Université de Montréal

Collaboration in Multi-Agent System: Contract Net and Beyond

Par

Jian Hua Mai

Département d'Informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de M. Sc.
en M. Sc. Informatique

Août, 2005

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé :

Collaboration in Multi-Agent System:
Contract Net and Beyond

présenté par :
Jian Hua Mai

a été évaluée par un jury composé des personnes suivantes :

Professeur(e) Esma AÏMEUR
président-rapporteur
Professeur Jean Vaucher
directeur de recherche
Professeur(e) Sylvie HAMEL
membre du jury

Mémoire accepté le 11 janvier 2006

Resumé

Les agents et les systèmes multiagents sont utilisés de plus en plus pour résoudre des problèmes dans des environnements ouverts, dynamiques et hétérogènes. Un bon protocole d'interaction peut aider à améliorer la performance globale de tels systèmes et diminuer les temps d'exécution aussi bien que la consommation de ressources. Ce mémoire étudie la collaboration dans les systèmes multiagents et se concentre sur le protocole CNP (pour Contract Net Protocol) qui est facile à comprendre et largement utilisé.

Sur la base de l'analyse et de la comparaison du CNP original avec ses variantes, une nouvelle extension nommée le CNCP-Nivelé est proposée dans ce mémoire. Ce nouveau protocole, qui combine et étend les variantes de CNP, vise à améliorer la flexibilité et réduire l'utilisation de ressource sans diminution de l'efficacité de temps propre à CNP.

Des tests avec une implémentation du protocole en JADE (une plate-forme multi-agent largement répandue) ont été démontré que notre protocole fonctionne tel que prévu et qu'il est surtout approprié pour des systèmes où les agents arrivent et quittent de façon imprévisible. Notre étude suppose une pénalité nulle pour les bris de contrats. En pratique, ceci pourrait réduire les avantages de notre protocole ; mais nos résultats expérimentaux illustrent qu'avec un usage approprié, le CNCP-Nivelé reste légèrement supérieur au Protocole Contrat Net original et ses extensions.

Les mots clés: le Système Multiagents, la Collaboration, le Protocole d'Interaction, le Contrat Net, le Nivelé-Engagement, CNCP Nivelé, CNCP

Abstract

Agents and Multi-Agent Systems are being increasingly used to solve problems in dynamic, heterogeneous and open environments. A good interaction protocol is critical in such applications; it can improve the overall system performance and lower execution times, as well as reduce the consumption of system resources. This thesis studies the collaboration in Multi-Agent Systems and concentrates on the Contract Net Protocol which is easy to understand and widely used.

On the basis of the analysis and comparison of the original Contract Net protocol and its variations, a new extension named Levelled-CNCP is proposed in this thesis. The new protocol, which combines and extends ideas from previous variants, aims to improve the flexibility and resource utilization efficiency without lowering the overall time efficiency.

Tests with an implementation in JADE (an open source multi-agent platform) show that the Levelled-CNCP can increase the flexibility and the resource utilization efficiency, especially in multi-agent systems where agents are not always present. Although the assumption that the zero penalty of decommitting from contract may be considered to affect the outcome, the experimental results illustrate that with careful implementing, this combinatorial protocol can bring us incremental improvement comparing to the original Contract Net protocol and its extensions.

Keywords: Multi-Agent System, Collaboration, Interaction Protocol, Contract Net, Levelled-commitment, Levelled CNCP, CNCP

Table of Contents

List of Tables

List of Figures

Acknowledgement

First of all, I want to thank my husband and my family from the bottom of my heart for supporting me passing through these special days. Even when I felt difficult to finish this thesis during my pregnancy and after I gave the birth of my baby, they encouraged me to take the challenge, and cared me so much. I am so grateful for their love and caring for my life.

I would like to appreciate my supervisor, Professor Jean Vaucher who gave me much advice about my study and his gentleness on accepting me after my previous supervisor left the department. Thank you also for advice on my life and encouragement, as well as those valuable critiques.

I would also like to thank all other people who have helped me.

Chapter 1 Introduction

In an open environment where agents [1] need to handle dynamic, complex and uncertain situations, such as in the competition of RoboCup [2], agent collaboration becomes a potential solution. This thesis focuses on the interaction protocol of agent collaboration which is one of the most important issues in agent applications. The Contract Net Protocol [3] (CNP) is well-known for its flexibility and robustness. It is widely used for task distribution and resource allocation in the AI area, especially in agent collaboration. The analysis on this protocol will be made in the following chapter, some existing extensions and adaptations to this protocol are also presented.

To illustrate how the CNP and its extension protocols work, the protocols are implemented to handle resource distribution problems [4] so that server resources can be optimally allocated to client requirements in a client-server network. The network resources to be considered here are those that are limited and exclusive (i.e. if one resource is occupied by a task, it cannot be occupied by other tasks).

1.1 Agent Technology

In these recent years, agent technology [1, 5] has been increasingly used in manufacturing management in industrial applications, business management in Electronic commerce, and many other areas that are related to Information technology. In the complex, dynamic and distributed domains, agent technology has its extraordinary advantages. This is why many researchers willing to continuously devote their efforts on improving agent technology.

