chapter 5 explains how to select the appropriate method to complete a task if there are multiple options in order to maximize the expected utility based on more complete information. Experimental results are shown and analyzed in Chapter 6. Finally, Chapter 7 concludes the thesis.

Chapter 2

Related Work

2.1 Overview

This chapter provides a survey on related research and technologies for collaborative plan adaptation with resource constraint. I classify related work into six categories and review them separately in the following sections, which are planning and plan representation, resource constraint reasoning and scheduling, plan adaptation, distributed planning, agent-based teamwork, and market-based agents. In each category, I select research work that best represent studies in this area to describe issues they address, their contributions and their limitations. Table **2-1** provides an overview of related research in those six categories and explains the relevance to this research.

Table **2-1**: Overview of Related Work

Category	Relevance to this research
Planning and Plan Representation	Explain what a planning problem is and the way to represent a plan
Resource Constraint Reasoning and Scheduling	Provide foundations for considering plans with resource constraint
Plan Adaptation	Provide foundations for planning and execution under an uncertain, dynamic environment
Distributed Planning	Provide the paradigm of solving complex problems with the collective effort of distributed knowledge, information and capabilities.
Agent-based Teamwork	Generate theoretical foundations for a team of agents collaborating on solving problems
Market-based Agents	Enable efficient resource allocation and optimization

2.2 Planning and Plan Representation

Automatic planning has been a persistent topic in the AI field for over 40 years. Generally speaking, planning is the problem of generating a course of actions that can reach a desired goal state from a given initial state. Most classical planning systems are given a representation for possible actions that can be executed in the environment, describing both the conditions required to trigger the action and the effects on the environment when the action is taken. Typically, they have the following additional assumptions in order to simplify the planning problem [1, 8]:

1. The environment can be represented in a sequence of discrete states, which are represented using a logical language to help the planner understand. Only those possible actions will affect the environment, which means there are no exogenous events that change the environment.
2. The possible actions are represented in STRIPS [7] or similar languages. The actions' preconditions are represented in the same way as define states, and the actions' effects are a set of terms to be added or deleted from the terms used to describe a state. An example of STRIPS is shown in Figure **2.1**
3. The goal is a condition in a desired state, which can be encoded by a logical sentence.
4. The planners return a plan in the form of a sequence of actions, either totally or partially ordered, with constraints specified.

The operator describes an action with three elements: ▪ A precondition formula ▪ An add-list ▪ A delete-list	Pickup(x) ▪ precondition: ONTABLE(x)∧HANDEMPTY∧CLEAR(x) ▪ add-list: HOLDING(x) ▪ delete-list: ONTABLE(x),HANDCLEAR,CLEAR(x)

Figure **2.1**: An Example of STRIPS Operator Representation[14]

Those assumptions are adopted in a large family in AI planning because they are powerful and seem reasonable. A wide range of behaviors can be captured with STRIPS-style operators, which allow a planner to perform backward chaining since given a goal one can set the preconditions required for an action or a sequence of actions as subgoals to achieve the main goal. In addition the logical representation allows a plan to be easily proved correct or not.

Tate's NONLIN, which is one of the most well-known nonlinear planners [9, 10] represents a plan as a schema with the following components:

1. A set of *steps*;
2. A set of anticipated *effects* of those steps;

3. A set of *links* relating effects to the steps that produce and consume them (record the purpose of the steps in the plan);
4. A set of *variable bindings* instantiating the operator schema;
5. A *partial ordering* on the steps;
6. A set of *open conditions*, i.e., unestablished goals;
7. A set of *unsafe links*, i.e., links the conditions of which could be falsified by other effects in the plan.

A plan is *complete* when it contains no open conditions and no unsafe links. Many subsequent planners [11**,** 12**,** 13] derived their own plan representations in a similar way from the representation used in NONLIN.

The STRIPS action representation does not support nondeterministic action outcomes, and classical planners assume there are no other sources of change in the domain than the actions taken by the performance agent based. Those systems therefore cannot be used to solve planning problems in real-world domains involving uncertainty. Also, the resource issue is completely neglected in classic planners. There is no provision for specifying resource requirements or consumption. In spite of those limitations, the ideas and algorithms according to classical planning approaches form the basis for many approaches to planning under the dynamically changing environment, which will be elaborated on in section **2.4**.

2.3 Resource Constraint Reasoning and Scheduling

Resource is an important issue in AI planning. Simply speaking, resources are those objects and products in the world that will affect solving a planning problem. Each resource has certain properties at a specific time and place. Actually, it is a vague concept, and there are many different kinds of resources depending on specific domains. Some resources are physical, which can be consumed or occupied. Some resources are more abstract, such as possibilities and availabilities.

As mentioned in the previous section, the traditional planning approach does not include resources. The resource issue has been considered part of the responsibility for a scheduling system to solve. The common conception of scheduling in AI is that it is a special case of planning in which the actions are already chosen, so the unsolved problem is how to determine a feasible order. Due to this trivialization, scheduling did not receive serious attention in AI until Fox et al. began work on the ISIS constraint-directed scheduling system [15, 94]. Since then, an increasing number of AI researchers have been working in the area. Two well-known textbooks [16, 17] on the subject define scheduling as the problem of assigning limited resources to tasks over time to optimize one or more objectives. Three key things are worth emphasizing in this definition:

1. Reasoning about time and resources is the core of scheduling problems, which have received only limited attention in the AI planning community;
2. Optimization problems exist in almost all scheduling problems, as it is easy to find a legal schedule but much tougher to find an optimal one;

3. Scheduling problems often involve choices. Those choices may be not limited to the choice of task ordering but include choices as to which resources to use to complete a task. Alternative resources may be available for a given task and have different costs and life cycles.

The general AI scheduling techniques are well embodied in research work addressing so-called resource-constrained project scheduling (RCPS) problems. In AI, the most common approach to solving a RCPS problem is to represent it as a *constraint satisfaction problem* (CSP) and to use general constraint satisfaction techniques [18, 19, 20]. Several planning systems have successfully transformed the scheduling problem into some sort of constraint satisfaction problem or linear programming problem in order to make use of solvers developed for those problems [21, 22]. A very natural way of representing RCPS problems as constraint satisfaction problems includes the following steps: 1) Define a variable that represents the start time of each task; 2) specify constraints enforcing the given task orderings; 3) for each time point and each resource, specify the constraint that the total usage of all active tasks at that point does not exceed the capacity of any resource; and 4) assign individual start times to tasks. Many of the initial works on scheduling as constraint satisfaction were done using this approach, and it continues to be the favored representation for certain applications. However, limitations result from the fact that the duration of each task is fixed and the set of choices depends on the number of time steps. The other common representation is based on correctly ordering two tasks so that they will not compete for the same resource. It assigns a Boolean variable for each ordered pair of tasks, representing that the first comes before

the second. And there are constraints among ordering variables, which encode predefined ordering constraints and enforce the proper order of tasks based on variable assignment. The start time for each task is not fixed but constrained. Some constraints require that if each task starts as early as possible, then no resource will be overused. We can see that the task-ordering approach well addressed the limitations in the previous way. For almost any realistic scheduling problem, the resulting search space is significantly smaller. This representation can be very useful in situations where durations are uncertain, as a partial ordering provides much more flexibility then a fixed time-stamped schedule.

Although the constraint framework is very general and can represent complex scheduling problems, there is a limit to how far the paradigm can be extended. When more choices are introduced into a scheduling problem, the number of variables increases significantly and the constraints become very complex. Thus, this approach often becomes impractical for problems that have many interacting choices.

Most interesting real-world problems exhibit characteristics of both planning problems and scheduling problems. They may involve action choice as well as time constraints, resources, and optimization. And most of the time those issues are interdependent. There have been attempts to extend classical planning techniques to treat resources by including techniques that have been developed in scheduling. The O-Plan planner [23] uses optimistic and pessimistic resource profiles to detect and resolve potential resource conflicts. The IxTeT planner [24] uses graph search algorithms to identify minimum critical sets of actions with conflicting resources. A disjunction of ordering constraints is then added to resolve the conflict. There have also been attempts to extend planning techniques to address continuous time and time constraint [25, 26, 27,

28]. I will not elaborate on those efforts since the time issue is not included in this research. Those extensions increase the relevance of classical planning techniques to scheduling problems. However, these are just a beginning step. Because the focus of classical planning is on selecting individual actions, rather than on organizing or synchronizing them, and on discrete state changes, rather than on multiple, interacting asynchronous processes, simply extending planning systems with durative actions and resource capacity constraints is unlikely to provide an effective solution for problems concentrating on resource allocation. On the other hand, the techniques typically used in scheduling systems to solve planning problems are usually problem-specific, which makes it difficult to generate more general solutions.

Recently, a new approach of integrating planning with scheduling has been attracting more and more AI researchers' attention. Krogt et al. propose a resource-based framework for both planning and re-planning [29], which directly integrates causal reasoning into resource allocation and scheduling algorithms. R-Moreno et al. developed a domain independent solver IPSS that integrates an AI heuristic planner synthesizing course of actions, with a constraint-based scheduler for temporal and resources reasoning [30]. There is a significant amount of ongoing research devoted to this approach, and it is strongly believed that more achievements will be reported in the near future.

2.4 Plan Adaptation

From the discussion outlined in section **2.2**, we know that most AI research in classical planning systems has been made under the assumption that the planning domain is certain: the planner's initial state is known absolutely, the effects of all actions are perfectly predictable and the planner is the only agent acting in an otherwise unchanging world. These assumptions have allowed sound fundamental work to be done and many interesting planning systems to be built. However they have also limited the applicability of these systems for real problems, where the assumptions rarely hold. Much research has been conducted to design planners that relax these assumptions for considering uncertainty.

The real world is always changing and unpredictable to certain degree. The planner has to do planning under uncertainty and with incomplete information in real-world domains. An agent working in a world where unexpected hindrances or opportunities could arise should continually be on the lookout for changes that could render its planned activities obsolete. Because its plans should undergo continuous evaluation and revision, such an agent will continually be planning, adapting the current plan to changes and interleaving planning with execution. The aim is to design AI planners for environments where there may be incomplete or faulty information, where actions may not always have the same results and where there may be tradeoffs between the different possible outcomes of a plan. Addressing uncertainty in AI planning algorithms will greatly increase the range of potential applications. As Blythe pointed, AI planning and decision theory would appear to be complementary while there are hard

problems in merging the two approaches [31]. Among the extensive literature on extending the classical planning to planning under uncertainty in a decision-theoretical approach, BURIDAN (a SNLP-based planner) [32], skeletal refinement planning [33], Prodigy and compilation to satisfiability [34], and Cassandra [35], represent a small sample of works being done in this area. A new approach to planning under uncertainty based on Markov decision processes (MDPs) seems promising as the main direction for future work, and it emphasizes extensions that make use of factored action representations, causal links, abstraction and other ideas from classical planning [36]. However, there is plenty of work to be done before we see practical decision-theoretical planning systems, due to the uncertain and dynamic nature of real-world domains.

Among these different approaches for dealing with uncertainty in real world domains, plan adaptation (also called case-based reasoning) seems promising. Adaptation is behavior of an agent in response to unexpected events or dynamic environments [37]. It is impossible for an agent to generate a plan considering all possibilities in the future. There are always unanticipated events or contingencies happening in the dynamic environment so that the generated plan is expected to be adaptable. Plan adaptation means modifying or repairing an old plan so that it solves a new problem. Planning by adapting previously successful plans is an attractive reasoning paradigm for several reasons. First, cognitive studies suggest that human experts depend on knowledge of past problems and solutions for good problem-solving performance. Second, computational complexity arguments show that reason from first principles requires time exponential to the size of the problem, which makes it hard to meet the requirement of real-time

planning tasks. Systems that reuse old solutions can potentially avoid this problem by solving a smaller problem: that of adapting a previous solution to the current task. Actually, many new problem-solving situations closely resemble old situations; therefore there may be advantages to using what has succeeded in the past to solve new problems. An adaptive planner typically accomplishes its task in three phases: First, given a set of initial conditions and goals, a similar plan, one that has worked in circumstances that resemble the inputs, is retrieved from the library. Second, modify the retrieved plan by adding and removing steps, changing step orders, or modifying variable bindings, until the resulting plan achieves the input goal. Finally, generalize the newly created plan and store it as a new case in the library. Hanks & Weld [2] developed a domain-independent algorithm for plan adaptation, which provided a common platform with which one can analyze and compare the various representation schemes and adaptation strategies, as well as explore in the abstract the potential benefits of the case-based approach to planning.

The result of several independent experiments shows that case-based planners using plan adaptation have consistently outperformed the base-level, first principles planner on which these case-based planners were constructed [38, 39, 40]. However, formal studies on the complexity of adaptation suggest that in the worst case, adaptation can be even harder than planning from scratch [41]. Fortunately, Au et al [42] point out the assumption leading to the claim that plan modification is harder than planning from scratch, and show that the Derivational Analogy, which is a widely used adaptation method [38, 39, 43, 44], does not fall under their assumption. In addition, they prove that

the adaptation with the Derivational Analogy can always make planning much easier instead of making it more complex.

Unsolved issues in the plan adaptation approach include how to guide adaptation in a more realistic domain by specifying heuristics, how to improve the analysis of transformational planning systems, and how to be independent of the STRIPS action representation.

2.5 Distributed Planning

Complex planning problems require multiple computational agents and/or human planners to work out a solution together. In this case, the planning is distributed, which means planning activity is distributed across multiple agents, processes, or sites under an environment. Distributed planning involves the collective effort of multiple planners to combine their knowledge, information, and capabilities to generate a global plan that each individual planner could not have done. The distributed parts of the generated plan should be compatible, which means that at least the agents should not conflict with each other when executing the plans. The challenge in distributed planning is how to coordinate activities of each planner efficiently to achieve an effective solution. Figure **2.2** shows a general process for distributed planning.

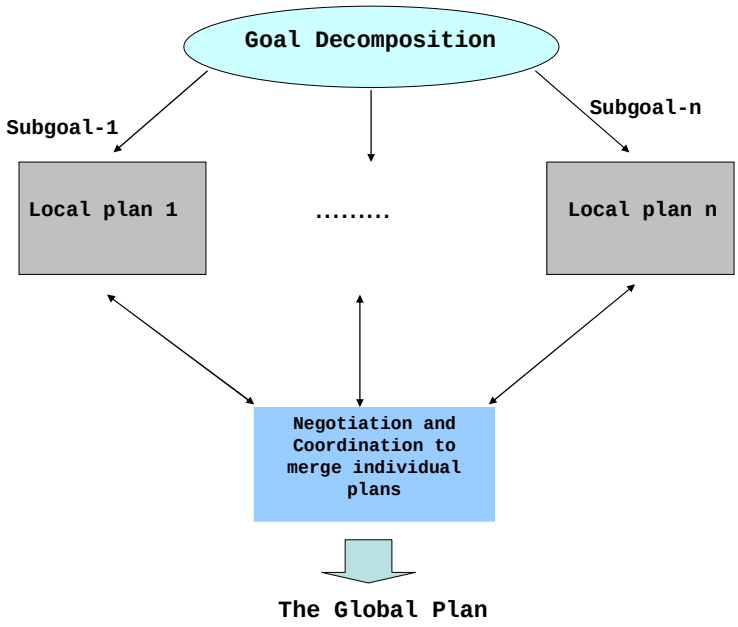

Figure **2.2**: The Flow of Distributed Planning

2.5.1 Abstraction Plan Decomposition

Much of the research in distributed planning is built on top of notions of abstract plan decomposition, such as hierarchical task network (HTN) planning [45]. HTN adopts the well-known *divide* and *conquer* approach to solving a complex planning problem. An abstraction-based plan representation allows a distributed planning agent to successively refine its planning decisions as it learns more about other agents' plans. A hierarchical plan representation also facilitates the interweaving of planning and execution. Agents can execute expanded plans at a certain level while still working on other unexpanded

nodes. Ephrati and Rosenschein [46] suggest an approach to multi-agent planning that contains heuristic elements. Their method makes use of subgoals, and derived subplans, to construct a global plan. Agents solve their individual subplans, which are then merged into a global plan. The suggested approach may reduce overall planning time and derives a plan that approximates the optimal global plan that would have been derived by a central planner, given those original subgoals. Similarly, in Durfee and Lesser's partial global planning (PGP) [47], each agent maintains a partial global plan, which stores its own partial picture of the plans of all the members of the group. PGP focuses on distributed execution and run-time planning and uses a specialized plan representation, where a single agent's plan includes a set of goals, a long-term strategy for achieving these goals, a set of short-term details and a number of predictions and ratings.

2.5.2 Plan Merging

Assuming that multiple agents have formulated their individual plans, now the challenge is how to identify and resolve potential conflicts between individual plans so that the plan merging is smooth. First, we consider the centralized approach for plan merging. Many researchers have studied how to solve the plan merging problem by a single agent [46, 48, 49, 50]. And Yang [51] developed a computational theory on the problem of plan merging. In Yang's formalization, there is a set of plans **P**, where each plan *Pi* is a tuple < **G**, **S**, **L**, **O** >: **S** is the set of plan steps in the plan; **L** is the set of causal links over steps in **S**; **O** is the set of ordering constraints over **S**; and **G** is the set of goal conditions achieved by the plan. Each plan *Pi* must be acyclic. Each step is of

some *type*, and has some *cost*. Steps are typed to restrict the problem complexity, and steps can merge if they are of the same *type*. An optimal solution to the plan merging problem is a merged partial-order plan of the same form as a subplan *Pi* in the original problem whose sum of the costs of the union of steps is minimal, and where the plan network is still acyclic.

Although the multiagent plan merging problem can be formalized similar to the method described above, Yang's assumption that the number of plans (or agents) is constant is unreasonable since the problem cannot be scaled up. Given a loosely-coupled multi-agent system, in which each agent has its own resources and capabilities to carry out the individual planning process, the unsolved problem is how causal links among different individual planning can be addressed in order to coordinate them effectively. The team plan is not a simple collection of different individual plans. All individual plans or subplans interact with each other. Conflicts may arise if two or more planning processes compete for certain resources. An individual planner cannot just act in a single case to represent and generate its plan without considering others' status and the overall situation.

Distributed planning systems require a coordination mechanism that facilitates dynamic collaboration of individual agents, with the goal of satisfying both local and global planning objectives. These agents can, in general, be coordinated by nodes of the dynamic framework in which they exist [52, 53]. For intelligent agent systems, the coordination structure should support the task-solving process using a generic mediation mechanism and should provide communication protocols to link the agents having

common interests. Some researchers are trying to develop a domain-independent solution to the coordination problem in a large agent society. Lesser et al [54, 55] developed Generalized Partial of Global Planning (GPGP) as a domain-independent framework for coordinating the real-time activities of small teams of agents. The basic idea behind GPGP is that each agent constructs its own local view of the activities that it intends to pursue, and the relationships among these activities. This view can be augmented by information from other agents, thus becoming partially global. Individual coordination mechanisms in GPGP help to construct these partial views and to recognize and respond to particular task structure relationships by making commitments to other agents. In this way, GPGP coordinates the activities of agents through modulating their local control with constraints and commitments.

A multi-agent teamwork model certainly can enhance coordination of distributed agents. Wilkins and Myers [56] used a multiagent model called the Multiagent Planning Architecture (MPA) to enable agents to handle plans and planning-related activities collaboratively. Communication protocols among agents are specialized for the exchange of planning information and tasks. Furthermore, MPA can facilitate the integration of agents with diverse techniques to solve complex planning problems. Task Analysis, Environment Modeling, and Simulation (TAEMS), developed by Decker [57], is another framework with which to model complex computational task environments that is compatible with both formal agent-centered approaches and experimental approaches. The design of coordination mechanisms for groups of computational agents depends closely on the task environment. Such dependencies include the structure of the

environment and the uncertainty in the environment. Different from most works that focus on designing coordination mechanisms on properties of the agents themselves alone, TAEMS also considers the structure and other characteristics of the task environment and uses them as the central guide to the design of coordination mechanisms.