1.2 Multi-Agent system

A Multi-Agent System [6] (MAS) is a collection of various kinds of agents, which

are working together to solve problems. Generally, data in a MAS are decentralized, and there is no centralized control. While having its advantages, the use of MAS brings many challenges such as how to coordinate a group of agents for achieving a common goal, how to enable agents to communicate and interact efficiently, and what communication language and protocol should be used, etc. Organizations like the Foundation of Intelligent Physical Agents (FIPA) have endeavoured to solve these problems.

1.3 Motivations

Since communication is one of the most important issues that ensure agents sharing their knowledge and resource, a certain set of rules about what and how the agents can communicate are also necessary to be defined when applying agent systems to solve problems. This set of rules is usually called a communication or interaction protocol.

As a flexible and robust interaction protocol, Contract Net Protocol (CNP) has been widely used for distributing tasks in many kinds of Multi-Agent System. Originally, it was designed for agents cooperating together to achieve some simple common goals. With more and more self-interested agents involved in agent systems, especially in business applications, the Contract Net Protocol needs to be revised. Some researchers have proposed their extensions or adaptations. Each extension has its own advantages and disadvantages. One may improve the efficiency but with lower flexibility; another may increase the flexibility while with lower efficiency. In this thesis, my goal is to analyze the Contract Net protocol and two successful revisions proposed by other researchers (see chapter 4 and 5), then try to combine the ideas from these two different existing extensions for realizing incremental improvement, comparing with applying just one of them. The experimental result shows that the combination makes the protocol more flexible without reducing the efficiency.

1.4 Overview of the thesis

Chapter 2 introduces some background information of this paper, including the concept of agent, Multi-Agent System and some issues involved in development in Multi-Agent System. Chapter 3 reviews the Contract Net Protocol (CNP), its advantages and drawbacks. Since my work is based on this protocol, I will give a detailed description. Chapter 4 shows the insight of one of CNP's extension, the Contract Net with Confirmation Protocol. Then based on some analysis; I propose a modification on this protocol. In chapter 5, I propose a combinatorial extension which is named CNCP with Levelled Commitment. Chapter 6 describes the implementation and the experiment design of the three protocols. Chapter 7 presents and analyzes the experimental results and discussions of the advantage and disadvantage of the protocols. The last chapter gives out the conclusion and pictures the future work that could be done.

Chapter 2 Background

This chapter provides some background information about the different concepts and technologies used in this thesis. Section 2.1 introduces the term agent, including its attributes. Section 2.2 describes the term of multi-agent system (MAS), as well as various issues and problems involved in the development of applications. The collaborations in multi-agent system will be described, as well as the comparison between two types of agent of MAS: the collaborative agent and the self-interested agent. Some generic interaction protocols for collaboration in multi-agent will also be introduced.

2.1 What is Agent

The concept of ***agent*** [1, 7] initially appeared in the research of Artificial Intelligence. There is no universally recognized definition for the term *agent.* Generally, an agent is a piece of software or a hardware, which should act intelligently like a human or react rationally to its perception. With agents being used more and more in a multitude of areas, there are multiplicities of roles agents can play. Thus, it is quite impossible and even very impractical to define "agent" in a simple way.

2.1.1 Attributes of Agent

Although none of the existing definitions on agent is recognized as a standard definition, there is a common agreement that a software intelligent agent usually has attributes of intelligence and acts on behalf of the user(s). In general, an intelligent agent can have the following important properties:

Autonomy: autonomy is an often cited characteristic of an agent. A definition given by Casterfranchi [8] is that *agents operate without the direct intervention of*

humans or others, and have some kinds of control over their actions and internal state.

Reactivity: Agents have the ability to selectively sense their environment (which may be the physical world, a collection of other agents, the Internet, or perhaps all of these combined), and respond in real time.

Adaptability: To operate and/or survive effectively in open environments agent systems must be adaptive. Adaptability of agents is the capability of an agent in response to unexpected events or dynamic environments [9]. Examples of unexpected events include the unscheduled failure of an agent, an agent's computational platform, or underlying information sources.

Social Ability / Cooperation Agents interact with other agents and human beings with some kinds of communication language [10]. Agents exchange their knowledge, beliefs and the plans to work together and solve larger problems, which are *beyond their individual capabilities* [7]. The user/owner of an agent specifies what actions the agent should be performed on his or her behalf, and the agent decides by itself how to perform and provides results. Agent cooperation can be also described as a form of ***collaboration***.

2.2 Multi-agent system (MAS)

The concept of multi-agent system is an outgrowth of the Distributed Artificial Intelligence (DAI) [4] — a sub-field of Artificial Intelligence. A multi-agent system is an aggregation of agents which are acting together to handle more complicate situations or to achieve a better result. Durfee et al. [11] defined a multi-agent system as "*a loosely-coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities*". Research in multi-agent systems is mainly concerned with how agents coordinate their knowledge, goals, skills, and plans jointly to take action or to solve problems.

Multi-agent systems provide a solution to systems that are decentralized and concurrent. Many applications of multi-agent systems are distributed, either spatially or functionally, thus a system made of autonomous agents will not collapse when one or more of its components fail since there won't be any single point of failure. The advantages offered by using multi-agent systems also include *Scalable architecture, Self-Configuring Systems, Faster Problem Solving, Flexible systems etc [12].*

2.2.1 The various types of agents involved in MAS

In the agent world, there are many different types of agents. According to the kind of owners they represent, agents may be distinguished into two types: collaborative agents, and self-interested agents. Following, let's see the detailed definitions of these two types.

> **Collaborative Agents** These agents emphasize autonomy and cooperation (with other agents) in order to perform tasks for their owners. *Their key attributes include autonomy, social ability, responsiveness and pro-activeness. In order to have a coordinated setup of collaborative agents, they may have to negotiate in order to reach mutually acceptable agreements in some matters* [13]. For example, robots of each team in RoboCup are collaborative agents.
>
> **Self-interested agents** These agents are designed by independent designers and have their own agenda and motivation. The self-interested agents [14, 15] just concern with the benefit and performance of the individual agent. For example, bidders in a bidding system are self-interested agents.

2.2.2 Collaborations in multi-agent systems

One of the key problems for multi-agent systems to solve is how to make the agents communicate with each other in order to successfully carry out tasks. With the help of computer networks, one can expect to get higher computational power and robustness by a set of distributed computers or multiprocessors. Since knowledge sources are not centralized, multiple agents must collaborate to achieve the goal(s). With multi-agent collaboration (MAC), one can solve complex and distributed problems. Wilsker [16] gives a comparison of several different collaborations widely used in multi-agent system.

1) Joint intentions

The ***Joint Intentions model***, proposed by Cohen and Levesque [17], is one of the first attempts to establish a formal theory of multi-agent collaboration. This model presents a shared responsibility towards other team members. Communication is required in this model. An intention-commitment is the precondition for agents' collaboration. Private beliefs should be mutually known regarding the team's commitment to achieve the joint goals.

2) Shared Plans

B. Grosz and C. Kraus' ***shared plans model*** [18] is a theory of collaborative planning. Working with this model, an action can be decomposed into multi-levels hierarchies and performed by two or more agents. There is no request to establish a commitment. No agent has an entire roadmap for the full plan activities throughout the course of their collaboration. In contrast to the joint intentions model, there is no communication requirement.

3) Planned Team Activity

Different from the above two collaboration models, ***planned team activity*** [19] means that an agent designer supplies plans to achieve some specific goals in advance. Agents have complete knowledge of full plans prior to their committing to join a team. Therefore, agent behaviour is bounded and predicable, but the collaboration is quite brittle in an unpredictable environment. Here, communication is necessary to inform any failure or abandonment while not including the private believes as in the *joint intention model.*

4) Contracting

Pollack et al. made a provision of contracting in their shared plans theory [20]. For reasons of convenience or lack of necessary knowledge, an agent may contract out some tasks. Nevertheless, this provision is limited for agents to commit towards some joint goal. With a broader view, contracting can solve some collaboration problem that the above three models cannot. In the case of individual agents having different intentions or even two or more competitive agents having the conflictive goal, collaboration needs to be modeled at the agent level rather than at the team level. Compared with the previous three models, only the contracting model can lead to a proper solution in that situation.

As the development of the global communication technology and applications of agent technology, agents working in open and dynamic systems are more and more widely used in tasks distribution and automating E-marketing (e.g. the automated supply chain formation.). Actually, this is why the contracting model has become more and more important for researchers to simulate today's e-market applications.

2.2.3 Interaction protocols for agent collaborations

The process of interaction in agents' collaboration allows several intelligent agents to combine their knowledge and skills and make the multiple agents work together effectively. The interaction protocol, an important issue involved in collaboration will now be discussed in detail.

Defining the interaction protocols is part of the specification of a multi-agent system. Generally, the definition of an interaction protocol for multi-agent system includes the following factors:

1) *A communication pattern* [21], which specifies what (message) goes first, what goes in the second, and so on, defines an allowed sequence of messages between agents, as well as the constraints on the content of the messages.
2) *A semantics that is consistent with the communicative acts* [21] within a communication pattern. It specifies the corresponding action(s) of any agent after receiving a message from another agent.

Here, the protocols constrain the exchanged messages' order, content and type during the communication process.

Foundation of Intelligent Physical Agents (FIPA) models the interaction protocols with the Unified Modeling Language (UML). In the following chapters, all the interaction protocols will also be presented with UML.

As one of the standardized interaction protocols by FIPA, the high flexibility and robustness allow the Contract Net Protocol[1], which is a set of conversation polices for allocation resources through a "call for bids" interaction process, to handle agents' communication in a dynamic environment. The comprehensive review of

[1] This will be described in more detailed in chapter 3.

this protocol, which is commonly used for collaboration in MAS, will be described in the next chapter.

2.3 Multi-Agent Platform-JADE

To compare the performance, the protocols that are presented in this thesis are implemented in JADE [22] — a FIPA compliant agent development framework which is implemented in the Java language and realized by Telecom Italia Lab.

JADE intends to simplify the implementation of multi-agent systems by providing a platform that fully complies with the FIPA specifications. The framework provides a set of tools that supports the debugging and deployment phase. At runtime, the platform can be distributed over a computer network. The benefit of implementing in Jade is that:

i. It's easy to start – anyone who knows about Java programming can easily get his/her hands dirty;
ii. It's fully complying with FIPA specification – any concept based on some FIPA specifications can be easily realized;
iii. It's open source and free – there's no need to pay for getting the platform; it's free to read the source code for better understanding it's structure, and use its class modules as examples[2];
iv. It's configurable at runtime – this property is good for testing the protocols because agents can be added, deleted, and moved across the network at runtime.