2.6 Agent-based Teamwork

Agent-based technologies have been developed as a new paradigm for conceptualizing, designing, and implementing various software systems. Wooldridge and Jennings [58] defined an agent as a computer system in some environment, capable of autonomous actions in order to meet its design objectives. A multi-agent system (MAS) is a loosely coupled network of software agents that interact to solve problems that are beyond the individual capacities or knowledge of each problem solver [3, 59]. An agent in a MAS has incomplete information or capabilities for solving a problem, and there is no global or central control system. In order to accomplish a task, agents must interact with each other through cooperation, coordination, and negotiation [60].

Agent-based teamwork can be viewed as a special kind of MAS that incorporates with explicit collaboration models. They have been widely used in simulating human teamwork, supporting complex decision-making, and teamwork training [61, 62, 63, 64, 65]. In the rest of this section, I will first introduce two teamwork theories that serve as the foundations for agent-based teamwork. Then, I will review and compare several major multiagent architectures modeling teamwork.

2.6.1 Teamwork Theory

Bratman's [66, 67] BDI (belief-desire-intention) provides a mental model of human reasoning. Beliefs represent the informational state of the agent. Desires (or goals) are the objectives or situations that the agent wants to bring about. Intentions are desires that the agent has committed to. A BDI agent is a particular rational agent with those three mental attitudes. Rao and Georgeff [68] provide a formal logical description of the BDI agent model.

Two important theories for modeling teamwork collaboration were derived from the BDI paradigm: Joint Intention Theory [69, 70, 71] and SharedPlans Theory [72, 73].

Joint Intention Theory describes how a team of agents can jointly act together by sharing mental states about their actions. An intention is viewed as a commitment to perform an action while in a mental state. And a joint intention is a shared commitment to perform an action while in a group mental state [70]. Communication is required to establish and maintain mutual beliefs and joint intentions. A team of agents jointly intend to perform an action if and only if the members have a joint persistent goal [71]. One problem with this framework is that the communication points must be built in and cannot be derived automatically, i.e., the exact time when agents should communicate with each has to be specified but not made the at the real time.

SharedPlans Theory is a formal framework for a set of agents to synthesize their partial plans to establish a "global plan". It emphasizes the need for a common high-level team model (shared plan) that allows agents to understand all requirements for plans that

might achieve a team goal, even if the individuals may not know the specific details of other individual plans [72]. The idea of partial shared plans is significant for the refinement process in collaborative planning. It enables partial plans to be modified over the course of planning by a team of agents without impairing the achievement of the shared goals, which is an essential requirement in this research on collaborative plan adaptation.

2.6.2 Agent-based Teamwork Architecture

Joint Intention Theory and SharedPlans Theory provide the foundation for agent-based teamwork architectures. Based on Joint Intention Theory, Tambe's STEAM [74] is an explicit, domain-independent teamwork model that enables agents to reason about commitments and responsibilities in teamwork to realize flexible coordination and communication. STEAM adopts a decision theoretical approach to make communication decisions. If a potential recipient probably already knew a piece of information, and the cost of collaboration is high, then agents do not communicate. STEAM has been applied in several simulation domains such as battlefield simulation and RoboCup soccer.

CAST (Collaborative Agent architecture for Simulating Teamwork) is an agent architecture for modeling teamwork [61, 65]. It is based on SharedPlans Theory, and also borrows some ideas from the joint intentions model, such as the joint intention (the shared goal). As a teamwork model, it enables agents, including both software agents and human agents, to anticipate information needs from teammates, and deliver needed information proactively [75]. There are four major components of CAST: a high-level

language named MALLET to capture teamwork knowledge flexibly for the team to adapt to a dynamic environment, a computational Shared Mental Model (SMM) represented in Predicate/Transition nets to reflect the status of the team, a set of algorithms to dynamically assign tasks to team members and anticipate information needs of teammates based on SMM, and a communication strategy for making decisions about whether information assistance should be carried out. In real-world planning domains, such proactive information delivery behaviors are highly valuable because of the huge amount of information inputs and the high time pressure. Figure **2.3** gives a high-level description of the CAST agent architecture.

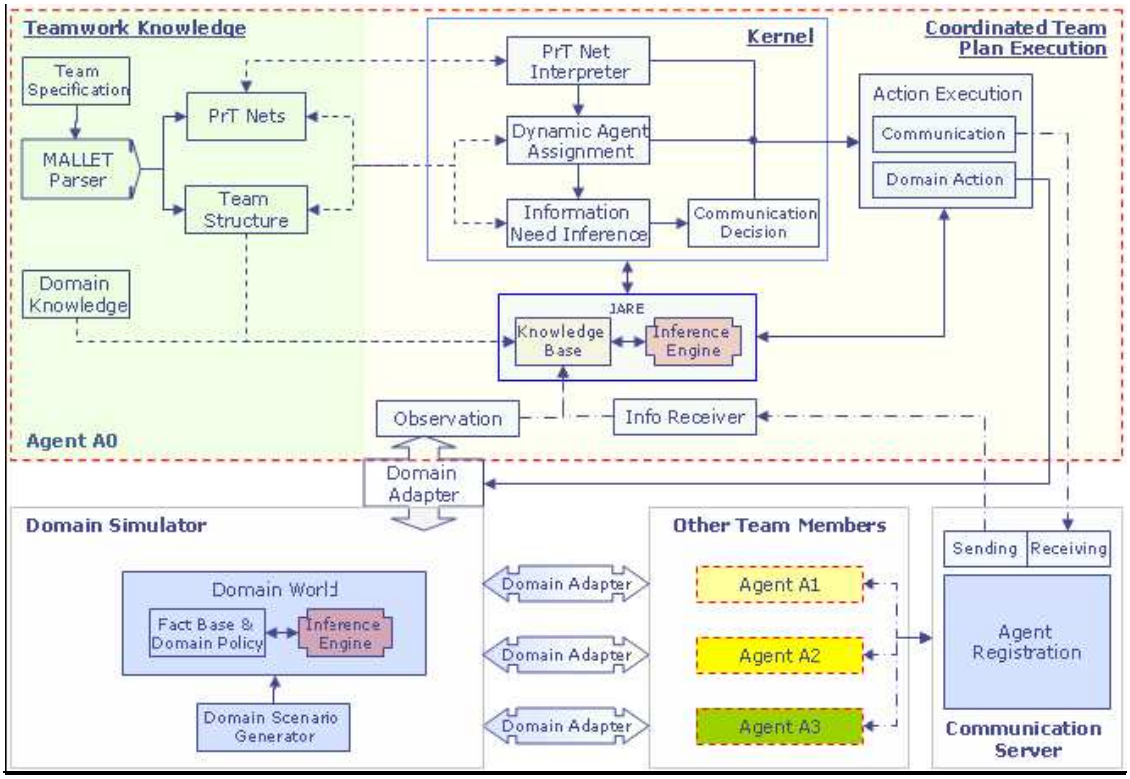

Figure **2.3**: An Overview of CAST Architecture

2.7 Market-based Agents

This section focuses on efficient agent collaboration in a market-based approach. Market-based systems have become an increasingly favorite approach for distributed, self-interested agents to solve decentralized problems. Due to the fact that information, resources, and decision-making power are usually distributed across agents, a market-based approach has the potential to optimize and coordinate activities such as resource allocation and information sharing among distributed agents.

Markets and auction mechanisms are easy to understand and are based on everyday experience. Market-based approaches enable a natural decomposition of the problem, which is well suited for distributed environments. A well-known example of market-based optimization is the file allocation problem in a distributed computer system [76]. One file is fragmented into records so that it can be distributed to the different computers in a network. The fragmentation is done in a way that minimizes the communication costs and the average processing delay associated with a file access. The system consists of n nodes interconnected through a communication network. The network is assumed to be fully connected. The processes running at each of the nodes generate accesses (queries and updates) to the file resource. If a process generates an access request that can not be satisfied locally, the access is transmitted to the node in the network that holds the requested information. Somewhat more formally, the problem can be described as follows. Let x_i be the fraction of the file resource stored at node i. The goal is then to find the optimal $(x_1, x_2, \ldots, x_n)$. The overall expected cost of access to the file system is given by Equation. **2.1**

$$
\begin{aligned}
c &= \sum_{i \in N} (\mathrm{cost\_access\sin g\_}x_i)\, prob(\mathrm{access\sin g\_}x_i) \\
&= \sum_{i \in N} (C_i + KT_i)x_i \\
&= \sum (C_i + \frac{K}{\mu - \lambda x_i})x_i
\end{aligned}
\qquad \textbf{2.1}
$$

In order to adapt to microeconomic theory, Kurose & Simha choose the equivalent problem formulation of maximizing the negative of the cost function. Thus, the utility function $U = -C$ is to be maximized [**76**].

The approach to solving this optimization problem is described as follows. It starts with an arbitrary initial feasible allocation. Each node *I* computes its marginal utility, evaluated at the current allocation. Then each node transmits this value to a central node, which computes the average value. The result is broadcast back to the individual nodes and the file resources are reallocated. Nodes with marginal utility below average utility get a proportional decrease in file resource, while nodes with marginal utility above average get a proportional increase. These steps are iterated until the termination condition is met, i.e. the algorithm terminates when the differences between the marginal utilities of the nodes is below some value and for all pairs of nodes. At each reallocation the algorithm must make sure that no node gets a negative allocation.

Below gives a brief review of how market mechanisms can be applied to distributed resource allocation problems in scheduling.

Auction mechanisms determine who should get the goods and at what prices. In addition to the straightforward single-item auctions, there are many auction environments where multiple items being traded are complements. For example, a left shoe and right shoe are complements that have greater utility acquired together than either acquired individually. Combinatorial auctions [77, 78, 79] allow bidders to bid on combinations of items, and thus directly express these synergies. Each bidder can bid on item bundles, and the auctioneer chooses the set of bids that maximizes the total value.

However, selecting the winning bids, called the general winner-determination problem (WDP), is *NP* -complete for combinatorial auctions [77, 80]. There are two types of approaches to optimal winner determination in the general case. The first one is using powerful general-purpose mathematical programming software. A general optimal winner determination problem can be formulated as a mixed integer problem so that it can be run directly on standard highly optimized software packages such as CPLEX [81]. The second approach is to develop search algorithms specifically for winner determination, combining AI search techniques and domain-specific heuristics. Recently researchers had made a great deal of progress in developing algorithms solving WDP efficiently in this way. For example, the BOB algorithm [80, 82] can solve auctions involving hundreds of items and thousands of bids within 10 seconds. Since my research does not focus on designing algorithms to solve WDP more efficiently, I directly used existing efficient algorithms for solving WDP to optimize the resource allocation in the adaptation process. Basically, different agents can place bids on bundles of resource items they need. And the auctioneer decides winners of this combinatorial auction to maximize the utility of the total resource items.

2.8 Summary

This chapter has reviewed related work in fields such as planning and plan representation, resource constraint reasoning and scheduling, plan adaptation, distributed planning, agent-based teamwork, and market-based agents. As introduced in Chapter 1, my research focuses on developing distributed intelligent computational systems that adapt existing plans online to solve potential resource conflicts caused by the changes in the dynamic situation. The representation of plans in this research extends classical plan representation by explicitly incorporating resource constraint. And an agent-based teamwork model is used for enabling collaboration between agents to adapt distributed plans. Dynamic resource allocation is the focus in the adaptation process and we used a market-based mechanism to achieve the optimization in allocating limited resources. Previous work did not provide adequate studies in this direction. In the subsequent chapters, I will present the framework for collaborative plan adaptation with resource constraint and supportive algorithms in details.

Chapter 3

A Framework for Collaborative Plan Adaptation (CPA) with Resource Constraints

This chapter starts with some motivations for proposing a collaborative plan adaptation framework. Then we describe four major components of the framework in Section 3.2 -3.5. Scope of this framework and comparison with related peer work are discussed in Section 3.6. Section 3.7 concludes this chapter.

3.1 Overview

Complex, dynamic environments afford challenging domains for multi-agent systems (MAS) applications, especially when they face the situation with scarce resources. Unexpected changes usually arise from these environments and bring impacts on the plans being executed or to be executed. A robust MAS application is expected to handle such changes correctly in a timely manner, in order to prevent the given tasks from failing. Since the total available resources are always limited, agents need to pay more attention to changes that affect the current resource status or trigger additional resource requirements. While the plan adaptation problem for a single agent had been investigated by many researchers, collaborative plan adaptation for a team of agents raises important issues that have not been fully addressed. These issues are challenging because information about changes and resource status is distributed among different agents, which makes it difficult for an individual agent to detect and resolve potential

resource conflicts due to the changes. This research studies these issues and focuses on two related research questions:

1. How an agent can efficiently shares information about the change and its impact on other teammates in a timely way?
2. How a team of agents can effectively collaborate on an optimal solution for resolving resource conflicts in adapting existing plans?

The resource-constrained collaborative plan adaptation framework attempts to answer these two questions by providing a domain independent approach to enable a team of agents to collaboratively adapt plans to changes that affect resources. It studies the collaborative adaptation process both within a single agent and interaction between different agents. In a single CPA agent, who serves as a resource coordinator, it includes the resource requirement, resource status, and resource seeking behavior. The interaction between agents focuses on information sharing, coordination to reach a global agreement, and helping behavior to satisfy other agents' resource needs.

The framework defines agents with the role called a resource coordinator. The objective of a resource coordinator is to keep monitoring available resources for the assigned task (i.e., a plan instance), to generate resource needs based on the resource requirement and resource status, and to have those needs satisfied. On the other hand, a resource coordinator needs to respond to resource requests from other resource coordinators. It makes the decision whether to give out the requested resources based on assessing tradeoffs among competing resource requests.

First, the framework makes an explicit representation for the resource requirement in the plan for completing a task. In order to execute the plan, not only all preconditions

need to be satisfied but also the resource requirement must be met. Resource needs are triggered by comparing the resource requirement with the current status of accessible resources.

Second, this framework integrates an agent-based teamwork model called R-CAST, which models high level collaboration and decision making process, and manages information flow in an efficient way. Based on this teamwork model, distributed resource coordinators can establish a shared mental state. Thus, they are willing to cooperate to reallocate their own resources in a way to maximize the global utility. This feature is much different from most market-based agents, who are self-interested and always try to maximize their individual profits.

Third, a market-based solution to dynamically allocate resources among resource coordinators is developed and implemented in the framework. There are two kinds of "bids" sent to a resource coordinator who serves as an auctioneer: Resource Request and Resource Offer. Resource requests are generated by resource needers through resource needs reasoning, which is based on a rule-based inference engine. It takes the resource requirement in a predefined plan as the input and compares it with the current resource status to generate resource needs. The bundle of needed resources, the price that the needer is willing to pay for the bundle, and the needer's name are encapsulated into a resource request. A resource offer is generated by a resource coordinator who is the potential resource provider. When a resource coordinator receives a resource request, it will first check whether this request is relevant (i.e., whether the request bundle or part of the bundle is available for the task being managed by the coordinator). If so, the coordinator calculates the price for a resource bundle that it can offer regarding the

request and generates the resource offer. All resource requests and resource offers are sent to an auctioneer as bids. The auctioneer starts a combinatorial auction to decide the winners for involved resources and notifies the allocation result to each bidder. If a resource requester is the winner, it waits to receive the requested resources that will be transferred from the current owner(s). Otherwise, it either resends the request at a later time or tries to seek the needed resources in other ways. If a resource provider is the winner, it does nothing but just keeps the current resources. Otherwise, it has to give up the offered resources and transfers them to the needer who has been determined as a winner by the auctioneer.

Last, a utility-based method selection is introduced to handle the situation that one task can be completed in alternative methods whose resource requirements are different. The significance of the method selection is to include the *opportunity cost* for a resource coordinator to make the decision whether to offer its own resources to the resource needers. Thus, the cost not only includes the transferring cost but also considers the value of the next best choice that has been given up. This issue is important especially when one method has been already selected to carry out the task. It is possible for an ongoing task to switch from one method to another method in order to relieve certain resources to satisfy incoming resource requests, while the plan execution for accomplishing this task can still continue.

An overview of the framework for collaborative plan adaptation with resource constraints is illustrated in Figure **3.1**. From this diagram, we can have a general idea about how information about changes is processed, how a resource coordinator generates a response to such changes, how a global decision is made based on distributed

knowledge, and how the decision is propagated to each coordinator involved. In the following sections, we will describe major components listed above in details and elaborate relationships among them.

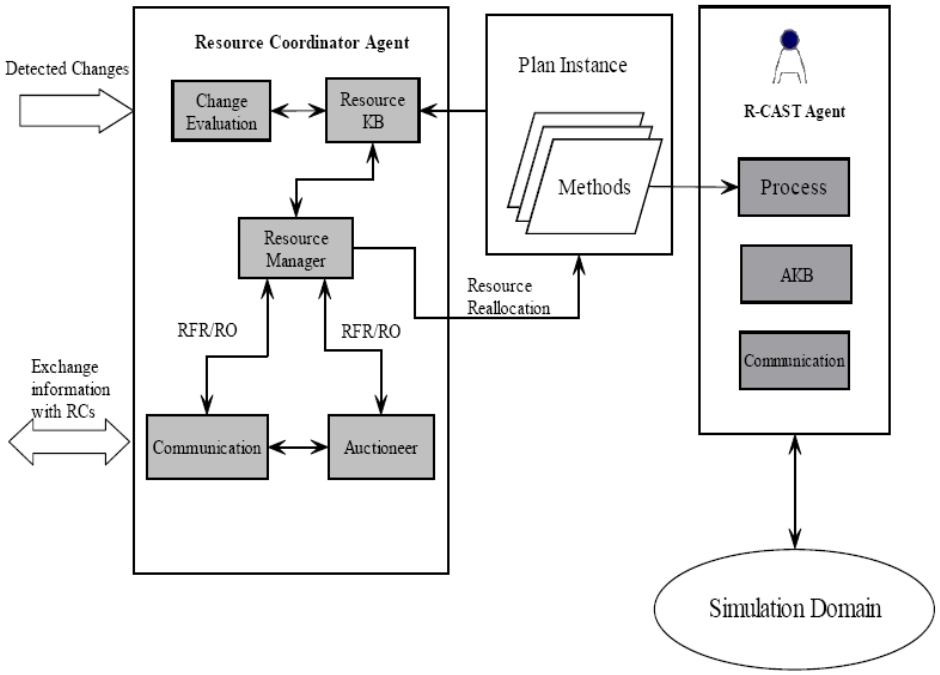

Figure **3.1**: The Resource-constrained Collaborative Plan Adaptation Framework

3.2 R-CAST based Agent Architecture

The R-CAST agent architecture [83] is built on top of the concept of shared mental models [62], the theory of proactive information delivery [75, 84], and the recognition-primed decision (RPD) model [85]. A collaborative RPD decision process was implemented in R-CAST with features such as relevant information sharing, decision process monitoring, and decision adaptation.

The major components of R-CAST are presented in Figure **3.2**. The RPD module uses domain knowledge, past experiences and the current situation awareness to produce a new or adapt an existing decision.