JADE uses Behaviour abstraction to model the tasks that an agent can perform and agents instantiate their behaviours according to the needs and capabilities. Behaviour is an abstract class that provides the skeleton of the elementary task to

[2]In fact Jade's behaviour *jade.proto.**ContractNetInitiator*** has been used as the major example for writing interactive behaviours in the implementation of all protocols and will be described in more detail later.

be performed [15]. An agent may have one or more behaviours.

JADE implements a base **Agent** class which handles message transports and threading issues. The agent developer should extend the **Agent** class and implement the agent specific tasks through handling one or more Behaviour classes [22] by *addBehaviour(..)* and *removeBehaviour(..)* function calls.

JADE provides the following behaviour classes:

- **SimpleBehaviour** must be used to implement atomic actions of the agent work.
- **ComplexBehaviour** defines a method *addBehaviour* and a method *removeBehaviour*, allowing the agent developer to create complex behaviour composed of several sub-behaviours.
- **OneShotBehaviour** is an abstract class that models atomic behaviours that must be executed only once.
- **CyclicBehaviour** is an abstract class that models atomic behaviours that will be executed continually.

Here is a simple example of JADE showing how to create an agent and add a **MyBehaviour**, an extension of **SimpleBehaviour** to it.

```
public class Manager extends Agent implements myOntology {

   protected void setup() {
   /* initialize the agent and register it to the JADE control platform
*/
       setMyOntology("myontologyname");
       setMyType("mytype");
       setMyOwner("myownername");
       register ( getMyType() );

       /* create and start a contractNet protocol initiator behaviour
*/
       //
       Behaviour b;
       ACLMessage cfp = newCFPMsg();  // create a CFP message object
       b = new MyBehaviour(this);//create a behavior object
       addBehaviour(b);   // add the object to the agent
```

```
    } // End of setup()

    .........

    /* specify your own behaviour requirement */
    class MyBehaviour extends SimpleBehavior {
       public void action(){
          ......
       }
       ......
    }
}
```

Chapter 3 Contract Net Protocol (CNP)

In 1980, G. Smith developed a model called Contract Net Protocol (CNP) [3] for task distribution and resource allocation. Smith built the model within the traditional collaboration concept of Distributed Artificial Intelligence. The CNP model defined a decentralized method by which collaborative agents work together to complete a complex, decomposable task through a process of bidding and tasks assignment. After that, much of the fundamental work for the interaction of multi-agent systems is based on this protocol.

3.1 Original Contract Net Model

CNP is a high-level definition without considering the physical architectures and implementation details. It has been widely used in real world applications such as scheduler for real-time control and supply chain formation in AI and MAS area. The original CNP defines a 1:N structure in which a task holder tries to assign its task(s) to a number of participants that are able to execute the task(s). As a protocol, the flow and characteristics of inter-agent coordination and task distribution of CNP follow a certain series of interaction patterns. Figure 3.1 below shows an example series of interactions to tell how CNP works.

In Figure 3-1, the "square face" agent, called the "manager", is the one that has a task to be assigned to other agent. The "round face" agents, called the "suppliers", are the potential contractors that can do the task. Notice that usually tasks can have some constraints. These constraints for contractor are addressed out when announcing the tasks. Some of the agents who have received the task announcement can't meet the requirements such as the agent presented by dashed line in figure 3-1, and then it will quit the bidding process. In step 4 of Figure 3-1, the contract is assigned to the "supplier" who gives the best bid. In such case this "supplier" becomes a contractor.

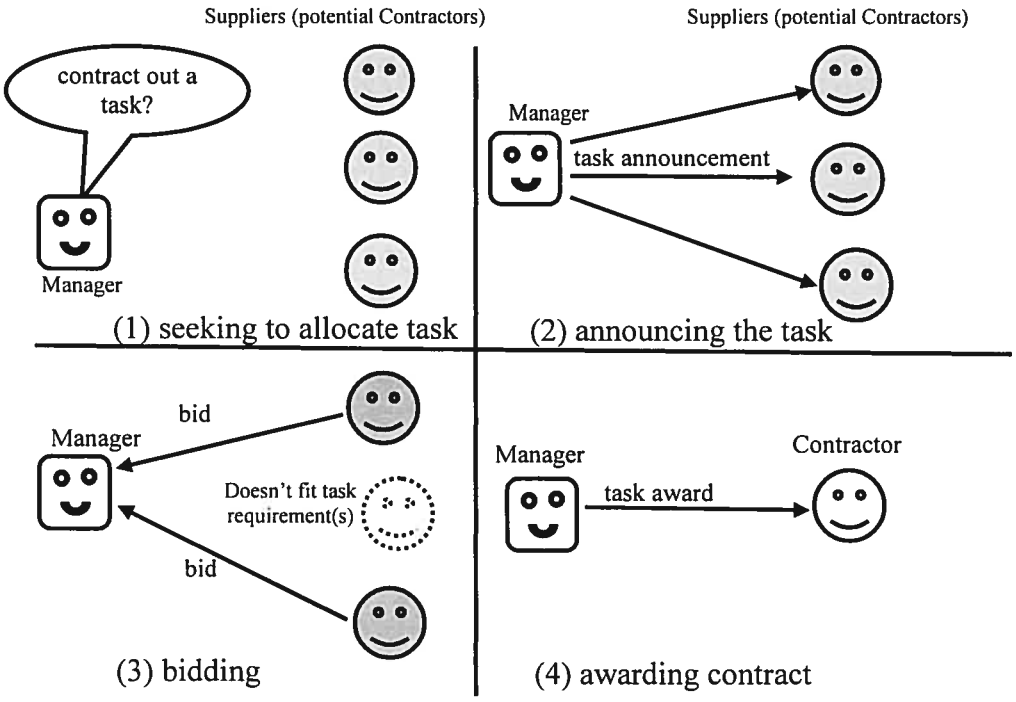


Figure 3-1 An example of the original Contract Net Protocol

The above illustrated process can be simplified into four phases:

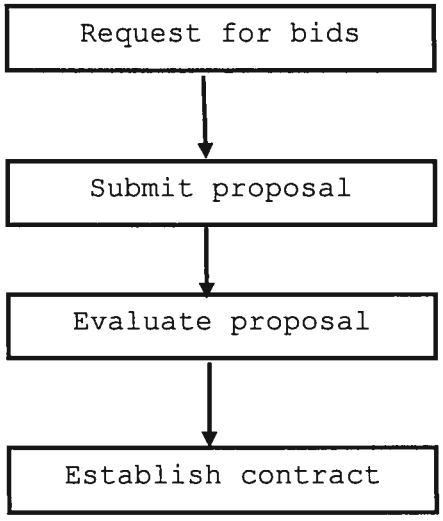


Figure 3-2 Diagram of the four phases for CNP

The first phase and the second phase can be done in parallel. The manager can send

the requests for bid to multiple suppliers at the same time, and the suppliers can give back proposals simultaneously.

In general, the Contract Net model works in the following way:

- A *manager* agent in a Contract Net decomposes a large task into a set of subtasks, and then *advertises* the subtasks to other agents in the net.
- The candidate participants who are willing to complete the subtasks respond the advertisement by *bidding.*
- The *manager* then selects proper candidates according to their *bids* and assigns the subtasks to them. The selected candidate(s) are called *contractors*.
- A contractor is expected to complete the assigned task. After a contractor accepts an assignment, it may further divide it into a set of smaller subtasks, and advertise them in the same manner as the original manager did.

The task assigned can be decomposed and contracted out recursively. In such a case, a contractor (supplier) can also be a manager simultaneously. While the contractor is working on its task, the manager may interact with it to receive progress reports, to cancel an in-progress task, or to receive results of the contractor's work.

To make CNP practical for using in MAS, FIPA has standardized CNP with a set of specifications. The FIPA's CNP will be described in the next section.

3.2 FIPA Contract Net Interaction Protocol

To make the CNP suitable to an agent-based system, FIPA adds *rejection* and *confirmation* communicative acts into the original proposed model. Furthermore, FIPA adds deadlines to all communicative acts. In this revised protocol[3], an agent,

[3] see www.fipa.org/specs/fipa00029/XC00029G.html#_Toc23830138

named **initiator**, takes the role of manager, which desires to have a task performed by another agent in the system and further wishes to optimise some characteristics of the task such as price, time consumption. For a given task, each potential contractor, now named **responder**, may either respond with a *propose* message, or respond with a *refuse*. The initiator can then select a best responder from those that sent out proposals and continue to set up a contract.

Since this thesis concentrates on multi-agent systems, the Contract-Net Protocol and all of its extensions presented in the following chapters are all based on the standardized FIPA CNP specification.

The UML diagram of FIPA CNP is shown in figure 3-3. In this diagram, the **initiator** sends out solicitations for proposals to other agents by FIPA ACL[4] message "*Call for Proposals*" (CFP), which specifies the task as well as any conditions (precondition) that the initiator places upon the execution of the task. The potential contractors (**responders**) receiving the *CFP* generate responses in which part of them are *proposes* and others are *refuses*.

The **responder**'s proposal includes the preconditions to be set out for the task, which may be the price, timetable when the task will be done, etc. Once the **deadline** passes, the **initiator** evaluates the received bids and selects an agent to perform the task that optimises its goals. The selected agent will be informed by "*accept-proposal*" message and the remaining agents will receive a *reject_proposal* message. Once the contractor has completed the task, it sends a message "inform-*done*" to the **initiator**. However, if the contractor fails to complete the task, a "*failure*" message will be sent.

[4] FIPA ACL (agent communication language) is a message-oriented agent communication language.