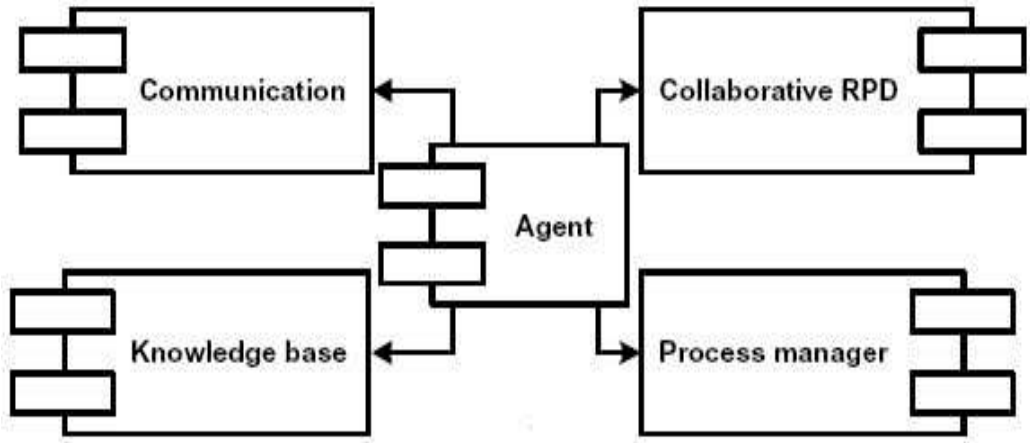

Figure **3.2**: R-CAST Architecture[83]

This study takes advantages of two important modules in R-CAST, which are the knowledge base (KB) and the process manager. The knowledge base is a forward-chaining rule-based system, which enables the agent to maintain what it believes regarding the external world and the other agents. The unique feature is that it is able to reason about missing information relative to the current tasks, and proactively finds ways to satisfy the inferred information requirements. Also, it is proof-preserving in the sense that the proof-trace of a query is preserved, which is important when information link-analysis is needed, and can be very useful in planning for information gathering in subsequent activities. The process manager manages the templates of predefined plans, each of which contains preconditions, termination conditions, effects, and a process body. In addition, we extend the plan templates to include resource requirements for this study, which will be described later in Section 3. The process manger can instantiate plan instances from appropriate templates. The execution of plan instances is scheduled by the process manger based on the constraints associated with the instances and the KB's current state. Figure **3.3** shows the extended architecture for a collaborative plan adaptation agent.

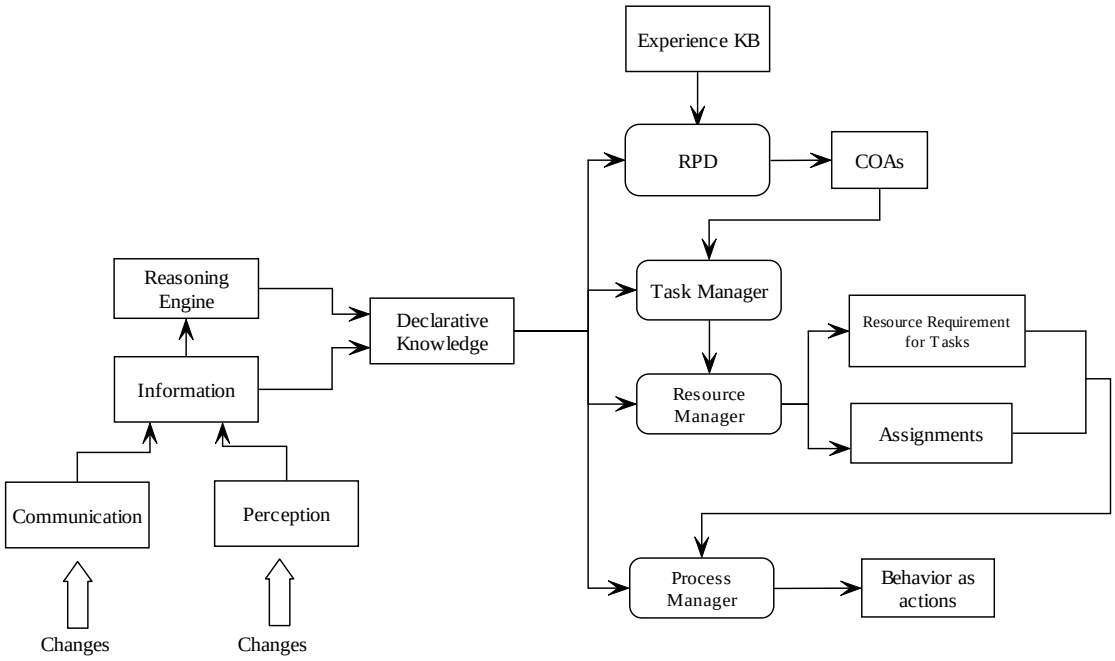

Figure **3.3**: The CPA Agent Architecture Extended from R-CAST

It is clarified here that the input plan is not used for an agent-based resource coordinator to execute the task of managing resources. Rather, the input plan is executed by an agent or a team of agents (depending on whether it is an individual plan or a team plan), who are responsible for the plan execution. Actually, there are two types of collaboration existing in the adaptation process: the collaboration between a resource coordinator and agents who are executing the plans, and the collaboration between multiple resource coordinators. In the first collaboration, a resource coordinator serves as an assistant agent who is dedicated to managing resources to ensure that the assigned task can be carried out by executor agents. In the other collaboration, which is the focus of this research, multiple resource coordinator agents collaborate with each other to coordinate their individual resource needs and work out a global agreement to resolve potential resource conflicts.

A CPA-based agent takes the advantage of the embedded proactive information delivery feature of R-CAST. As a result, it can infer who is the potential needer for a piece of information and proactively delivers such information. Also, since a resource coordinator shares knowledge about the plan with executor agents, their shared mental model enables the coordinator to proactively provide the updated information of resource status to executor agents so that they know whether the current task is ready to be executed or not.

The major responsibilities for an agent-based resource coordinator are described as follows. First, it takes a task instance from the task manger. The task could be a plan or a single action. Next, the resource coordinator parses the task instance and infers the missing resources based on the resource requirement of the task and the current resource

availability. Then, it generates requests for the missing resources and sends them to other coordinators. Different resource coordinators collaborate on resolving possible conflicts due to competing resource requests and develop a global agreement on the allocation of limited resources, trying to maximize the global utility. Finally, if the needed resources are achieved and the task's resource requirement is satisfied, the resource coordinator notifies the adaptation result to executor agents so that the ready assignments can be executed.

3.3 Resource Constraint

As mentioned in previous chapters, the resource constraint must be considered as well as preconditions for a plan to be successfully executed. Different from information, which can be shared by multiple agents at the same time, a resource item can only be used by one agent at one time point. Thus, when multiple agents are trying to use the same resource item concurrently, a conflict will arise and it needs to be resolved. In a dynamic, uncertain environment, unexpected changes usually bring impacts on existing plans regarding the resource constraint. For example, some changes may violate the condition under which the existing resource constraint is satisfied or may introduce additional resource constraints that need to be satisfied. Consequently, resource needs are triggered in order to seek missing resources to repair the violated resource constraint to avoid the failure of the task. Here we define a resource need as an agent (either a human agent or a software agent) needs to possess a resource bundle R by time t, under a specific context, to perform a task. In this research, we assume the total available

resources are always limited so that the resource constraint is emphasized in the process of adapting plans to changes.

Below we discuss how an agent can infer resource needs by keeping an internal representation of the resource constraint and resource status.

3.3.1 Resource Representation

The resource representation serves for two purposes. First, it enables a software agent to model the state of resources in its own knowledgebase. Second, it specifies the resource constraint in a logic-based way that an agent can understand and try to satisfy the constraint.

3.3.1.1 Resource Status

An agent-based resource coordinator maintains its belief about the status of all accessible resources in the knowledge base. In order to efficiently organize such knowledge, we designed two structures for representing resources. The first structure is *resource class*, which is used to describe multiple resource instances of the same type (i.e., instances with the same functionality such as a troop of helicopters) or uncountable resources that may be measured (e.g., 5 gallons of gasoline). A resource class is denoted by a *predicate name* together with a field describing the number (or amount) of resources belonging to this class. Also, it includes a complete list of its attributes and corresponding

values. One special attribute is the *type*, which is used to indicate whether this type of resource is consumable or reusable.

The second one is *resource instance*, which represents an individual resource item. A resource instance is always associated with the name of a class that it belongs to. It also has an *id* different from other instances and domain-specific attributes together with values. Comparing to attributes defined in the resource class, it provides more detailed information such as the *ownership*, its current *status* and what *plans* may use it within what *time periods*.

Resource classes and resource instances are associated together. When there is a change on resource instances, for example, a previously available resource instance becomes inaccessible (not temporarily occupied), the agent needs to update its knowledge about resources by deleting this instance and modifying the attribute indicating the number of instances for the corresponding resource class. Syntax of those two structures and examples are listed in Table **3-1**, from which we can see they provide a concise description of resources that are relevant to an agent with respect to the plan adaptation problem.

Table **3-1**: Two Structures for Resource Representation

	Resource Class	***Resource Instance***
Syntax	(Resource_Class (class ?n) (number/amount ?x) (type ?t) (attributes + values))	(Resource_Instance (class ?n) (id ?i) (attributes + values) (ownership ?owner) (status available/occupied/reserved) ((plan ?id)(begin_action ?ida)(end-action ?idb)))
Examples	(Resource_Class (class Helicopter) (number 5) (type reusable) (fuel full) (loc base)) (Resource_Class (class Fuel) (amount 200 gals) (type consumable) (loc station1))	(Resource_Instance (class Helicopter) (id h1) (attributes + values) (ownership ANG) (status available) ((plan rescue)(begin_action nfly_to_targetA)(end-action return_to_base)))

3.3.1.2 Resource Requirement

In R-CAST agents, processes or predefined plans are used to model procedures for operation. The process manager stores the templates of plans and executes a plan after instantiating it. A plan contains preconditions, effects, termination conditions, fail conditions, a contingency plan, and a process body. It doesn't specify the resource requirement explicitly and the plan can be executed without satisfying any resource constraint. For a resource coordinator agent, however, it must understand the resource requirement of the task being managing so that it can infer resource needs and try to seek those needed resources. Thus, a plan for an agent (i.e., an individual plan) or for a team of agents (i.e., a team plan) for accomplishing a task needs to be extended in this framework to incorporate the resource requirement. A resource coordinator shares the plan with agents who are responsible for executing the plan, and extracts information about the resource requirement from it.

In our framework, a plan explicitly specifies its requirement for certain resources. The resource requirement can be viewed as a set of constraints just like the preconditions to satisfy. Actually, such a requirement describes the minimum set of resources that an agent needs to achieve in order to execute the plan. As long as the available resources meet this requirement, adding extra resources will not affect the outcome of executing the plan. Table **3-2** illustrates an example plan that specifies how to deliver stuffs to a destination with the resource requirement.

This plan template defines a task "deliver the object *o* from the start location *x* to the destination *y* " when the variable *?obj* is bound to an object *o* (e.g., a pile of

sandbags), the variable *?start* is bound to a place's name *x* (e.g., the base of the National Guards) and the variable *?dest* is bound to a place's name *y* (e.g., a leaking levee at New Orleans). The minimum resources required by this plan are two helicopters and two pilots or three boats and three drivers. The first set of resources is the default option, while the second one is an alternative to satisfy the resource requirement. It is easy to understand why the default option is preferred since it's much faster to deliver the object to the destination using helicopters than using boats, especially when the delivery task is in an urgent situation.

Table **3-2**: An Example Plan for Delivering Stuffs to a Destination.

```
(plan deliver_to(?start ?dest ?obj)
   (res-requirement (helicopter 2)(pilot 2)(loc ?start)
    alternatives (1 (boat 3)(driver 3)(loc ?start))
    )
   (termcondition (current_loc ?dest =))
   (utility 500)
   (process
    (seq
    (load ?obj)
    (move_to ?dest)
    (unload ?obj)
    )
   )
)
```

3.3.2 Resource Needs Reasoning

Based on the resource representation described in the previous sections, a resource coordinator agent can infer resource needs of a task that it is currently managing. Similar to how information needs are generated in a CAST agent [75, 86], a coordinator agent built on top of R-CAST generate missing resources as the needs by comparing the resource requirement defined in the target plan and its total available resources. Let C_i denote the resource coordinator for a task T_i, RR_i denote the resource requirement of the plan[3] for performing T_i, which is a set of resource classes {R_1, . . . , R_n}, and RA_{C_i} denote the total available resource instances for C_i, the resource needs can be defined as:

$\mathrm{Re}\,sNeeds(T_i \mid RA_{C_i}) = \{r \mid r \in RR_i \wedge r \notin RA_{C_i}\}$, which is a resource bundle containing missing resources for the task T_i.

Different resource instances belonging to the same resource class are considered equivalent when their attributes are consistent to the additional constraint in the resource requirement (e.g., (loc ?start)). For example, two resource instances r_1 and r_2 are equivalent, denoted by $r_1 \equiv r_2$, when they belong to the same resource class R_1 and the attribute *loc* has the value x after the variable *?start* is bound to x.

From above, we can see that the missing resources are actually made of a set of resource classes {R_1, . . . , R_n}. Each element of it specifies a class of resources including

[3] Here we only assume there is one resource requirement for a plan. Multiple resource requirements for alternative methods defined in a plan will be addressed later.

the number of resource instances that are needed and certain attributes that are bound to constrained values. For example, in the delivering stuffs plan defined in Table **3-2**, after the variable *?start* is bound to the location x , the first selected resource requirement becomes *(helicopter 2)(pilot 2)(loc x)*, which means it requires two resource instances in the class *helicopter* and two resource instances in the class *pilot*, and the attribute *loc* of them must be bound to x . After the resource coordinator extracts such information about the resource requirement, it first checks the resource classes in the knowledgebase. If such required resource classes are found, it further queries if there are enough resource instances satisfying the specific requirement. Supposing there is only one resource instance h_1 in the class *helicopter*, who is available and located at x , and no resource class *pilot* found, the resource coordinator will generate resource needs as *(helicopter 1)(pilot 2)(loc x)*, which indicates the currently missing resources.

After the missing resources are figured out, the resource coordinator sends requests for them to other coordinators who may be the potential provider. A resource coordinator who receives a resource request generates a response based on the status of its own resources and the assigned task. Details about the interaction between the resource needer and the resource provider will be covered in the following sections.

3.4 Adaptive Resource Allocation

The dynamic nature of real world domains requires the allocation of scarce resources to be dynamic rather than to be fixed in order to handle the situation when there

are emerging tasks and changes that affects existing tasks. Usually resource needs are triggered by certain changes and the current resource allocation cannot satisfy such needs. Distributed resource coordinators need to collaborate together by exchanging relevant information such as resource needs, resource status, and task utility, to reach a global agreement on reallocating the total resources. The objective of the adaptive resource allocation is not only to resolve conflicts due to competing resource needs but also to maximize the expected utility from a global perspective.

In the remainder of this section, first we briefly introduce the idea of using market-based mechanism to decide the winner for resources requested by multiple agents. Next, combinatorial auction mechanism is elaborated since resource coordinators usually have to bid on combinations of resource items as well as a single resource item. Then, we describe how to calculate the bid price for a resource bundle based on utility estimation. At last, the detailed process of market-based adaptive resource allocation is illustrated with supporting algorithms.

3.4.1 Market Mechanism

Market mechanisms (e.g., auctions) have been shown to be an efficient way to optimize and coordinate activities such as information sharing and resource allocation among distributed agents, under certain conditions. Market architectures connect sellers with buyers using price to provide a means for low-cost communication of value. There are several market-based implementations for MAS successfully demonstrated in simulations and physical robot-based applications [88, 89, 90, 91].

In this research, the market for agent coordinators to reallocate resources is based on auctions for resources that multiple tasks are competing for. Through auctions, a team of resource coordinators can exchange information about the value of resources to each task efficiently. An auction for resources starts when an agent coordinator detects missing resources for its assigned task to be executed. It generates a request for a set of missing resources denoted by A, together with a price that it is willing to pay for those resources. The request including the price is sent to an auctioneer as a bid. The price is calculated based on estimating the expected utility of such a task, which will be further explained later. At the same time, the request for missing resources is sent to other coordinators who may possess the needed resources. Assume that each coordinator who receives such a request has a set of its own resources denoted by B, let $\Re = A \cap B$, then the receiver generates a bid for each element in its power set of $\Re$ except the empty set. Similarly, the bid price is determined by estimating the contribution of the bidding resource bundle to the task's expected utility. Bids from all potential providers are sent to the auctioneer, who will determine the winners following the principle of maximizing the total outcomes before a given deadline.

The role of an auctioneer can be dynamically assigned to any coordinator based on their current work load, since usually the process of determining winners for combinatorial auctions involves much computational cost. Another advantage of enabling each coordinator with the capability of being an auctioneer is to avoid the situation that a predefined auctioneer suddenly crashes and no else agents can take the responsibility from it to continue the process.

3.4.2 Combinatorial Auctions for Bundled Resources

Combinatorial auctions are introduced to solve the problem when a bid consists of multiple resource items, i.e., a bundle of resources. Individual items in the bundle are complementary. So the price is associated with the bundle. The bid won't be accepted if only part of this bundle can be offered (e.g., a left shoe only or a right shoe only won't be accepted when a person wants to buy a pair of shoes). Combinatorial auctions allow bidders to bid on combinations of items, and thus directly express these synergies. Each bidder can bid on multiple bundles, and the auctioneer determines the set of bids that maximizes the total value of items as the winners.

The *winner determination problem (WDP)* in determining the allocation of limited resources is described as following. Let B_i denote the bundle of missing resources for a task and p_i is the bidding price for B_i. Bidders (i.e., resource coordinators who are seeking for missing resources or who are offering a resource bundle) submit n bids as bundle/price pairs (B_i , p_i). Given the fact that the auctioneer may accept any combination of non-conflicting bids and charge the sum of the associated prices (or called OR bidding), and a decision variable $x_i \in \{0,1\}$ for each bid (B_i , p_i), the WDP becomes the following integer programming problem:

Maximize $\sum_{i=1}^{n} p_i \cdot x_i$ subject to $\sum_{i \in Bids(r)} x_i \leq 1$ for all resource items r , where $Bids(r) = \{i \in [1,n] \mid r \in B_i\}$.

Figure **3.4** shows a combinatorial auction in which there are four resource items for bidding {R_1, R_2, R_3, R_4}, and participating agents submit bids like "\$8 for {R_1, R_4}" and "\$3 for {R_3, R_4}".

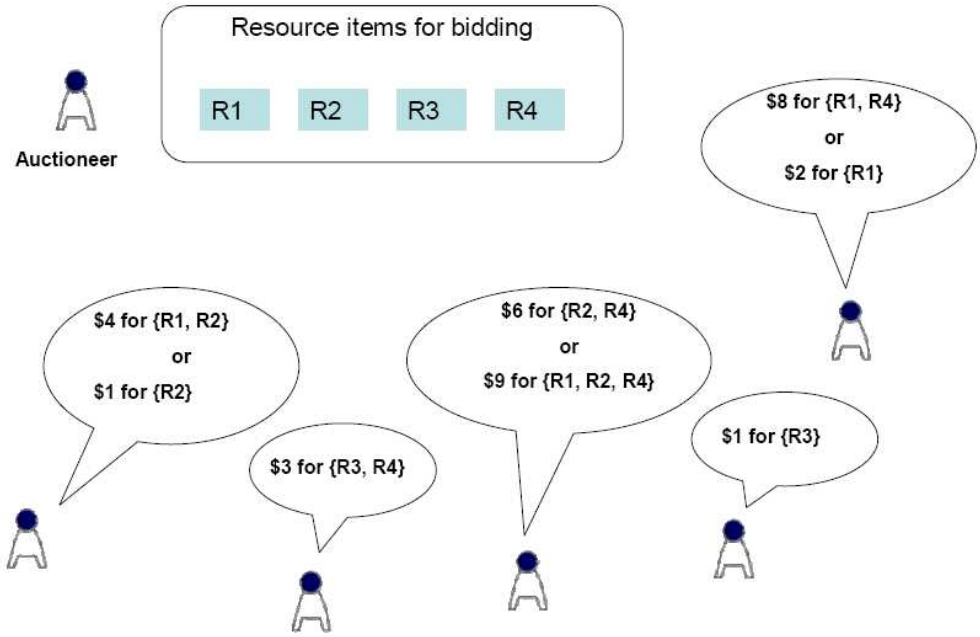

Figure **3.4**: A Example of Combinatorial Auction

In this auction, the acceptable bid sets are in Table **3-3** . From the result, we can see each of the last two bid-sets gives the maximum revenue, which is \$10, so the bids in either of them can be chosen as the winners.