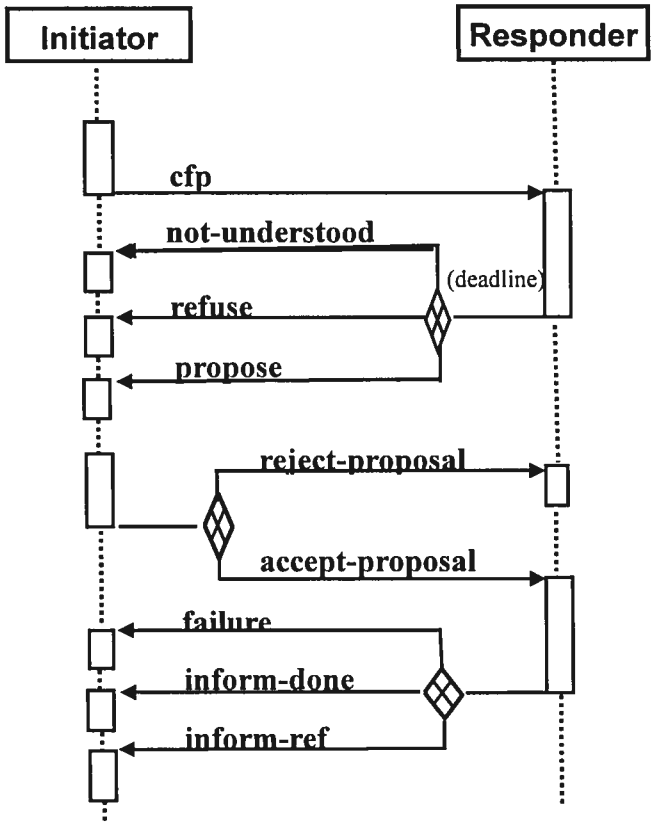
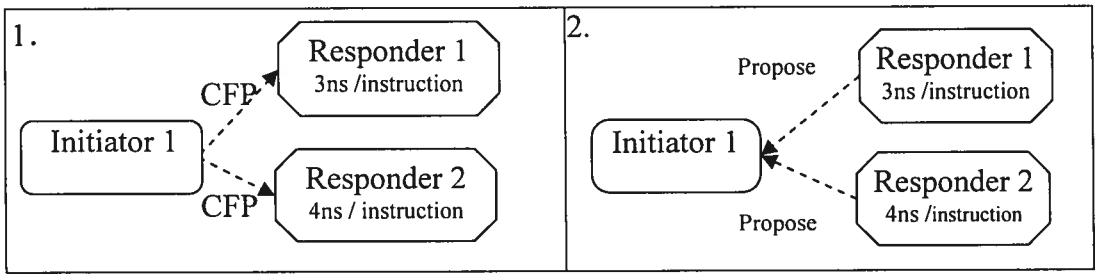


Figure 3-3 UML Diagram of FIPA contract net interaction protocol

To see how CNP can be applied in real applications, a typical example of CNP is shown in Figure 3-4 which illustrating the process of allocating a computational resource to a task in a client-server network. In Figure 3-4, **Initiator1** is the client agent holding a task which needs certain amount of computing; **Responder1** and **Responder2** are the server agents representing two computational resources with different processor speed. One can see that in this situation, CNP can successfully allocate the faster server (**Responder1**) to handle the task.



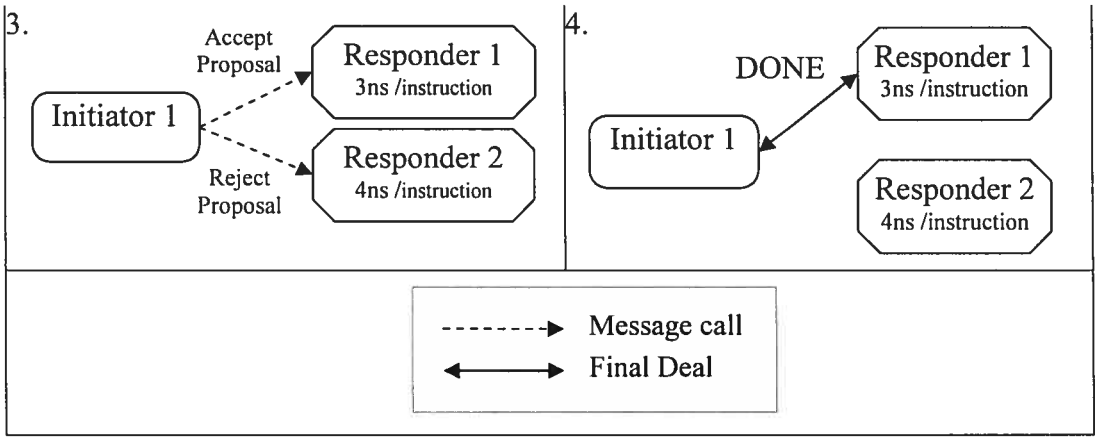


Figure 3-4 An example of contract net protocol

In CNP, either side of a communication can inform the other side that it did not understand the received message. This is achieved by returning a *not-understood* ACL message. The situation of "not-understood" can occur at any point in the interaction protocol.

The deadline is specified by the *reply-by* parameter in the ACL message to prevent the **Initiator** from waiting for all replies indefinitely in case of some Responders fail to reply in time. Zoltan and Prasenjit cited the importance of the deadline in their research report [23], where they analyzed the scalability and performance of the CNP in JADE development environment. According to their experimental results, the performance of the protocol (CNP) depends largely on the value of the deadline [23, 24]. In this regard, there will be more discussion in the section 7.4 of this thesis.

Figure 3-5 shows the state diagram of FIPA CNP. Here, one can clearly see that this protocol is a pattern for a simple interaction type. Elaboration on this protocol will almost certainly be necessary in order to specify all cases that might occur in an actual agent interaction.

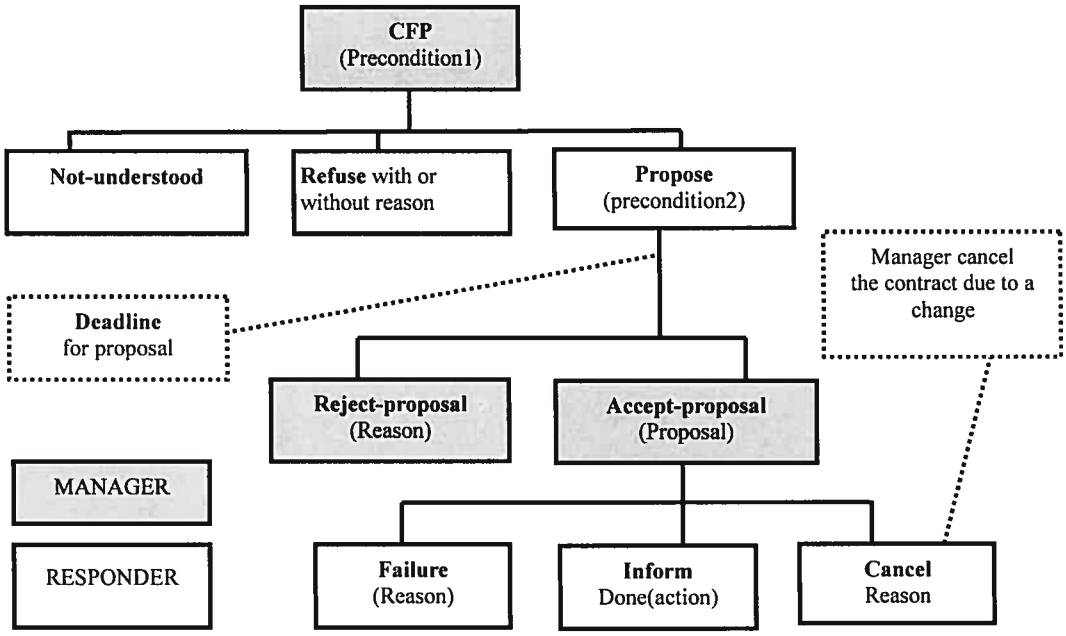


Figure 3-5 State diagram of FIPA CONTRACT NET Protocol

3.3 CNP for interactions involving different types of agents

As mentioned in the previous chapter, there are two types of agents – collaborative agents and self-interested agents. Different interaction comes with different types of agents. When applying CNP in MAS that consist of collaborative agents, since the agents have a compatible goals, the role of CNP is to allocate tasks and make the agents in the network share the necessary resources. While for the systems including competitive, so called self-interested agents, CNP works like a rule for a running race to make sure that every agent in the network competes with others for a prize under certain rational and suitable rule(s).

Figure 3-6 shows the differences between the two types of interactions. In the upper part of Figure 3-6, all *candidates* have private information and want to set up contracts with the *Human resource manager*, but usually only one or a few of them can succeed. The interaction between the *human resource manager* and each *candidate* is relatively independent. Thus the interactions in this part are the

self-interested interactions.