Table **3-3**: The Acceptable Bid-Sets for the Combinatorial Auction Example

Bid-Sets	Total Revenue
"\$2 for {R_1}" + "\$1 for {R_2}" + "\$3 for {R_3, R_4}"	\$6
"\$4 for {R_1, R_2}" + "\$3 for {R_3, R_4}"	\$7
"\$2 for {R_1}" + "\$6 for {R_2, R_4}" + "\$1 for {R_3}"	\$9
"\$9 for {R_1, R_2, R_4}" + "\$1 for {R_3}"	\$10
"\$8 for { R_1, R_4}" + "\$1 for {R_2}" + "\$1 for {R_3}"	\$10

The winner determination problem is computationally complex (NP -complete and inapproximable) and any optimal algorithm for the problem will be slow on some specific problems. There are two types of approaches to optimal winner determination in the general case. The first one is using powerful general-purpose mathematical programming software. As described above, a general winner determination problem can be formulated as a mixed integer problem. So it can be run directly on standard highly optimized software packages such as CPLEX. The second approach is developing search algorithms specifically for winner determination, combining AI search techniques and domain-specific heuristics. Recently researchers had made a great deal of progress in developing algorithms solving WDP efficiently in this way.

In this research, we modified a fast optimal algorithm solving WDP called CABOB, which can solve combinatorial auctions involving hundreds of items and thousands of bids within ten seconds. In the CABOB algorithm, the core technique is a depth-first search through the space of disjoint bid-sets. A search tree is constructed with the received bids and each path in the search tree consists of a sequence of disjoint bids (i.e., those bids do not share items with each other). A path ends when no bids can be appended to it, which means for every bid, some of the bid's items have already been used on the path. At each search node, the algorithm calculates the revenue from the path and compares it with the best revenue found so far to determine whether the current path is the best solution. Thus, after the search is completed, it provides an optimal solution. In addition, the CABOB algorithm uses the A*-admissible heuristic that estimates the revenue that items not yet covered by the bid-set might add in. This heuristic speeds up the search process by pruning of partial bid-sets that certainly won't generate as much

revenue as the best solution found so far. More details about the CABOB algorithm can be found in [80, 82].

The major modification is adding exclusive-or (XOR) constraints in the search formulation to express bidders' general preferences (not only with complementarity but also substitutability). Because we allow a resource coordinator to have alternative resource requirements for a task, one coordinator may generate multiple bids, each of which contains different bundles of resources. Each bidder submits exclusive-or constraints with all its bundle bids, so at most one bid can be accepted. In this way, it avoids the situation that some resource coordinators get redundant resources, while other resource coordinators may be short of such resources.

3.4.3 Auction-based Resource Reallocation

Based on the fundamental knowledge about market mechanisms and combinatorial auctions, we describe the detailed process of reallocating resources among distributed resource coordinators following an auction-based approach. An overview of this process is illustrated in Figure **3.5**.

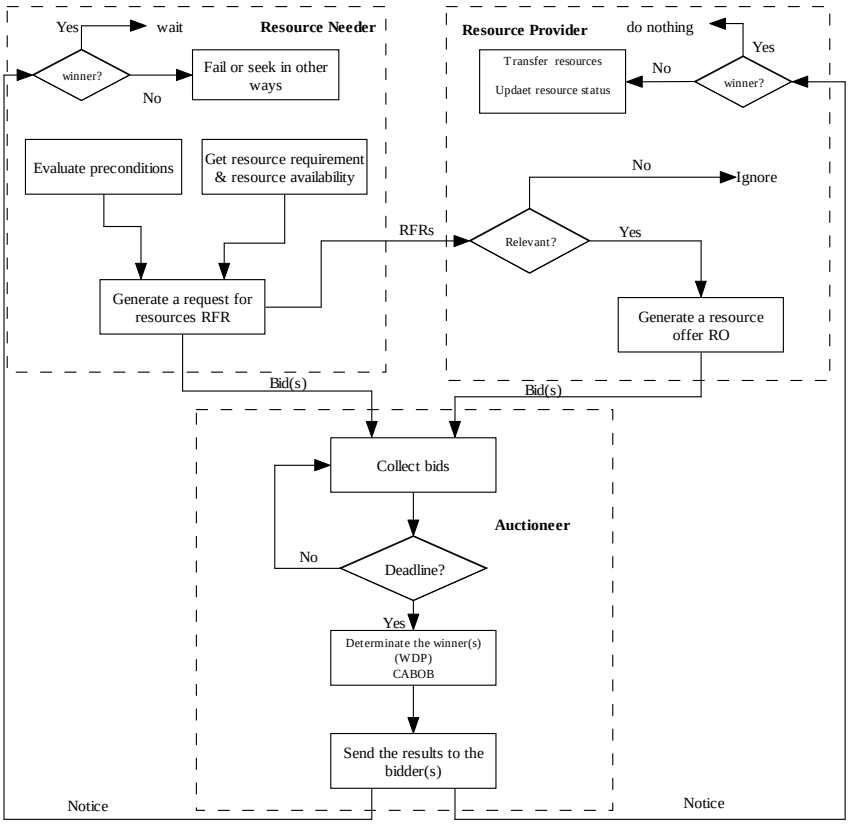

Figure **3.5**: The Auction Process for Adaptive Resource Allocation

There are two types of bids in the auction: one is a *request for resources* (RFR) and the other is a *resource offer* (RO). A request for resources is generated by a resource coordinator to seek missing resources. It is defined as a 3-tuple $\langle RB_i, P_i, C_i \rangle$. RB_i is a resource bundle including all missing resources for task T_i , which is defined as $RB_i = \text{Re}sNeeds(T_i \mid RA_{C_i}) = \{r \mid r \in RR_i \wedge r \notin RA_{C_i}\}$. C_i is the resource coordinator managing task T_i. And P_i denotes the price that C_i is willing to pay for RB_i. A resource offer is generated by a resource coordinator in responding to such a request. It is also defined as a 3-tuple $\langle RB_k, P_k, C_k \rangle$. RB_k is a resource bundle offered by a resource coordinator C_k , who is managing the task T_k, regarding the request $\langle RB_i, P_i, C_i \rangle$ from the coordinator C_i. It is defined as $RB_k = \{r \mid r \in RB_i \wedge r \in RA_{C_k}\}$. There is also a price P_k with the resource offer indicating the price that C_k asks for offering the resource bundle RB_k. When there are multiple RFRs from different resource needers (e.g., C_i, …, C_j) received, C_k synthesizes all those RFR s and generates a single RO , in which $RB_k = \{r \mid r \in RA_{C_k} \wedge (r \in RB_i \vee ... \vee r \in RB_j)\}$ and the price P_k is decided by considering the complementarity of the offered resources to the task T_k.

All RFRs and ROs generated in the process are collected as bids by a resource coordinator with the role of an auctioneer. It should be pointed out that such a resource coordinator may also generate RFR or RO , since it is also managing a task. Before starting the winner determination process, the auctioneer needs to check on the set of bids

to remove *RFR*s that are invalid. An invalid *RFR* is a bid whose required resource bundle (or part of the bundle) cannot be provided by all *RO*s. For example, if the resource bundle defined in a *RFR* contains a resource item "r_x", which doesn't exist in the resource bundle of any received *RO*, such a *RFR* is considered invalid and should not be included in the bid set. The cause of invalid *RFR*s is: when a resource coordinator generates a *RFR*, it doesn't know whether the missing resources are available to other coordinators since the knowledge about resource status is distributed among agents. If we don't exclude invalid *RFR*s in the auction, it will be impossible to satisfy the resource need of such a *RFR* even if it is a winning bid.

The auctioneer starts the winner determination process on the set of collected bids to determine whether each bid is accepted (i.e., a winning bid) or rejected (i.e., a losing bid). By using the modified CABOB optimal algorithm for solving the winner determination problem, the sum of accepted bids' prices is maximized and each resource coordinator can only have at most one *RFR* accepted. The auctioneer sends a notice indicating whether the bid is accepted or rejected to each bidder afterward.

The process of adaptive resource allocation doesn't end after the determination is made. The following step is to match *RFR*s with appropriate *RO*s so that involved resource coordinators can establish "the provider and the needer" relationship among them and start the transferring of resources. For each winning *RO*, the bidder keeps the offered resources. For each losing *RO*, the bidder starts transferring its resource bundle to the resource coordinator who sent the corresponding *RFR*. Bidders of winning *RFR*s receives a confirmation that the requested resources will be transferred soon while

bidders of losing RFRs are notified that their requests cannot be satisfied so they have to either fail current tasks or seek alternatives.

3.4.3.1 Bid Price

Above we introduce the general process for the adaptive resource allocation. A key issue that hasn't been fully explained so far is how a resource coordinator determines the price for a bid. Here we propose a way to generate the price for a resource bundle based on estimating the expected utility of a task that needs the resource bundle. The greater expected utility a task has, the higher the price of resource bundle is.

We adopt the most widely used probabilistic approach to estimate the expected utility of a task. Given an instantiated task T_i, there are two major factors affecting its expected utility $EU(T_i)$:

- The probability that T_i is completed to certain degree: $P(S_t)$, where S_t denotes the status of T_i at this stage.
- The utility of T_i when it is in the status S_t: $U[T_i(S_t)]$.

The second factor is addressing a partially completed task. In the general case, a partially complete task still has certain utility. The first factor is more complicated because the probability of the task T_i being in the status S_t depends on many things such as preconditions, resource requirements, and even uncertainties.

Assuming only preconditions and resource requirements are considered, the probability of T_i in S_t can be calculated by a function f taking conditions and resource status as inputs, which is:

$P(T_i \quad in \quad S_t) = f(C, R)$, C denotes the current conditions and R denotes the current resource status when T_i is executed, and T_i ends up with the status S_t.

Meanwhile, the utility of T_i in S_t can be represented by a function f' with the argument S_t, which is:

$$U[T_i(S_t)] = f'(S_t), t = 1,..., m$$

For example, a simple way to implement this function is calculating the percentage of completed goals comparing to total given goals.

Based on the two functions above, the estimated expected utility of T_i is given by Equation **3.1** :

$$EU(T_i) = \sum_{S_t} U[T_i(S_t)] \times P(T_i \quad in \quad S_t) = \sum_{t=1}^{m} f(S_t) \times f(C, R) \qquad \textbf{3.1}$$

Usually, the degree that a task is completed to and the expected utility of a partially completed task are vague and domain-specific. In this research, we choose a simplified version of the stated problem assuming each task will only have two possible outcomes: *success* or *failure*. And whether a task will succeed or fail depends on conditions, resource status and exceptions. In a task T, suppose there are *n* conditions and *m* resource requirements. Suppose for each condition C_i, the probability that C_i is

satisfied is $P(C_i = ture) = P_i$ (i = 1…n); for each resource requirement R_j, the probability that R_j is satisfied is $P(R_j = ture) = P_j$ (j = 1…m), and the probability of exception is P_e, then the probability that T will succeed is:

$$P_{succ} = (\prod_{i=1}^{n} P_i \prod_{j=1}^{m} P_j)(1- P_e)$$

The probability that T will fail is: 1- P_{succ}.

Let the utility of task T be 0 when it fails, and the utility of a successfully executed task T be U_t, the expected utility of task T is given by Equation **3.2** :

$$EU(T) = P_{succ} \times U_t + (1 - P_{succ}) \times 0 = U_t (\prod_{i=1}^{n} P_i \prod_{j=1}^{m} P_j)(1- P_e) \qquad \textbf{3.2}$$

The utility for a successfully completed task (i.e., U_t) is based on predefined values by domain experts. First, there is a base utility value defined in the plan template for a task (e.g., (utility 500) in the example in Table **3-2**). Second, there are rules reflecting how the variable bindings affect the base utility in a plan instance. For example, a plan template *extinguish_fire(?loc)* has a much higher utility when the variable *?loc* is bound to a school than when it is bound to an empty warehouse, since it is much more important to save people's lives than materials. Last, different methods to complete a task may have different extra utilities in addition to the base utility.

If the bid is a resource offer, the resource coordinator also needs to include switching costs in the price. For example, most time the offered resource bundle is located in a different place from where the needer requires the resources to be. Thus,

there are always costs associated in the process of transferring the request resources from the current place to the target place. The switching costs are included in generating the price for a resource offer in order to keep the auction fair.

Since all agents are cooperative in our system, the aspect that the auctioneer often serves as a trusted third party is not a concern. In order words, every resource coordinator set the bid price honestly. There is no trick for a bidder such as hiding the true price of a bid to increase the probability of winning such resources for its own benefits.

3.4.3.2 Algorithms

Above presents an overall description of the auction-based resource reallocation and the principle of deciding bid prices. This section we use two algorithms to describe the procedure for a resource coordinator to adapt the current task to changes affecting resources, as a resource needer and a resource provider, respectively. Some notations used in the algorithms are listed below:

T_i : Task i;
C_i : Coordinator agent for Task i;
M_{ij} : Method j in Task i;
PC_{ij} : Preconditions for M_{ij};
RA_{C_i} : Resource Availability for coordinator C_i;
RR_{ij} : Resource Requirement of M_{ij};
RB_{ij} : Resource Bundle that is missing for M_{ij}, defined as the set $\{r \mid r \in RR_{ij} \wedge r \notin RA_{C_i}\}$;
P_{ij} : the price for RB_{ij};
RFR : Request For Resources, defined as a 3-tuple $\langle RB_{ij}, P_{ij}, C_i \rangle$;
RO : Resource Offer, defined as 3-tuple $\langle RB_k, P_k, C_k \rangle$.

Algorithm 1 is used for a resource coordinator to analyze detected changes, infer missing resources, generate requests for such resources, and collect replies for potential resource providers. The complexity of Algorithm 1 is $O(n)$. So the computational cost of it is linearly proportional to the number of methods defined in T_i.

Algorithm 2 is used for a resource coordinator who receives $RFRs$ from other coordinators, to analyze the request, generate a resource offer if the requested resources are available, and transfer such resources to the needer if selected. The computational complexity of Algorithm 2 is $O(n)$, which means the computation cost is linear to the number of received $RFRs$.

Algorithm 1 for a resource coordinator C_i who is managing a task T_i that may have missing resources due to changes. Assume there are multiple methods defined in the plan for T_i to be completed, each of which has its own resource requirement.

Begin

For each method M_{ij}

evaluate the preconditions for M_{ij};

get the resource availability RA_{C_i} for M_{ij};

get the resource requirement RR_{ij} for M_{ij};

If $\{r \mid r \in RR_{ij} \wedge r \notin RA_{C_i}\} \neq \phi$,

let $RB_{ij} = \{r \mid r \in RR_{ij} \wedge r \notin RA_{C_i}\}$;

calculate the expected utility $EU(T_i \mid M_{ij})$ of using M_{ij} to complete T_i;

create a bid $RFR = \langle RB_{ij}, P_{ij}, C_i \rangle$, P_{ij} is based on $EU(T_i \mid M_{ij})$;

End If

End For

send RFRs to other coordinators $C_m, m \neq i$;

receive notices from the auctioneer C_{auc};

If one of RFRs is a winning bid,

receive RB_{ij} offered by other coordinators;

notify agents the RR of T_i is satisfied;

End If

Else

notify agents T_i is not ready for execution;

wait for l time steps to repeat the process;

End else

End

Algorithm 2 for a resource coordinator C_k who receives RFRs from other coordinators. C_k is managing the task T_k.

Begin

For each received $RFR = \langle RB_i, P_i, C_i \rangle$, $i = 1,..., j$
  get the resource availability RA_{C_k};
  If $\{r \mid r \in RB_i \wedge r \in RA_{C_k}\} \neq \phi$,
    let $RB_{ki} = \{r \mid r \in RB_i \wedge r \in RA_{C_k}\}$;

End For

let $RB_k = RB_{kl} \vee ... \forall RB_{kj} = \{r \mid r \in RA_{C_k} \wedge (r \in RB_i \vee ... \vee r \in RB_j)\}$
calculate the expected utility $EU(T_k)$;
create a bid $RO = \langle RB_k, P_k, C_k \rangle$, P_k is based on $EU(T_k)$ and potential costs;
send RO to C_i and the auctioneer C_{aucr};
receive notices from C_{auc};
If RO is a winner bid,
  keep the current resources;
Else
  change the status of resources in RB_k to "*reserved*";
  trigger the transferring of RB_k to coordinators who are winners;
  update local resource status RA_{C_k};
End else

End

The potential costs mentioned in Algorithm 2 usually include more than the transferring cost. When there are multiple methods for completing the task T_k, the coordinator needs to consider opportunity costs due to switching from one method to another method. Detailed discussion of the method selection will be covered in **3.5**.

Algorithm 3 is used for a resource coordinator who serves as an auctioneer, to collect bids from other coordinators, determine the winners for resources being competed for, and establish the needer-provider relationship among involved resource coordinators. The complexity of Algorithm 3 largely depends on the complexity of the CABOB algorithm, which runs an $O(|E|+|V|)$ time depth-first search in a bid graph [82].

Algorithm 3 for an auctioneer C_{auc} who receives bids($RFRs$ or ROs) from all resource coordinators. Assume the current time is t and the deadline for starting the auction is t'.

Begin

While $t \neq t'$

collect $RFRs$ and ROs from all coordinators;

update t';

End While

divide all collected bids into two sets { RFR } and { RO };

remove invalid $RFRs$ from { RFR }; A $RFR \left\langle RB_{ij}, P_{ij}, C_i \right\rangle$ is invalid if $\{r \mid r \in RB_{ij} \wedge (\forall C_k, r \notin RA_{C_k}, RO_{C_k} \in \{RO\})\} \neq \phi$

start the winner determination process (CABOB) on { RFR } $\vee$ { RO };

send notices to involved resource coordinators (win or lose);

If the bidder is a winner and its bid is RFR

send info about the resource provider(s) to the bidder ;

If the bidder is a loser and its bid is RO

send info about the resource needer(s) to the bidder;

calculate the expected total utility;

End

It has been noted that the auction should be done in a timely manner, which means there is a deadline for making the auction decision. The time sensitive auction is required for resource coordinators to adapt to the dynamically changing situation. There is always requirement for a task to be accomplished at a certain time. Otherwise, conditions may change and lead the achieved goals to be meaningless. For example, a task of rescuing people from a firing building needs to be carried out before a deadline. If the deadline is missed, which implies there won't be any people still alive in the building, the rescue task becomes useless even if it is ready to be performed now. So, a task must have all required resources available before the deadline is reached or it will fail for sure. Similarly, a resource offer from a potential provider is also associated with a life cycle. Its validity is subject to changes after the life cycle ends. The auctioneer has to update the deadline for starting the winner determination process based on the time constraint of received resource requests and resource offers. When the deadline is approaching, the auctioneer is forced to make the decision, even if there are still $RFRs$ or ROs on the road. In other words, the auctioneer decides winners for resources with incomplete information. Thus, such an "optimal" solution is not really optimal, considering those $RFRs$ and ROs that haven't been counted due to lags. However, the produced solution is the best option given the specific context, in which the total received bids are incomplete. Moreover, this approach avoids the failure of executing a task due to missing the deadline, which may lead to significant utility loss.

3.4.3.3 Utility Update

After resources have been reallocated, each resource coordinator needs to update the information about the task utility as well as its knowledge on resources. Due to the scarcity of resources, there are always a certain number of tasks whose resource requirements cannot be satisfied. After a resource coordinator receives the rejection for its *RFR* s from the auctioneer, and there are no other ways to achieve the missing resources, it treats the task as an unrealizable task and the task's expected utility becomes 0. The resource coordinator will continue being active in the process. If there are still some resources owned by this coordinator for the unrealizable task, the coordinator should release those resources and send this update to the auctioneer. Such released resources can be viewed as a *RO* whose price is only decided by the transferring cost. The auctioneer takes such a *RO* into consideration for new auctions in the future.

3.5 Utility-based Task Method Selection

In previous sections we mentioned the challenge introduced by alternative methods to perform a task. When a task having multiple methods is seeking for missing resources, its resource coordinator generates multiple *RFR* s, each of which represents the resource need for a method. And the modified CABOB algorithm for solving WDP considers the exclusive-or constraint to avoid the case that a needer receives redundant resources. However, when a task has alternative methods and its resource coordinator receives *RFR s* from resource needers, the situation is much more complicated, especially

if one method has already been selected and it is using certain resources being requested. As explained before, the plan adaptation allows a plan to be modified while it is being executed. Thus, even if one method in a plan is being executed, it is possible to switch to an alternative method and continue the plan execution. And resources currently occupied by the selected method are subject to being reallocated to tasks of other resource coordinators.

The challenge comes from how a resource coordinator assesses tradeoffs between alternative ways to accomplish a task so that it can make a correct decision whether to offer the requested resources. Scarcity of resources is the cause of such tradeoffs, and tradeoffs result in an opportunity cost. As mentioned in Chapter 1, the opportunity cost of a resource is defined as the value of the next best alternative that has been given up due to being short of this resource. Any decision that involves a choice between two or more options has an opportunity cost. When the resource coordinator detects that the requested resources are currently being used by a selected method (which is currently the best alternative for completing the task), it calculates the loss of utility due to switching from this method to the next best alternative that is ready. The loss of utility is viewed as an opportunity cost and will be used to set the price for the offered resources instead of setting the price based the task's expected utility, since the task may still succeed with one of other alternatives.

An example of a plan with multiple alternative methods is described in Table **3-4**. In this example, there is a plan for traveling from the current place *?start* to a destination *?dest*. A resource coordinator C_1 is assigned to manage resources for an instance of the plan. There are two alternative methods in the plan to accomplish the

traveling task, each of which has different resource requirements and different method utilities in addition to the based utility of this plan. Also, there are preferred conditions for an agent to select the right method depending on the situation. The first alternative is preferred when the agent faces a time-critical situation, which requires it to arrive at the destination as soon as possible. Thus, the method of traveling by flight will be selected to fit the demand. The resource requirement of the method *by_flight* indicates it needs at least one airplane and one pilot. Those needed resources must be achieved for the agent to execute the selected alternative method. Similarly, the second alternative chooses to travel by driving (the method *by_driving*). The required resource bundle includes a car and a driver. It is suitable for the situation that time is not a critical issue and the budget is low.

Supposing that the method *by_flight* was selected under a certain context and the plan execution is ongoing, there is a change arising in the environment, which triggers a new task instance. The resource coordinator (C_2) assigned for the new task instance generates a request for the resource bundle (AP_1, PI_1), which is an airplane being used by the ongoing plan instance. When C_1 receives the request from C_2, it does not decide the price for offering the required resource bundle based on the utility of its current task's expected utility because it is still possible to complete the task with the other alternative method even if the required resource bundle for the first method had been given out. Thus, the price is actually determined by the opportunity cost of selecting the next best alternative. If other costs such as the transferring cost are not considered, the opportunity cost in this case is 40, which is the difference of two method utilities (120 and 80, respectively).

Table **3-4**: A Plan with Alternative Methods to Accomplish a Task

```
(plan travel_to(?start ?dest)
   (precondition (current_loc ?start)
   (termcondition (current_loc ?dest =))
   (utility 500)
   (choice travel_method
   (by_flight
   (prefcondition (time-critical true))
   (res-requirement (airplane 1)(pilot 1)(loc ?start))
   (method_utility 120)
   (process
      (seq
      (plan_route ?start ?dest)
      (fly_from_to ?start ?dest)
      )))
   (by_driving
   (prefcondition (time-critical false)(budget low)
   (res-requirement (car 1)(driver 1)(loc ?start))
   (method_utility 80)
   (process
       (seq
       (plan_path ?start ?dest)
       (drive_from_to ?start ?dest)
     )))
)
```

The following sections will provide a utility-based algorithm for a resource coordinator to assess the opportunity cost for switching a task between alternative methods and further discussion on strategies of using alternatives will also be covered.

3.5.1 Notations

Before introduce the algorithm for assessing tradeoffs between alternative methods to accomplish a task, a list of notations is provided below.

T_i: Task i;

C_i: Coordinator agent for Task i;

M_{ij}: Method j in Task i;

$U(T_i)$: The base utility of T_i;

$U(M_{ij})$: The additional utility of M_{ij};

RA_{C_i}: Resource Availability for coordinator C_i;

RR_{ij}: Resource Requirement of Method j in Task i;

RB_{ij}: Resource Bundle that is missing for Method j in Task i, defined as the set $\{r \mid r \in RR_{ij} \wedge r \notin RA_{C_i}\}$;

P_{ij}: the price for RB_{ij};

RFR: Request For Resources, defined as a 3-tuple $\langle RB_{ij}, P_{ij}, C_i \rangle$;

RO: Resource Offer, which is a resource bundle offered by a resource coordinator, defined as a 3-tuple $\langle RB_{kl}, P_{kl}, C_k \rangle$;

$\max(\{S\ \})$: the maximum of the set $\{S\ \} = \{s1, s2, ..., s3\}$.

3.5.2 Opportunity Cost

The opportunity cost of switching between alternative methods in a task can be determined in different ways, depending on whether the resource requirements of other alternative methods are satisfied or the probability that such requirements will be satisfied in the near future. Algorithm 4 describes a way to calculate the opportunity cost by checking the readiness of other alternative methods and their additional utilities. In this way, when a resource coordinator assesses the opportunity cost of switching between alternative methods, it only considers alternative methods in which both resource requirements are satisfied by the current resource availability and preconditions are met.

Algorithm 4 for a resource coordinator who receives a $RFR = \langle RB_i, P_i, C_i \rangle$, to calculate the opportunity cost due to switching between alternative methods to offer the requested resources.

Begin

If $RB_k = \{r \mid r \in RB_i \wedge r \in RA_{c_k}\} \neq \phi$

check the status for each $r, r \in RB_k$;

If r is occupied,

determine the method M_{kj}, $r \in RR_{kj}$, who is currently using r;

For other alternative methods $M_{kl}, l \neq j$

check its preconditions;

check its resource requirement RR_{kl};

If both are satisfied,

add M_{kj} to $\{M\}$;

End For

let $u = \max\{U(M_{il})\}, M_{il} \in \{M\}$;

return opportunity_cost = $U(M_{ij}) - u$;

End If

End If

End Begin

First, the resource coordinator C_k analyzes the received RFR by examining if the requested resource bundle or part of the bundle is owned by the task T_i. If so, C_k checks the status of such resources to see if they are currently being engaged and determines which method is using them based on the resource requirement for each method. Next, C_k looks for other alternative methods whose preconditions and resource requirement are both satisfied. Last, among the set of those alternative methods, C_k selects the one with

the highest additional utility and calculate the difference between this utility and the additional utility of the selected method as the opportunity cost for the switching. The opportunity cost combined together with the resource transferring costs becomes the price for the resource bundle offered by C_k in responding to the received RFR. All resources mentioned in the method switching process are reusable resources and the proposed solution is not good for consumable resources, since consumable resources won't be available even after the current method in executing is terminated.

3.5.3 Further Discussion on Alternative Methods

Enabling alternative methods to accomplish a task provides more flexibility in the plan adaptation. Meanwhile, it introduces more challenges for an agent to assess tradeoffs between alternatives to make the right choice. Algorithm 4 simplifies the problem by letting a resource coordinator consider only those alternatives whose preconditions and requirements have been satisfied already. However, it is possible for a method whose resource requirement is not satisfied currently, to get such missing resources from other coordinators. And under certain conditions, the extra effort spent on seeking resources to satisfy an alternative method is worthwhile, which means the utility gained is greater than the seeking cost plus the utility loss. An example representing such as case is illustrated in Figure **3.6** below:

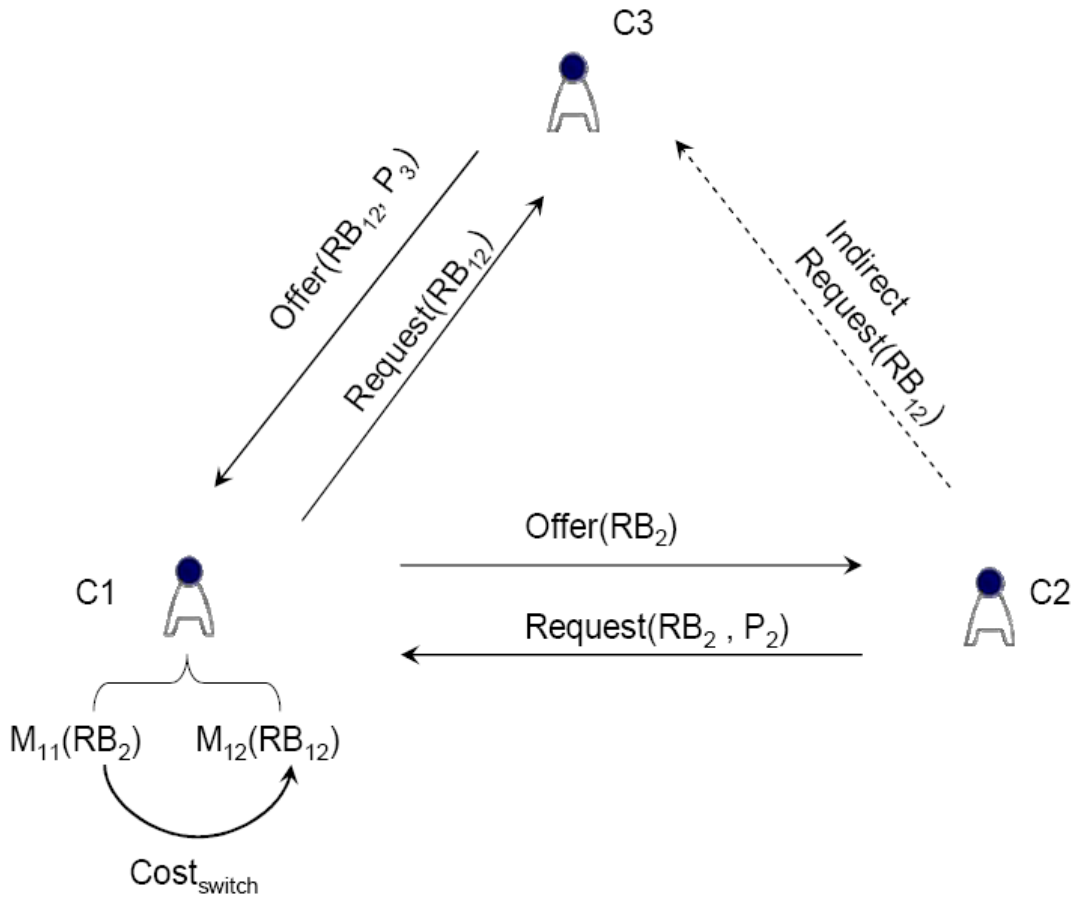

Figure **3.6**: An Example of Indirect Resource Needs

In this example, there are three involved resource coordinators C_1, C_2, and C_3. The resource coordinator C_1 managing a task T_1 receives a $RFR\langle RB_2, P_2, C_2\rangle$ from C_2, Currently, resources in the required bundle RB_2 are being occupied by a selected method M_{11}. Suppose T_1 has another alternative M_{12}, whose preconditions are met while its resource requirement is not satisfied due to a missing resource bundle RB_{12}, and a third resource coordinator C_3 is offering resources in RB_{12} at a price P_3. Given no other resource providers or needers are included in this case, the task T_1 can still be executed by switching from the method M_{11} to the method M_{12}, if C_1 receives RB_{12} from C_3. At the same time, RB_2 is released by C_1 and will be transferred to C_2 in order to satisfy its resource request. The problem is how those coordinators know whether their efforts in reallocating such resources are worthwhile. The proposed solution for this problem is still based on utilities. The coordinator C_1 is the key decision maker in this example since it handles both a *RFR* from C_2 and a *RO* from C_3. First, there is a loss of utility for reallocate RB_{12} from C_3 to C_1, which equals to the price P_3. Second, there is an opportunity cost for C_1 to switch T_1 from the method M_{11} to the method M_{12}, which is denoted by $P_{11} - P_{12}$. Third, the utility gain is the expected utility of T_2 (i.e., P_2, whose resource requirement is satisfied by the reallocation. Thus, the criterion for judging whether such a reallocation process should be carried out is whether the equation: $P_2 > P_3 + (P_{11} - P_{12})$ holds. If yes, C_1 infers that the total expected utility will increase as a result from the resource switching and sends notices to C_2 and C_3 to trigger the

process; otherwise, C_1 keeps its current method selection and rejects the request from C_2 since the total expected utility will be affected negatively by doing the reallocation.

The situation will be more complex if we expand the example shown in Figure **3.6** further. For example, if there are also multiple methods defined in the task T_3 of coordinator C_3, and required resources in the bundle RB_{12} are currently being used by one selected method M_{31} while other alternative methods have missing resources. So, the resource coordinator C_3 should also consider whether to switch T_3 between alternative methods in the same way as C_1 does. Obviously, the computation cost will increase significantly in certain conditions and eventually becomes intractable as the number of such decisions increases. Meanwhile, making a series of those decisions usually takes too much time so that it cannot be guaranteed that the first requester will get to know the result in time, which means its request might have expired when the requested resources arrive. Future research is expected to address those issues and a balanced solution is desired.

Information seeking behavior for satisfying preconditions in a plan is another challenge and it is a broad topic in the field of team-based agents. Due to the scarcity of time and resources, we don't attempt to cover it in this research. So, in the discussion above, alternative methods with unsatisfied preconditions are not included.

3.6 Discussions

3.6.1 Scope of the CPA Framework

We provide a framework for collaborative plan adaptation with resource constraint. In this section, we will examine the validity and limitations of this framework.

The proposed framework focuses on adapting current plans to changes in the environment by consistently reallocating resources among distributed tasks. A market-based approach is adopted to maximize the total utility of all tasks after the adaptation. More specific, the framework uses a combinatorial auction for a team of agents who act as resource coordinators to exchange information about resource needs, resource availability, task utilities, and opportunity costs. The time constraint on each bid (RFR or RO) ensures the adaptation process can converge. A modified CABOB algorithm implemented in an auctioneer agent is able to solve the winner determination problem in combinatorial auctions efficiently. And the opportunity cost due to tradeoffs between alternative methods is calculated based on comparing the loss of utility and the gain of utility.

The planning problem is not covered in the CPA framework so agents don't need to develop plans for given tasks from the scratch. Existing plans are generated by either human experts or advanced planning systems. Resource coordinators directly take predefined plans as inputs and extract information such as resource requirement, base utilities, etc. Also, the initial allocation of limited resources is done before the adaptation process. Each coordinator has its own resources at the beginning and the knowledge about such private resources is not shared with other coordinators.

One limitation of this framework is that one resource coordinator can only manage one task at a time. Theoretically speaking, it is possible for one coordinator to manage multiple tasks concurrently. However, extra efforts will be introduced to a resource coordinator to coordinate those tasks locally due to interdependency between them. Such a coordination process is totally centralized, because only this resource coordinator has the information about resources related to the tasks while such information is not shared with other coordinators. Our framework focuses on the collaboration between distributed coordinators to solve the adaptation problem so it is not necessary to include the locally centralized adaptation within a single coordinator.

Communication is critical for the collaboration between resource coordinators. If the communication is not reliable (e.g., messages may be delayed, impaired, or lost.), the performance of agents implementing this framework will be seriously affected. Currently, the framework doesn't provide a module specially for assuring the communication and assumes messages are always exchanged intact timely, which limit the applicability of CPA agents in situations where communications are vulnerable.

The framework addresses alternative methods for accomplishing a task, which represent substitutability. Resource dependency embodies substitutability too. For example, if one resource item r_1 can be produced from two other resource items r_2 and r_3, a task who needs r_1 can choose to seek r_2 and r_3 , in case that r_1 is not achievable. The challenge is that extra actions may be introduced to convert r_2 and r_3 into r_1. If new actions are added, it will involves causal relation reasoning, which is a key issue in AI

planning problems. The CPA framework doesn't cover the issue of resource dependency to avoid introducing the complexity of AI planning.

3.6.2 Comparison with Peer Work

Our framework captures the agent-based teamwork in solving distributed plan adaptation problems. Different from the centralized approach to adapt plans [2, 38, 39, 40], knowledge about plans, changes and resource status is totally distributed among different agents, which necessitates communications and collaborative behaviors between agents.

The resource constraint is explicitly incorporated in a CPA-based agent, comparing to many planning and plan adaptation systems that ignore it. Some of them do concern other types of constraints such time and causal relations [25, 26, 28]. However, since agents in a wide range of domains have to carry out tasks with scarce resources, the resource constraint is important to be addressed in order to make the result of planning or plan adaptation valid.

Market-based agents have become an increasingly favorite approach for solving decentralized problems such as resource allocation and information sharing. Under the right conditions, each individual agent tries to maximize its own profit in the market, which under the right conditions leads to a globally efficient outcome [90, 91, 92]. Comparing to those self-interested agents, a significantly different aspect of agents in the CPA framework is they are cooperative so that the objective of each agent is to maximize the global revenue instead of for its own sake. In other words, each agent honestly

generates and submits its bids in the auction and there is no need to include a trusted third-party for justice

A brief summary of those comparisons is provided in Table **3-5** :

Table **3-5**: Comparisons of the proposed CPA Framework and Related Work

Comparison Items	The Resource-constrained Collaborative Plan Adaptation Framework	Peer Work
Problem Structure	distributed	centralized
Multiagent System	collaborative	competitive
Resource Constraint	integrated with plans	not considered in planning or plan adaptation

3.6.3 Implementation Guideline

After the CPA framework is developed, we implemented it on top of R-CAST agent architecture, which was described in section **3.2** . The implementing, testing and refining are guided by the "Living Laboratory" paradigm, which was proposed as an ecological approach to integrate theory and practice, continuous process improvement, and tool development in multi-operator systems [93]. The LivingLab is a concurrent research program that includes an array of several research elements: tools, ethnographic studies, paradigms, and prototypes. These elements are mutually informative and concurrently exploit feedback potentials. Observations of team in their work environment, along with acquiring knowledge from different team member perspectives, provide the basis for ecologically valid paradigms to perform empirical research. In turn, feedback from empirical design studies is used as a basis for designing collaborative technology, to evolve lab infrastructure for future studies, or to help validate models built from tools. Figure **3.7** outlines the general process of the LivingLab approach.

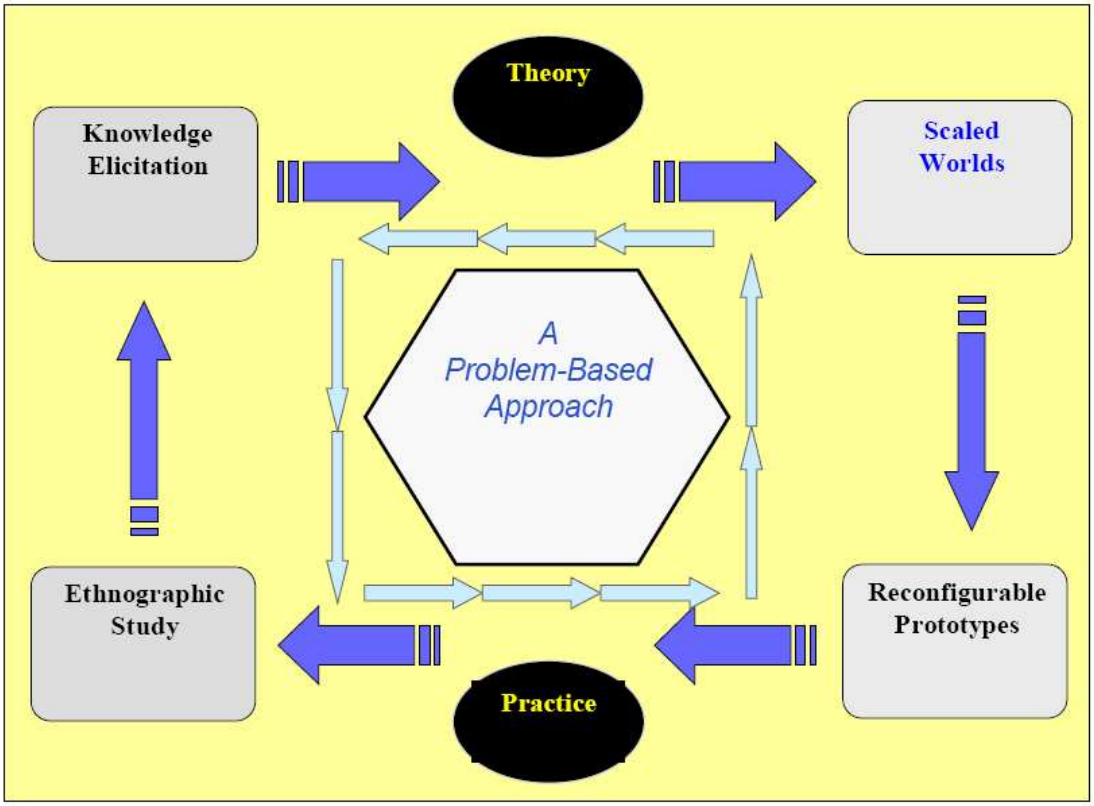

Figure **3.7**: The LivingLab Approach for Problem Solving (McNeese et al. 2005)

LivingLab approach has four major phases, which are knowledge acquisition, scaled world, reconfigurable prototypes and ethnographic study. In the field of knowledge acquisition, qualitative methods are preferred to elicit human experts' knowledge in a given domain, because such knowledge usually are qualitative data such as concepts, strategies and experiences. Specific methods such as observations and interviews can be introduced in this phase. A scenario representing the context is needed to be designed before carrying out those methods. For example, if we want to elicit knowledge about changes that frequently cause the scarcity of resources in a hurricane rescue scenario, we may do interviews with human experts who are experienced in carrying out various rescue tasks under similar situations. Some cognitive system engineering tools such as concept mapping, story boarding design and IDEF can also be used in this phase. Incorporating the knowledge from human experts into the design of software agents can enable them to work in a more intelligent way.

After knowledge acquisition, we develop a scaled world as the test bed for software agents. A scaled world simulates the real world, emphasizing on the environment and problems that agents need to interact with. It is much easier for researchers to control (e.g., isolate certain variables) and testing in a simulator costs much less then testing in the real world. After the scaled world is ready, we implement the agent model within the CPA framework. Interdependency among different functional modules needs to be considered carefully in the implementation process. The implemented agent model is called reconfigurable prototype, which will be tested in the scaled word. During the stages of scaled world and reconfigurable prototypes, the

quantitative method is more frequently adopt. The measurement of the performance of a design is often based on quantitative data. By changing control variables in experiments, the corresponding changes in dependent variables will be showed quantitative data, which are in the forms of tables, curves, or patterns.

The whole process is iterative. The implemented agent architecture is tested in the scaled word, refined according to the result, and tested again, until the performance meets a required standard. At this stage, we can say the implementation is ready to be applied to the real world.

3.7 Summary

This chapter develops a framework using a market-based approach to adapt existing plans to changes in the environment by consistently reallocating resources among distributed tasks.

Major components of the resource-constrained collaborative plan adaptation framework are elaborated in this chapter. A market-based approach is adopted to maximize the sum of utilities of all tasks, which is a major result of the adaptation. More specific, the framework uses a combinatorial auction for a team of agents who act as resource coordinators to exchange information about their resource needs, resource availability, task utilities, and opportunity costs. A modified CABOB algorithm is implemented for a resource coordinator with the role of an auctioneer to solve the winner determination problem in combinatorial auctions efficiently. Detailed algorithms are presented to explain how an agent with different roles acts in the auction process. And a

utility-based approach to calculate the opportunity cost due to tradeoffs between alternative methods is proposed and discussed.

At the end of this chapter, we provide a discussion on the developed framework. Assumptions and limitations of the framework are covered. And we compare it with relate work to demonstrate the novelty of this research. Also, we describe a general implementation guideline in realizing the CPA agent model based on the framework.

Chapter 4

Experiments and Results

In previous chapters, we introduced the resource-constrained collaborative plan adaptation framework, the agent architecture, and related algorithms. This chapter describes experiments designed for evaluating the proposed ideas and algorithms in the framework and testing the implemented solutions.

The framework is aimed to adapt distributed plans to changes in a wide range of domains such as hurricane rescue, military, emergency first response and anti-terrorism. Two characteristics are shared by those domains. First, the situation is always changing, which may cause some previously available resources to become unavailable or introduce emerging tasks with additional resource requests. Second, the total resources are limited, which makes it unlikely to satisfy every task's resource requirement. The framework must go through testing by designed experiments in a simulated environment. The simulation should be designed to incorporate those two characteristics mentioned above.

Even though the proposed framework allows plan adaptation interleaving with plan execution, the adapted plans are not executed in widely used simulators (e.g., RoboCup, RoboCup-Rescue, DDD, etc) for studying agent collaboration. The reason is there are lots of factors impacting the execution of adapted plans other than the resource constraint, which is the focus of our framework. Usually those simulation scenarios are difficult to modify so we cannot control unwanted factors. As a consequence, the

performance of agents who execute the adapted plans may not truly reflect the validity and effectiveness of the adaptation process.

The chapter introduces experiments specially designed for testing two basic features of the framework. The first experiment evaluates the performance of a team of resource coordinators who use combinatorial auction to exchange relevant information to dynamically reallocate resources, by comparing to the performance of a team of agents who simply follow the "request and reply" mode. The second one demonstrates how the performance of such a resource coordinator team can be enhanced by allowing alternative methods for completing a task and enabling coordinators to assess *opportunity cost* due to the tradeoff between alternative methods.

4.1 Introduction

This experiment tested two features of the CPA framework by comparing it with an agent model without those features. Different from a CPA-based agent, the compared agent model generates the response to resource requests simply based on information about the local task and resource. Also, the experiment compared the CPA agent model with and without the feature of utility-based alternative method selection. The results of the experiment suggest that the proposed auction-based adaptive resource allocation can improve the global utility via efficient agent collaboration. Also, the results show that the utility-based method selection can further enhance the performance of collaborative plan adaptation by allowing alternative methods and opportunity cost estimation.

We designed a simulated hurricane relief scenario for experiments to test the solutions in the framework. The scenario simulates a dynamic, resource-rich environment, in which existing plans are subject to changes and the result of plan execution is directly associated with resource availability. In the following sections, we first describe the designed scenario and experiment settings. Then, the detailed procedure of how to conduct the experiments is introduced. Next, the experimental results are analyzed and presented. Finally, a summary wraps up this chapter.

4.2 . Scenario Design and Experiment Settings

In a dynamic environment, intelligent agents usually have to face unexpected changes, which will force them to adapt current plans in order to keep the execution from failure. Since those agents are cooperative, they may exchange information and resources in order to maximize the global utility of those plans under the updated situation. The problem studied here is how an agent evaluates the tradeoff so that it can make a correct decision whether to switch its own resources to its teammates or not. These following conditions are typical features of the domain we are studying:

1. The total resources are limited, which means that required resources for completing all tasks (including emerging tasks) exceed all current available resources.
2. To complete a task, there might be multiple options, each of which has different resource requirements.

3. Different teams are allocated resources that are private to them after the initialization.
4. There is cost associated with sharing information about own resources to other teams.
5. Information of utilities (both resource utility and plan utility) is distributed initially.

4.2.1 A Hurricane Relief Scenario

In this scenario, a Category 5 hurricane hits a Major Metropolitan Area (MMA). Sustained winds are at 160 mph with a storm surge greater than 20 feet above normal. As the storm moves closer to land, massive evacuations are required. Certain low-lying escape routes are inundated by water anywhere from 5 hours before the eye of the hurricane reaches land. In addition to the massive destruction caused by the hurricane itself, there are also areas within the MMA and scattered inland areas that have sustained severe damage from tornadoes that were generated by the storm. Storm surges and heavy rains cause catastrophic flooding to low lying areas. Rainfall from the hurricane, in combination with earlier storms, causes significant flooding in multiple states along the coast. We focus on the following three hurricane relief tasks:

1) Deliver_Food (?dest, ?food, ?deadline): deliver foods (?food) to a large group people who have been isolated in a flooded area (?dest); priority level: 3, deadline: in 24 hours (?deadline).

2) Fix_Levee (?loc, ?bags, ?deadline): transfer sand bags (?bags) to a specific place (?loc) in order to fix a leaking levee; priority level: 4, deadline: in 10 hours (?deadline).
3) Rescue_People (?dest, ?safe_place, ?deadline): rescue a few of persons from a dangerous place (?dest), which will be flooded soon, to a safe place (?safe_place); priority level: 5, deadline: in 2 hours (?deadline).

Each task is associated with a priority level and a deadline that it must be completed before. The priority level contributes to the expected utility of a task and the deadline determines the latest time by which the task's preconditions and resource requirements must be satisfied. The most critical resource shared among those tasks is a troop of helicopters operated by pilots. One helicopter may switch from one task to another task dynamically. Making such a switch is not arbitrary but depends on many facts in the current situation (e.g., the priority of an emerging task, the task's deadline, and the current location of helicopters, etc). An example of such a hurricane relief scenario is illustrated in Figure **4.1**:

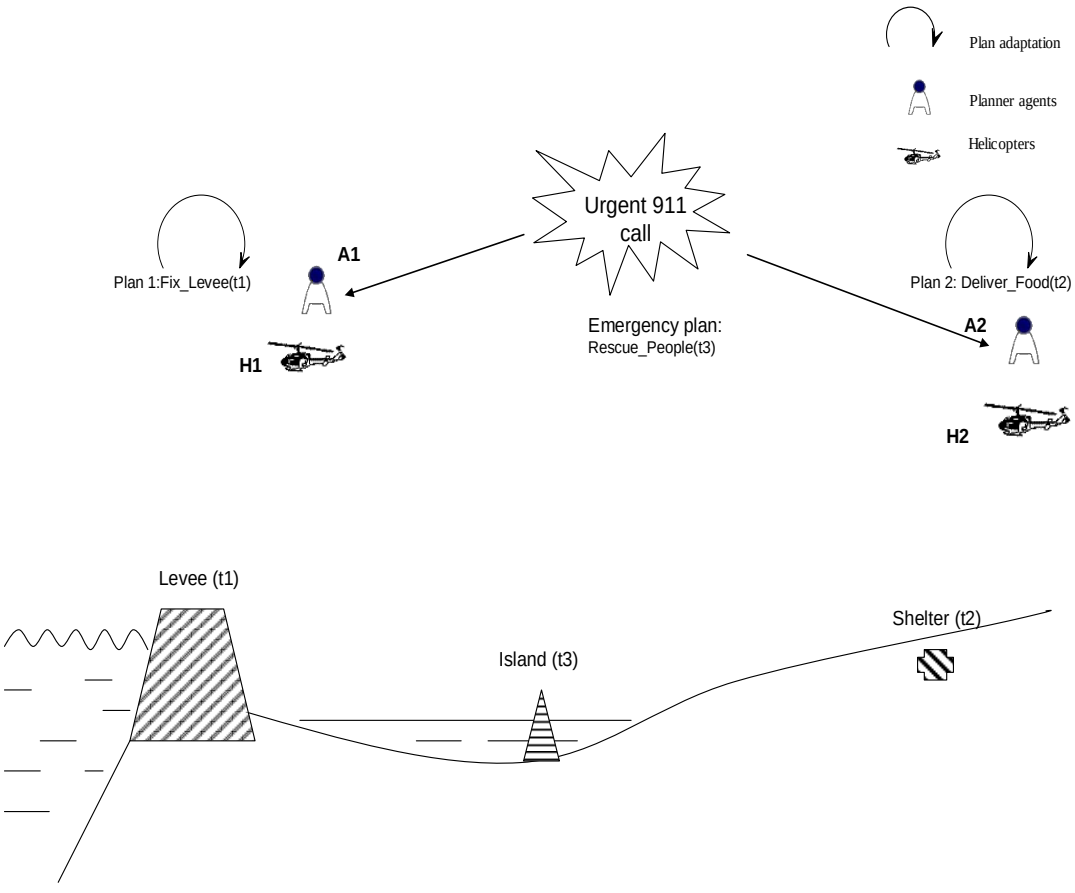

Figure **4.1**: A Hurricane Relief Scenario

In this scenario, one helicopter H1 is in the process of carrying out the task Deliver_Food (?dest, ?food, ?deadline), which requires the helicopter to arrive at the destination with loaded food before the predefined deadline. Another helicopter H2 is engaged in a task named Fix_Levee (?loc, ?bags, ?deadline), which H2 has to transfer sand bags to the specific location to fix the broken dam. Meanwhile, there is an emerging task Rescue_People (?dest, ?safe_place, ?deadline) which requires at least one helicopter as the resource to rescue a group people from a very dangerous place before they will be flooded at the estimated deadline. More helicopters are desirable in case that there are too many people to be rescued. Both coordinators A1 and A2, representing tasks Deliver_Food and Fix_Levee respectively, receive the request from the coordinator A3 of the task Rescue_People. Each of them needs to decide whether to switch its own helicopter from itself to the emerging task in order to maximize the global outcomes. In order to avoid redundant resource transferring, the resource requester will also make a decision to select the right provider in case that there are multiple willing providers. In this case, supposing A3 places a maximum willing price 500 on its request for one helicopter: (?h, 500, A3), and A1 generates a bid for its own helicopter H1 at a price 300: Bid (H1, 300, A1) and A2 generates a bid for its own helicopter H2 at a price 400: Bid (H2, 400, A2), it turns out the bid from A1 will be accepted finally and H1 will be switched from the task Deliver_Food to the task Rescue_People. The expected global utility will be 900, which is more than the previous expected global utility 700 if we don’t consider the switching cost here.

A plan example of for the task Deliver_Food is listed in Table **4-1**

Table **4-1**: A Plan Example

```
(plan Deliver_Food(?dest ?food ?deadline)
   (res-requirement (helicopter 2)(pilot 2)(loc ?start)
    alternatives (1 (boat 3)(driver 3)(loc ?start))
    )
   (termcondition (current_loc ?dest =)(current_time ?deadline
>=))
   (utility 500)
   (process
    (seq
    (load ?food)
    (move_to ?dest)
    (unload ?food)
    )
   )
)
```

Major resource types in the hurricane relief scenario are listed in Table **4-2**:

Table **4-2**: Resource Types

Type	Name
type 1	Helicopters
type 2	Boats
type 3	Pilots
type 4	Drivers
type 5	Foods
type 6	Sandbags

4.2.2 Configurations of Three Teams

There are three teams of resource coordinators in the experiment. Resource coordinators in different teams have different strategies for handling changes on resources. We denote the teams as Team A, Team B, and Team C, separately.

In Team A, each resource coordinator follows a simple “request and reply” way to handle resource requests. More specific, when a request is received, a resource coordinator first checks the resource availability to see if the requested resource bundle is currently owned by itself. If yes, it satisfies the request by giving out the resources being

requested. Otherwise, it just ignores the request. In this process, a resource coordinator doesn't take the utility information into consideration.

In Team B, resource coordinators are implemented based on the CPA agent model. They detect changes that impact current resources, generate resource requests, exchange relevant information via combinatorial auctions, and reallocate resources in a way to maximize the global utility. However, they are not endowed the ability to select the right one among alternative methods.

Resource coordinators are almost as same as those in Team B, except that an agent in Team C can reason about opportunity cost by assessing tradeoffs among alternative methods. Information about opportunity cost is included in generating bid prices. As a result, the adaptation is more flexible and more efficient.

The input predefined plans are the same for all three teams. Those plans are instances of three different tasks: Deliver_Food (dest, food, deadline) Fix_Levee (loc, bags, deadline), and Rescue_People (dest, safe_place, deadline), which are described previously. There is at least one instance for each task. Let t1, t2, t3 denote three instances, each of them belongs to a separate task. Resource requirement and utility information extracted from the plan instances are listed in Table **4-3**

Table **4-3**: Resource Information of Three Basic Task Instances

Task Instance	T1, managed by C1	T2, managed by C2	T3, managed by C3
Resource Requirement	(task t1 500 (m1 50 (r1 r4)) (m2 40 (r2)) (m3 30 (r5)))	(task t2 700 (m1 80 (r2 r4 r7)) (m2 70 (r6 r8)))	(task t3 750 (m1 150 (r1 r3)) (m2 30 (r8)) (m3 10 (r4)))
Resource Status	(resource type1 r1 (owner m1)) (resource type2 r2 (owner m2)) (resource type3 r4 (owner m1))	(resource type1 r7 (owner m1)) (resource type2 r6 (owner m2))	(resource type3 r3 (owner m1))
Selected Method	m1	n/a	n/a

Table **4-4** shows the initial resource allocation for those three task instances.

Table **4-4**: Initial Resource Distribution

Task Name	Deliver_Food		Rescue_People		Fix_Levee	
The number of instances	1		1		1	
Available Resources	helicopters	r1	helicopters	r7	sandbags	r3
	pilots	r3	boats	r6		
	boats	r2				

4.3 Procedure

The simulation starts with the initial configuration. As the time goes on, emerging tasks are generated randomly, who may trigger competing resource requests. The total utility of all succeeded tasks are calculated after the process terminates. We use the number of total triggered tasks (both succeeded and failed) as the control variable to demonstrate the tendency of how the team performance responses to the number of tasks.

During the experiment, an interface implemented in the agent model is provided to show information about the adaptation process. Also, with this interface, human agents are able to interact with such an agent-based resource coordinators by querying and inserting facts on resource status. However, human labor doesn't contribute to the adaptation result, which means human agents don't interfere with the adaptation by providing certain guidance. It is possible for a team having enhanced performance by including human coordinators and agent coordinators but that is out of the scope of this research.

Figure **4.2** shows the interface of a resource coordinator. From the resource manager monitor, we can view the current task being managed by the agent, the status of resources owned by the agent, the auction process, and the result of resource reallocation, etc.

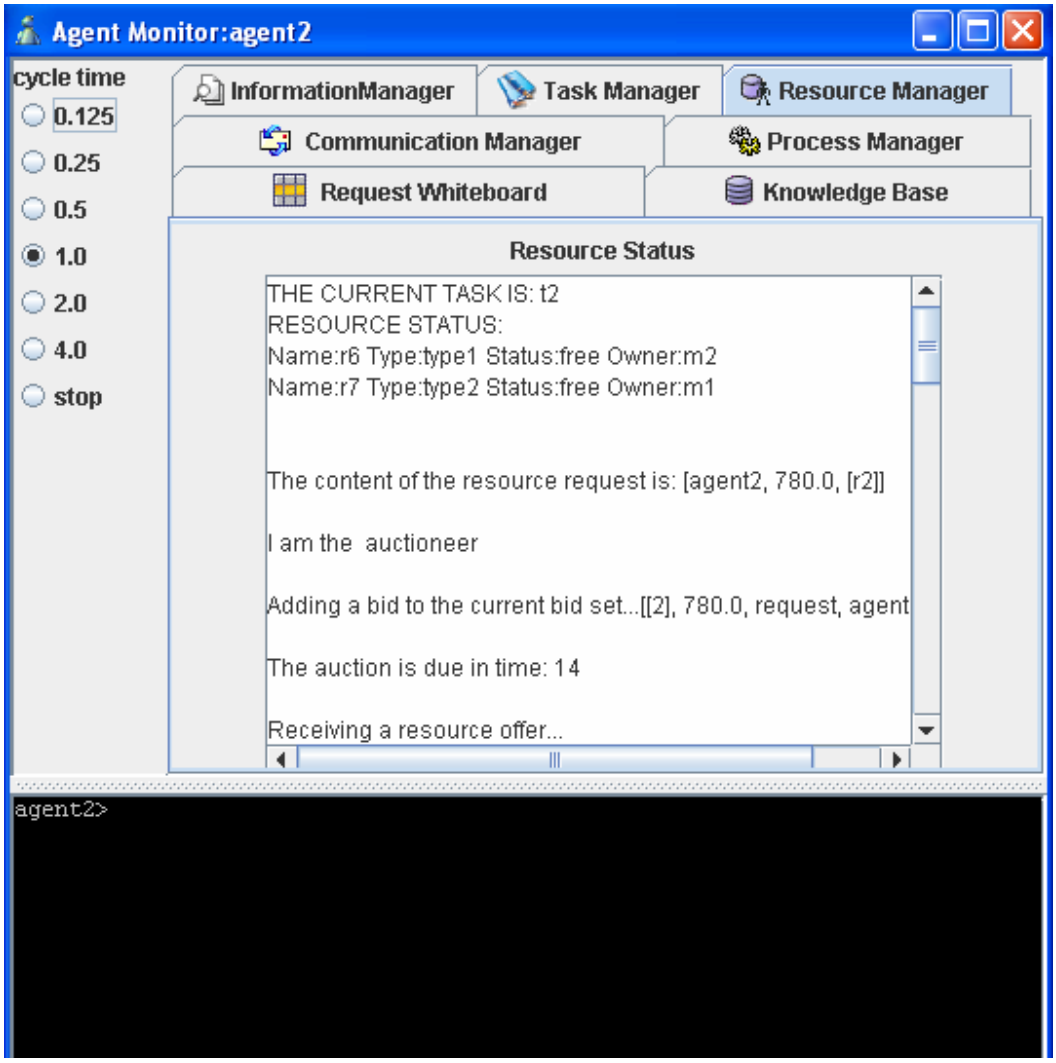

Figure **4.2**: Resource Coordinator Interface

The first experiment is used to compare request-reply based resource exchange strategy (Team A) with the adaptive resource allocation strategy (no alternative methods reasoning) (Team B).

The experiment setting ups are listed as following:

1. The total number of task instances (or resource coordinators) increases from three to seven, (i.e., there are total five cases). Thus, the number of task instances is taken as the control variable to show how the team performance is related to it. There are three basic task instances t1, t2, t3 which are described in Table **4-3**. They are predefined and each of them has a set of resources allocated initially.
2. In addition to three base task instances, there are other task instances added as the total number of task instances increases. All involved resource items are labeled as r_i, where $i \in [1,2,...,10]$. We randomized the resource requirement of each emerging task as a subset of [r1,…,r10]. The value of each task's base utility is also randomly assigned between [600 – 1000], and is a multiply of 50.
3. For each team with a given number of tasks, we test it with eight runs. Each run takes around three minutes.
4. For Team B, a resource coordinator is designated to take the auctioneer role randomly.

The second experiment is used to show how the utility-based alternative method selection strategy affects the performance of the team using adaptive resource allocation. There are two teams: Team B and Team C. The configuration of Team B is as same as the one in the first experiment. The configuration of Team C is based on the configuration of Team B, with additional information below:

1. The resource requirement for alternative methods is generated randomly from [r1,…,r10];
2. The value of additional utility for each method is randomly selected from [10, 200], and is a multiple of 10;
3. Each resource coordinator is configured to enable the utility-based method selection function.

4.4 Data Analysis and Results

Results from the two experiments are combined together so the three teams can be compared together.

Table **4-5** and Figure **4.3** list and plot the average performances of three teams evaluated by the global utility of adapted task instances. And Figure **4.4** plots the average performances of three teams based on the number of completed task instances.

Table **4-5**: Comparing The Global Utility

Task Number	Team A	Team B	Team C
3	681.25	722.50	751.25
4	757.50	1455.00	1480.00
5	1326.25	2086.25	2303.75
6	1716.25	2493.75	2990.00
7	2325.00	3171.25	3637.50

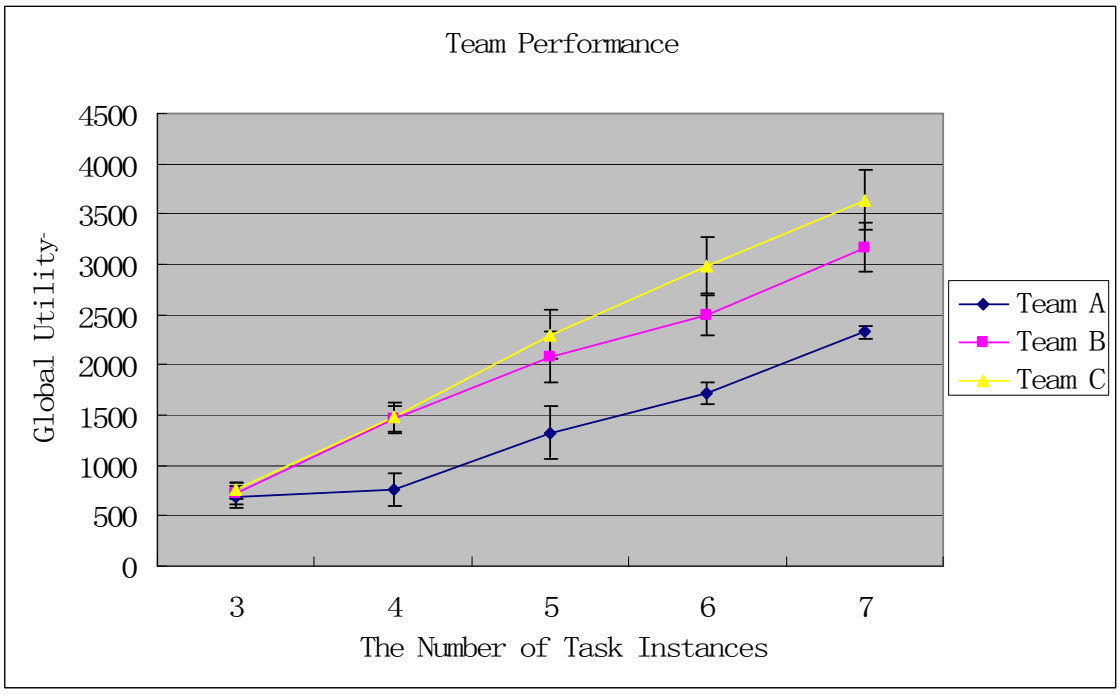

Figure **4.3**: Performance by Global Utility for Each Team

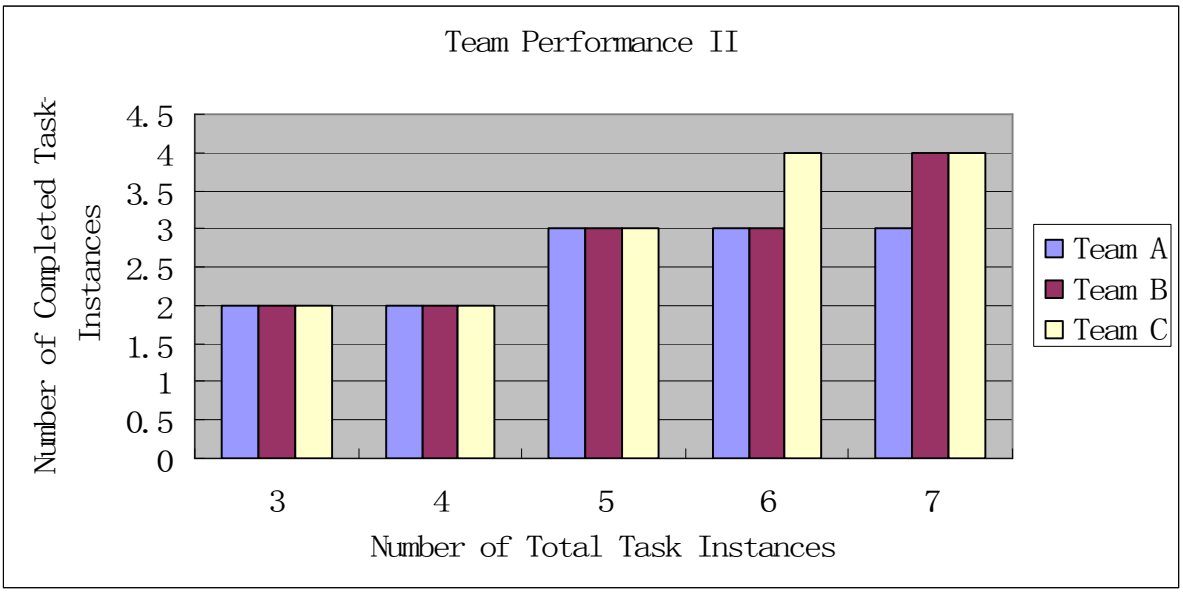

Figure **4.4**: Performance by Number of Completed Task Instances

In Figure **4.3**, each point denotes the average value of the global utility obtained from the eight test runs. And the variance on each point is plotted along with the average value. From the plots, we can see that the performance of Team C is the best based on the global utility. The performance of Team A is the lowest. And Team B performs between them. The differences between their performances follow the trend to increase as the number of total task instances increases. Thus, we can see the adaptive resource allocation and the utility-based method selection can help agents effectively adapt plans to resource-related changes in the environment.

However, from Figure **4.4**, we can see that the number of completed task instances doesn't obviously get improved by using the solutions in the CPA framework

comparing to by using the “request and reply” based strategy. It is because a CPA-based resource coordinator is aimed to maximize the global utility based on the total limited resources, instead of trying to complete more task instances. Given the situation that some task instances may have very high expected utilities, it is worthwhile to switch resources from several other tasks of low expected utilities to satisfy the resource requirement of a task with a high utility. As a result, the total number of completed task instances may even decrease sometimes. No matter how the number of completed task instances may vary, the global utility is still maximized by using CPA-based resource coordinators to solve the adaptation.

To further evaluate the findings above, we used paired t-tests to prove that Team B performs better than Team A, and Team C outperforms Team B.

The procedure for comparing the means of global utilities in Team A (μ_1) and the means of global utilities in Team B (μ_2) is listed below:

1. The null hypothesis H_0: $\mu_1-\mu_2 =0$, and H_a: $\mu_1-\mu_2 < 0$
2. Let $\alpha = 0.05$
3. Use Minitab to calculate the paired t-procedure for the two samples. The output is shown in Table **4-6**

Table **4-6**: Paired T-test Results for Team A and Team B

Paired T for Team A – Team B

	N	Mean	StDev	SE Mean
Team A	5	1361.25	686.11	231.23
Team B	5	1985.75	942.2	312.59
Difference	5	624.50	204.72	71.49

95% lower bound for mean difference: 102.57

T-Test of mean difference = 0 (vs < 0): T-Value = 1.37 P-Value = 0.0134

Since the p-value is 0.0134 < 0.05, the null hypothesis is rejected. We can claim that the average global utility generated by Team B is significantly greater than the average global utility generated by Team A. The result reinforces the findings that the market-based adaptive resource allocation can perform better than the simple "request and reply" strategy, from the perspective of the global utility.

Similarly, we compared the performances of Team B and Team C using paired-test and the procedure for comparing the means of global utilities in Team B (μ_1) and the means of global utilities in Team C (μ_2) is listed below:

1. The null hypothesis H_0: $\mu_1-\mu_2 = 0$, and H_a: $\mu_1-\mu_2 < 0$
2. Let $\alpha = 0.05$

3. Use Minitab to calculate the paired t-procedure for the two samples. The output is shown in Table **4-7**

Table **4-7**: Paired T-test Results for Team B and Team C

Paired T for Team B – Team C

	N	Mean	StDev	SE Mean
Team B	5	1985.75	942.2	312.59
Team C	5	2232.50	1152.53	374.32
Difference	5	246.75	84.21	23.14

95% lower bound for mean difference: 42.23

T-Test of mean difference = 0 (vs < 0): T-Value = 1.64 P-Value = 0.0364

Since the p-value is 0.0364 < 0.05, the null hypothesis is rejected. We can claim that the average global utility generated by Team C is significantly greater than the average global utility generated by Team B. The result shows that the utility-based alternative method selection can further enhance the performance of market-based adaptive resource allocation, from the perspective of the global utility.

From the analysis above, we can find that the performances of three teams are close when the number of total tasks is small. As the number of total tasks increase, the performance difference of Team A, Team B and Team C increases too. The global utility generated by Team A doesn't increase as fast as the global utilities in Team B and Team

C. Also, Team C shows the capability to maximize the global utility when there are enough task instances so that the alternative method selection can benefit the coordination between different resource coordinators, regarding the allocation of limited resources.

The data analysis above is based on the assumption that the data is normally distributed and the variance is homogeneous. The size of total data for each team is $5 \times 8 = 40$, which is large enough to be considered as a normal distribution. To further validate the assumption, a residual plot for the data is constructed in Figure **4.5**.

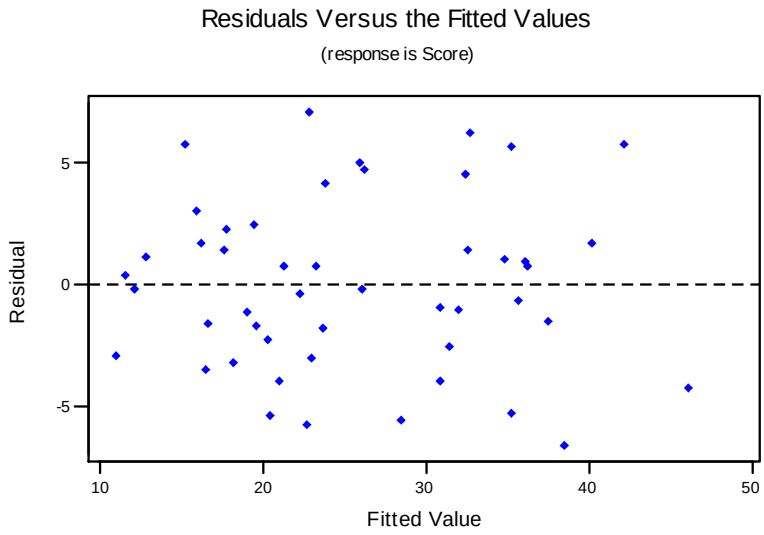

Figure **4.5**: The Residual Plot for Experimental Data

From the plot, we can see that the data are actually normally distributed and there is homogeneous variance among the data.

4.5 Summary

The experiments evaluated two critical components in the framework of collaborative plan adaptation with resource constraint. A hurricane relief scenario was designed to conduct the experiments. The scenario simulates a dynamic, resource-rich environment, in which existing plans are subject to changes and the result of plan execution is directly associated with whether the resource requirement can be satisfied in time. We used the global utility as the measurement for evaluating team performance. And statistical methods are applied to analyzing data and testing hypotheses.

The results suggest that enabling resource coordinators to use combinatorial auction to exchange relevant information can effectively reallocate resources when changes trigger competing resource needs. And the adaptation significantly enhances the global utility of all tasks given the total limited resources, comparing to making a resource coordinator response to resource requests simply based its local information. Also, an experiment is used to compare the performance of resource coordinators with and without the feature of utility-based alternative method selection. The result shows that the utility-based method selection can further improve the performance of collaborative plan adaptation by giving a higher global utility. The result is due to allowing alternative methods and opportunity cost estimation so that the resource reallocation is more flexible and efficient, especially when the number of total tasks is large. However, the results don’t support the claim that the two features of CPA-based

resource coordinator can significantly increase the number of completed tasks, since there is no direct relation between the global utility and the number of completed tasks.

Chapter 5

Conclusions and Future Work

This thesis proposes a resource-constrained collaborative plan adaptation (CPA) framework, within which intelligent agents are implemented to collaborate on adapting existing plans to changes that impacts the resource constraint, in a dynamic environment. First, the CPA framework includes an agent-based resource coordinator to manage resources for a task by inferring missing resources for the task and sending requests for such resources to other coordinators. Next, the CPA framework uses combinatorial auctions for a team of resource coordinators to exchange relevant information such as resource needs, resource status, task utility, and opportunity cost, etc. An algorithm for solving the winner determination problem in combinatory auctions is implemented to decide how the limited resources should be reallocated to involved coordinators. The global utility of total tasks is maximized in this way. Finally, a utility-based alternative method selection algorithm is developed for a resource coordinator to assess its *opportunity cost* for offering its own resources to others by considering the tradeoff between alternative ways to accomplish a task.

The CPA framework derives from existing agent-based teamwork theories and is implemented on top of R-CAST, a novel agent architecture composed of multiple functional modules such task manager, knowledgebase, and communication manager, etc. The framework and the implemented agent architecture were tested in designed experiments simulating a hurricane relief scenario. The experiments focused on

evaluating two critical features of the CPA-based agents: the market-based adaptive resource allocation and the utility-based alternative method selection. Experimental results show that the two features can significantly improve the adaptation performance by enhancing the utilization of the limited resources.

5.1 Contributions

In Chapter 1, we introduced two research questions: 1) how an agent can efficiently shares information about the change and its impacts with other teammates in a timely way and 2) how a team of agents can effectively collaborate on an optimal solution for resolving resource conflicts in adapting existing plans.

In answering the two research questions, this research has made the following major contributions. First, we proposed a theoretical framework for collaborative plan adaptation with resource constraint. Different functional components are integrated within this framework to support the adaptation process in a distributed way. With this framework, agent-based resource coordinators can share relevant information (such as utility, resource status, and resource requirement, etc), and collaborate with each other to reach a global agreement on how to allocate resources among distributed tasks to achieve the maximum global utility. The framework enables a team of agents to collaboratively resolve conflicts among competing resource needs to adapt plans to the changing environment, based on assessing tradeoffs among those competing resource needs. More specific, it uses combinatorial auctions for such a team of agents to exchange information about opportunity cost. And an agent assesses its *opportunity cost* for offering a resource

assigned to a task T_i by considering the tradeoff between alternative ways to accomplish T_i.

Second, we built resource coordinator agents on top of a novel multi-agent architecture R-CAST with extended functions. Each resource coordinator is responsible for managing resources for its assigned task and interacting with other resource coordinators in the adaptation process.

Third, we implemented an agent-based combinatorial auction mechanism for a team of resource coordinators to exchange relevant information efficiently. A modified algorithm for solving the winner determination problem was implemented for a resource coordinator, serving as an auctioneer, to decide the resource reallocation aiming to maximize the global utility based on the limited resources.

Last, an algorithm for a resource coordinator to evaluate alternative methods in a task and to select the right one was developed and implemented. Given multiple alternative methods to accomplish a task, the algorithm enables a resource coordinator to assess the opportunity cost of offering resources to other coordinators considering tradeoffs between those alternative methods.

Those contributions were verified by experiments described in Chapter 4. The result shows that limited resources can be dynamically allocated to distributed tasks in adapting to resource-related changes through the collaboration of multiple resource coordinators. The market-based mechanism makes it possible to fully utilize such limited resources to achieve a maximized global utility. And the alternative method selection

algorithm provides the flexibility and the optimality for completing a task from a broader view.

5.2 Future Work

This research completed the first step to solve distributed plan adaptation problems considering the resource constraint. However, there are still many issues that not fully addressed. This section describes some of those problems that need to be covered by research in the near future.

In the section 3.5.3, we discussed that the alternative method reasoning can lead to a very complicated situation, if each coordinator considers every possible alternative in the projected world. The number of possible solutions will increase exponentially as the number of steps that an agent looks ahead increases. Thus, the computation cost and the required time for a resource coordinator to assess all possible alternatives will soon be out of control. Even though more foresights can contribute to finding the optimal solution, it is expected to find a balanced solution considering both the optimality and the computation cost.

Similar to the issue addressing alternative methods, another challenging issue to be explored is resource dependency reasoning. Resource dependency embodies substitutability between different sets of resources. The behind idea is that a resource item can be produced via certain actions from a set of other resource items. However, it will unavoidably introduce the complexity of AI planning along with the benefit that the solution can be more adaptive.

Load balance is another issue to be considered. The algorithm for solving WDP takes considerable computational power of an agent, especially when large-scaled tasks are involved and the number of bids is huge. If there is no mechanism for load control, a designated agent serving as the auctioneer can easily get overloaded. As a result, the auction decision cannot be made in time and the adaptation will be terminated unsuccessfully. It is desired to keep monitoring the workload of each agent and to dynamically assign the auctioneer role to an agent with less workload.

At last, experiments in this research are relatively simple comparing to real-world domains. In order to show the scalability and domain independence of the proposed framework, more experiments involving more types of resources, in more complex cross domains are expected to be designed and conducted.

Bibliography

1. J. Hendler, A. Tate, and M. Drummond. AI Planning: Systems and Techniques. *AI Magazine*, vol. 11, pp. 61-77, 1990.

2. S. Hanks and D. S. Weld. A Domain-Independent Algorithm for Plan Adaptation. *Journal of Artificial Intelligence Research*, pp. 319 - 360, 1995.

3. The Intelligence Software Agents Lab in Carnegie Mellon University. http://www.cs.cmu.edu/~softagents/multi.html

4. K. Decker, K. Sycara, and M. Williamson. Intelligent Adaptive Information Agents. *Journal of Intelligent Information Systems*, vol. 9, pp. 239 - 260, 1997.

5. G. Rabideau, R. Knight, S. Chien, A. Fukunaga, and A. Govindjee. Iterative Repair Planning for Spacecraft Operations in the ASPEN System. *International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS)*, Noordwijk, The Netherlands, 1999.

6. V. Lesser. A Retrospective View of FA/C Distributed Problem Solving. *IEEE transaction. Systems, Man, and Cybernetics*, vol. 11, pp. 1,347 - 1,362, 1991.

7. R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, vol. 2, pp. 189 - 208, 1971.

8. S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

9. A. Tate. Project Planning Using a Hierarchic Non-linear Planner. *D.A.I. Research Report*, No. 25, August 1976.

10. A. Tate. Generating Project Networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pp. 888-893, Boston, Mass. USA, August 1977.

11. J. S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, pp. 103-114, 1992, Boston, MA. Morgan Kaufmann.

12. D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 634-639, 1991, Anaheim, CA.

13. A. Barrett, S. Soderland, and D. S. Weld. Effect of step-order representations on planning. *Technical report 91-05-06*, 1991.

14. R. Fikes, P. Hart and N. Nilsson. Learning and executing robot plans. *Artificial Intelligence*, 3:251-288, 1972.

15. M. Fox. ISIS: a retrospective. In *Intelligent Scheduling*, Morgan Kaufmann, 3-28, 1994.

16. K. Baker. *Elements of Sequencing and Scheduling*. 1998.

17. M. Pinedo. *Scheduling Theory, Algorithms and Systems*. Prentice Hall. 1995.

18. J. Beck and M. Fox. A generic framework for constraint-directed search and scheduling. *AI Magazine,* 19(4), 101-130, 1998.

19. J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the 12th National Conference on AI,* 1092-1097, 1994.

20. S. Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the 11th National Conference on AI,* 139-144, 1993.

21. S. A. Wolfman and D. S. Weld. The LPSAT engine and its applications to resource planning. In *Proceedings of the sixteen International Joint Conference on Artificial Intelligence (IJCAI-99),* San Mateo, CA, 1999.

22. J. Koehler. Planning under resource constraints. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI-98),* 1998.

23. B. Drabble and A. Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In *Proceedings of the 2nd International Conference on AI Planning Systems,* 243-248, 1994.

24. P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proceedings of the 14th International Joint Conference on AI,* 1643-1649, 1995.

25. D. Smith and D. Weld. Temporal planning with mutual exclusion reasoning. In *Proceedings of the 16th Joint Conference on AI,* 326-333, 1999.

26. S. Vere. Planning in time: windows and durations for activities and goals. Pattern *Analysis and Machine Intelligence* 5, 246-267, 1983.

27. N. Muscettola. HSTS: integrating planning and scheduling. In *Intelligent Scheduling*, Morgan Kaufmann, 169-212, 1994.

28. J. Allen and J. Koomen. Planning using a temporal world model. In Proceedings of the 8th International Joint Conference on AI, 741-747, 1983.

29. R. v. d. Krogt, M. d. Weerdt, and C. Witteveen. A Resource Based Framework for Planning and Replanning. Presented at *IEEE/WIC International Conference on Intelligent Agent Technology*, Halifax, Canada, 2003.

30 M. D. R-Moreno, A. Oddi, D. Borrajo, A. Cesta and D. Meziat. IPSS: a hybrid reasoner for planning and scheduling. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04)*, pp. 1065-1066, 2004.

31. J. Blythe. An Overview of Planning under Uncertainty. *AI Magazine*, vol. 20, pp. 37-54, 1999.

32. N. Kushmerick, S. Hanks, and D. Weld. An Algorithm for Probabilistic Planning. *Artificial Intelligence*, vol. 76, pp. 239-286, 1995.

33. P. E. Friedland and Y. Iwasaki. The Concept and Implementation of Skeletal Plans. *Journal of Automated Reasoning*, vol. 1, pp. 161-208, 1985.

34. M. Veloso, J. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe. Integrating Planning and Learning: The Prodigy Architecture. *Journal of Experimental and Theoretical AI*, vol. 7, pp. 81-120, 1995.

35. L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, vol. 4, pp. 287-339, 1996.

36. J. Blythe. Decision-theoretic Planning. *AI Magazine*, vol. 20, 1999.

37. K. Decker, K. Sycara, and M. Williamson. Intelligent Adaptive Information Agents. *Journal of Intelligent Information Systems*, vol. 9, pp. 239 - 260, 1997.

38. M. Veloso and J. Carbonell. Derivational analogy in PRODIGY: Automating case acquisition, Storage and Utilization. *Machine Learning*, 249-278, 1993.

39. L. Ihrig and S. Kambhampati. Plan-space vs. State-space planning in reuse and replay. *Technical report*, Arizona State University, 1996.

40. H. Munoz-Avila. Case-based maintenance by integrating case index revision and case retention policies in a derivational replay framework. *Computational Intelligence* 17, 2001.

41. B. Nebel and J. Koehler. Plan reuse versus plan generation: a theoretical and empirical analysis. *Artificial Intelligence*, 76, 427-454, 1995.

42. T. Au, H. Munoz and D. Nau. On the complexity of plan adaptation by derivational analogy in a universal classical planning framework. In Proceedings of the 6th European Conference, pp 13-27, 2002.

43. S. Bhansali, M. T. IIarandi. When (not) to use derivational analogy: lessons learned using APU. In Aha, D., ed. In *Proceedings of AAAI-94 Workshop: Case-based Reasoning*, 1994.

44. B. Blumenthal and B. Porter. Analysis and Empirical Studies of Derivational Analogy. *Artificial Intelligence*, 67, 287-328, 1994.

45. K. Erol, D. Nau, and J. Hendler. HTN Planning: Complexity and Expressivity. In Proceedings of *the Twelfth National Conference on Artificial Intelligence*, pp. 1123-1128, 1994.

46. E. Ephrati and J. Rosenschein. Divide and Conquer in Multiagent Planning. In Proceedings of *the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 375-380, 1994.

47. E. Durfee and V. R. Lesser. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics*, pp. 63-83, 1991.

48. M. M. d. Weerdt and C. Witteveen. A resource logic for multi-agent plan merging. In *Proceedings of the 20th Workshop of the UK Planning and Scheduling Special Interest Group*, 244-256, 2001.

49. J. F. Horty and M. E. Pollack. Evaluating new options in the context of existing plans. *Artificial Intelligence,* 127(2), 199-220, 2001.

50. J. S. Cox and E. H. Durfee. Discovering and exploiting synergy between hierarchical planning agents. In *Proceedings of AAMAS 03*, 2003.

51. Q. Yang. *Itelligent Planning*, Springer-Verlag, Berlin, 1997.

52. H. E. Durfee. *Coordination of Distributed Problem Solvers*: Kluwer Academic Publisher, 1988.

53. V. Lesser. Cooperative Multiagent Systems: A Personal View of the State of the Art. in *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, 1999.

54. V. Lesser, K. Decker, T. Wagner, A. Garvey, and B. Horling. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems,* vol. 9, pp. 87 - 143, 2004.

55. V. Lesser, K. Decker, N. Carver, A. Garvey, D. Neiman, M. Nagendra Prasad, and T. Wagner. Evolution of the GPGP Domain-Independent Coordination Framework. *CMPSCI 98-05*, 1998.

56. D. E. Wilkins and K. L. Myers. A Multiagent Planning Architecture. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 1998.

57. K. Decker. TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. In *Foundations of Distributed Artificial Intelligence*, 429–448, 1996.

58. N. R. Jennings and M. J. Wooldridge. Applications of Intelligent Agents. *Agent Technology: Foundations, Applications, and Markets*, 3-28, 1998.

59. N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Int Journal of Autonomous Agents and Multi-Agent Systems*, vol. 1, 7-38, 1998.

60. M. Wooldridge. *An introduction to Multiagent Systems*. Chichester, England: John Wiley & Sons, 2002.

61. J. Yen, J. Yin, T. R. Ioerger, M. Miller, D. Xu, and R. A. Volz. CAST: Collaborative Agents for Simulating Teamwork. *In Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 1135-1142, 2001.

62. J. A. Cannon-Bowers and E. Salas. A framework for developing team performance measures in training. In *Team Performance Assessment and Measurement: Theory, Research and Applications*, 45-62, 1997.

63. M. R. Endsley and W. M. Jones. A model of inter- and intrateam situation awareness: Implications for design, training and measurement. In *New trends in cooperative activities: System dynamics in complex environments*, 46-47, 2001.

64. M. S. Miller, J. Yin, R. A. Volz, T. R. Ioerger, and J. Yen. Training teams with collaborative agents. In *the Fifth International Conference on Intelligent Tutoring Systems*, 63-72, 2000.

65. J. Yen, X. Fan, S. Sun, R. Wang, C. Chen, K. Kamali, M. S. Miller and R. A. Volz. On modeling and simulating agent teamwork in CAST. In *the 2nd International Conference on Active Media Technology*, Chongqing, P. R. China, 18-33, 2003.

66. M. E. Bratman, D. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349-355, 1988.

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

68. A. S. Rao and M. P. Georgeff. BDI Agents: from theory to practice. In *Proceedings of the First International Conference on Multiagent Systems*. 1995.

69. P. R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. In *Intentions in Communication*, P. Cohen, Ed. Cambridge. MA: MIT Press, 221-255, 1990.

70. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3), 213-361, 1990.

71. P. R. Cohen and H. J. Levesque. Teamwork. In *Handbook of Multi-agent Systems*, 487-512, 1997.

72. B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 269-358, 1996.

73. B. Grosz and S. Kraus. The Evolution of SharedPlans. In *Foundations and Theories of Rational Agencies*, A. Rao and M. Wooldridge, Eds., 227-262, 1999.

74. M. Tambe. Toward Flexible Teamwork. *Journal of Artificial Intelligence Research*, vol. 7, 1997.

75. J. Yen, X. Fan, and R. A. Volz. On Proactive Delivery of Needed Information to Teammates. In *Proceedings of the AAMAS 2002 Workshop of Teamwork and Coalition Formation*, Italy, 2002.

76. J. F. Kurose and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Transactions on Computers*, 38(5):705-717, 1989.

77. P. Cramton, Y. Shoham, and R. Steinberg. Introduction to Combinatorial Auctions. *Combinatorial Auctions*, MIT Press, 2005.

78. L. Jimsberger and B. Grosz. A combinatorial auction for collaborative planning. In *Proceedings of ICMAS-2000*, 2000.

79. W. Walsh, M. Wellman, and F. Ygge. Combinatorial auctions for supply chain formation. ACM Conference on Electronic Commerce, 2000.

80. T. Sandholm and S. Suri. BOB: Improved winner determination in combinatorial auctions and generalizations. In *Artificial Intelligence,* 2003.

81. A. Andersson, M. Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. In *Proceedings of the 4th International Conference on Multiagent Systems*, pages 39-46, 2000.

82. T. Sandholm, S. Suri, A. Gilpin, and D. Levine. CABOB: A fast optimal algorithm for combinatorial auctions. *International Joint Conference of Artificial Intelligence,* pages 520–526, 1999.

83. X. Fan, S. Sun, M. McNeese, and J. Yen. Extending the recognition-primed decision model to support human-agent collaboration. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and Multiagent systems,* pages 945-952. ACM Press, 2005.

84. X. Fan, J. Yen, and R. A. Volz. A theoretical framework on proactive information exchange in agent teamwork. *Artificial Intelligence,* 169:23-97, 2005.

85. G. A. Klein. Recognition-primed decisions. In W. B. Rouse, editor, *Advances in man-machine systems research*, volume 5, pages 47-92. JAI Press, 1989.

86. X. Fan, R. Wang, S. Sun, J. Yen, and R. A. Volz. Context-Centric Needs Anticipation Using Information Needs Graphs. *Journal of Applied Intelligence,* Vol. 24, No. 1, 2006.

87. D. R. Henderson. *The Concise Encyclopedia of Economics*. Indianapolis: Liberty Fund, Inc., ed. David R. Henderson, 2002.

88. T. Mullen, V. Avasarala, and D. L. Hall. Customer-Driven Sensor Management. *IEEE Intelligent Systems*, 21(2): 41-49, March/April 2006.

89. R. Zlot, A. Stentz, M. B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings of the International Conference on Robotics and Automation,* 2002.

90. M. B. Dias, D. Goldberg, and A. Stentz. Market-based multirobot coordination for complex space applications. In *the 7th International Symposium on Artificial Intelligence,* Robotics and Automation in Space(i-SAIRAS), 2003.

91. M. B. Dias and A. Stentz. A free market architecture for distributed control of a multirobot system. In *6th International Conference on Intelligent Autonomous Systems (IAS-6),* pages 115-122, 2000.

92. R. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12), 1980.

93. M. D. McNeese. An Ecological Perspective applied to Multi-Operator Systems. *Human Factors in Organizational Design and Management*, pp. 365-370, 1994.

94. T. Dean, R. J. Firby and D. Miller. Hierarchical Planning involving Deadlines, Travel time, and Resources. *Computational Intelligence,* Vol. 4, No. 4, pp. 381-398, 1988.

Appendix A

Agent Configuration Example

```
#########################################################################
#########
#                 Agent configuration file for agent "agent2"
#########################################################################
#########

############ about the agent overall configuration
##############################
# the agent name, must be unique for each agent
agentName = agent2

agentIcon = ./images/agent.gif

# the components of the agents
agentComponents = kbImpl comImpl processImpl domainImpl imImpl taskImpl
resourceImpl

# to show gui or not, if false no gui will be shown,
# even you decide to use a component gui
useGUI = true

# the relative speed, will affect all component锟絤 speed
# e.g. effects: speed = cycleSpeed * processClock
cycleSpeed = 1.0

# initial status of agent, stopped if choose "true"
isStoped = false

# the networks that this agent belongs to
agentNets = Gift_Team

agentNodeDisplay = null

displayedMessage = Count Inform

################### about request whiteboard ###################################
########## whiteboard is a built-in component and it is a must #################
# The thread cycle time for the whiteboard, in 1/1000 sec
whiteboardClock = 1000

# use whiteboard gui or not,
whiteboardGUI = true

################### about domain adapter ########################################
# The domain implementation, must be selected when using a process manager
```

```
domainImpl = edu.psu.domainadapter.ExampleDomainAdapter

# use domain gui or not, ExampleDomainAdapter doesn't have a gui
domainGUI = false

################## about the knowledge base ######################################
# The knowledge base implementation
kbImpl = edu.psu.activeknowledgebase.AKB

# the kb file for this agent
kbFile = ./test/test.kb

# the thread cycle time for the kb, in 1/1000 sec
kbClock = 1000

# show the kb gui or not
kbGUI = true

# if you want to hear a voice when query for natural reply
kbSpeakNatrualReply = false

defaultFactProviderDegree = 0.6

defaultFactNeederDegree = 0.7

################## about the resource manager  ###################################
# The resource manager implementation
resourceImpl = edu.psu.resource.ResourceManager

# the task file for this agent
resourceFile = ./test/task2.resource

# the thread cycle time for the tm, in 1/1000 sec
resourceClock = 1000

################## about the task manager  #######################################
# The task manager implementation
taskImpl = edu.psu.task.TaskManager

# the task file for this agent
taskFile = ./test/test.task

# the thread cycle time for the tm, in 1/1000 sec
taskClock = 1000

# show the task gui or not
taskGUI = true
```

```
################## about the recommender   ####################################
# The recommenderimplementation
recommendImpl = edu.psu.recommend.Recommender

# the task file for this agent
optionSets =  decisionImpl processImpl

# the thread cycle time for the recommender, in 1/1000 sec
recommendClock = 1000

# show the recommendgui or not
recommendGUI = true

# evaluator that evaluate evaluation condition set
recommendRevaluator = edu.psu.domainadapter.ads.ADSEvaluator

# recommend ((love ?person me))

################## about the process manager ###################################
# the implementation of the process manager
processImpl = edu.psu.process.ProcessManager

# the process specification file
processFile = ./test/test.process

# the initial process that the agent should execute, null for none
processInitialProcess = null

# the clock cycle for process manager, in 1/1000 sec
processClock = 2000

# if or not to show the process monitor
processGUI = true

# if you want to enable voice for speak operator
operatorSpeak = false

# true, multiple processes can be live and executing in parallel
multipleLiveRootProcess = true

# true, process will terminate at end; false, the process will repeat at the end
# false, then you should define termination conditions to terminate processes
processTerminateIfEnd = true

# true, a terminated process will be removed from the process set
# false, if you want to keep dead process (cost more memory)
processRemoveIfInactive = true
```

```
# if the simulation should be in a new (cloned) KB, or in the agent's current kb
# agent should simulate in a new KB
simulationNewKB = true

# if precondition is tested, will the result affect the execution of
# the simulation? true, the result will not affect; false, the failed
# precondition will stop the simulation
simulationRelaxPrecond = true

# if the precondition is going to be tested in simulation
simulationTestPrecond = true

# how many levels will simulation decompose a process
simulationDepth = 1

############## The Communication Manager ##########################################
# the implementation of the communication manager
comImpl = edu.psu.communication.CommunicationManager

# a directory implementation
dirImpl = edu.psu.communication.SimpleDirectory

# the directory specification, dirImpl must be able to read this file
dirFile = ./test/test.dir

# how many times the count message should be send for a ping function
pingNumber = 15

# the cycle time for communication, in 1/1000 sec
comClock = 1000

# if to display the communication manager
comGUI = true

################## The Information Manager ########################################
# the implementation of the information manager
imImpl = edu.psu.irp.InformationManager

# a default investigation strategy
defaultStrategyImpl = edu.psu.irp.strategy.InqueryOrientedInvestigationStrategy

# the cycle time for im, in 1/1000 sec
imClock = 1000

# to display the im gui or not
imGUI = true
```

Appendix B

Predefined Plans

Task Deliver_Food:

```
(plan Deliver_Food(?dest ?food ?deadline)
   (res-requirement (helicopter 2)(pilot 2)(loc ?start)
    alternatives (1 (boat 3)(driver 3)(loc ?start))
    )
   (termcondition  (current_loc  ?dest  =)(current_time  ?deadline
>=))
   (utility 500)
   (process
    (seq
    (load ?food)
    (move_to ?dest)
    (unload ?food)
    )
   )
```

Task Fix_Levee:

```
(plan Fix_Levee(?loc ?bags ?deadline)
  (res-requirement     (helicopter     2)(pilot     2)(sandbags
300)(loc ?start)
   alternatives (1 (truck 3)(driver 3)(sandbags 300)(loc ?start))
   alternatives (2 (boat 4)(driver 4)(sandbags 300)(loc ?start))
   )
  (termcondition    (current_loc    ?loc    =)(is_fixed    true)
(current_time ?deadline >=))
  (utility 700)
  (process
   (seq
   (load ?bags)
   (move_to ?loc)
   (unload ?bags)
   (fix ?loc)
   )
  )
```

Task Rescue_People:

```
(plan Rescue_People(?dest ?safe_place ?deadline)
   (res-requirement (helicopter 3)(pilot 3)(loc ?start)
    alternatives (1 (boat 1)(driver 1)(loc ?start))
    )
   (termcondition (current_loc ?safe_place =)(people_in_danger 0
=)(current_time ?deadline >=))
   (utility 1000)
   (process
    (seq
    (move_to ?dest)
    (move_people ?dest ?safe_place)
    )
   )
```

VITA

Rui Wang

College of Information Sciences and Technology

The Pennsylvania State University

Education

Ph.D. Information Sciences and Technology *Dec. 2006*

The Pennsylvania State University, University Park

B.S. Computer Science *June. 2001*

University of Science and Technology at China (USTC)

Experience

The Pennsylvania State University *Aug. 2001-Aug. 2006*

Research Assistant

Network Center at USTC *Aug. 1999-June. 2001*

Research Assistant and Instructor

Affiliations

Member of AAAI

Member of ACM

Member of IEEE