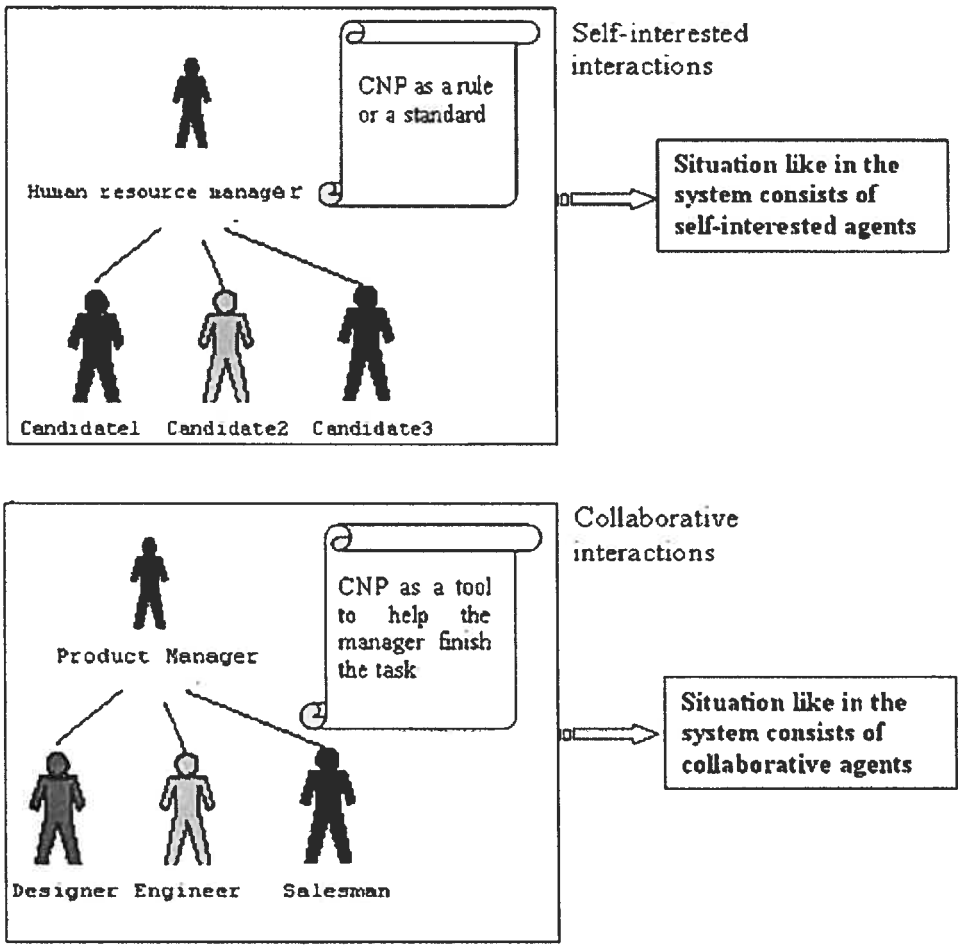


Figure 3-6 Examples of different type interactions

In the lower part, the *designer*, *engineer* and *salesman* need to work together with the *product manager* to achieve a common goal. There is no competition among the agents. Thus the interactions in this part are collaborative interactions.

The original CNP was designed to solve the problem of task assignment or resource allocation. For collaborative agents, an initiator can select a responder just based on behalf of its own and the current system condition(s), thus the contracting result is always optimized. For self-interested agents, however, they

are competitive, i.e. the responders have similar roles or functionalities. It cannot guarantee to be best off by accepting any individually rational contract that comes along. This means that the original Contract Net protocol is not exactly suitable for self-interested agents. Especially, when situations change, and an accepted contract may look less desirable in time, the initiator can cancel the contract even after both sides have agreed to a commitment. In FIPA specification, the situation of cancellation is handled as a Cancel exception. This is not the case for self-interested agents who expect some reward for their efforts. So simply cancelling the contract after the contractor has partially finished it without any payment is not acceptable.

With the development of E-marketing, systems involving self-interested agents are becoming more and more common. It is necessary to design different variations of the protocol so that each self-interested agent system does as its designer wants. Actually, some researchers have proposed some adaptations of original CNP to make sure this widely used protocol can work well in multi-agents systems involving self-interested agents. The detail of several extensions and adaptation of the original Contract Net will be described later.

3.4 Problems with FIPA CNP

Nothing is absolutely perfect, so is the Contract Net Protocol. It's been more than twenty years since Smith published his first version of Contract Net Protocol. Even after it became one of the standardized interaction protocols in FIPA, there are still lots of work to make it more practical and more efficient. This section will discuss some practical problems in applying Contract Net.

Compared with other interaction protocols, such as the Coloured Petri-Net Protocol [33], Contract Net is more flexible and closer to today's open-end marketplace concepts. Despite its high flexibility, the original Contract Net has its own problems:

- Less efficient
- Undefined trust issue
- Sub-optimal

1) Relatively lower efficiency

In general, since the initiator doesn't know which responder will answer its request, it broadcasts the request for proposal message to all members in the network and waits for all of them to answer back. For a large number of agents collaborating with CNP, it is often necessary for the initiator to wait for a long time to collect all proposals and decide the optimal contractors. The waiting time and other computing resources to handle such a large number of message transmissions for decision-making may become a bottle-neck. The slower the system is, the lower the overall system performance will be. On the other side, even if all of the responders commit to do the task, just one can get the contract. In this case, most of the system resource spending on the interaction has been squandered.

For the problem of message traffic, some researchers try to propose a mechanism that reduces the amounts of relevant information in agents communication [16], while some others try to find a suitable decision strategy in the negotiation issues, such as the one described by Karl Kurbel et al. [25]. In Kurbel's model, the negotiation strategy allows the initiator to pre-select some of the agents in the net to participate in the negotiation. The advantage of this model is that the initiator does not have to waste time on too much communications, but the pre-selection process limits the flexibility of the system.

To solve the problem of wasting the system resource, some researchers have proposed their solutions which will be discussed later. Actually, there are many other researches that focus on looking for the appreciate strategy

for improving the efficiency. They are not going to be described here.

2) Undefined trust issue

CNP relies on trust; the agents assume that all the others are believable and credible. Then, both sides of the interaction will assume that the other side will give out accurate information, thus there is no need of verifying. This is true for a closed traditional agent-based system, such as in the ***acquaintance network*** [6], trust is a natural characteristic and the agents should know more about the others' credit standing before any interaction. While for today's open systems like Internet, it's impossible to get all the information about the others.

In Internet based online shopping systems, trust has been considered as one of the critical issues [26]. For example, trust is almost the first important factor that will affect on whether the customer will purchase the merchandise from the online store. In the current widely used CNP, there is no constraint to the agents on their trust issue. This shortage will limit the use of CNP in practical e-market systems. As yet for an open-end system like the Internet, it's hard to solve this problem. Some of the MAS platforms like the JADE[5] provide certain solutions for the security problem, while they are still in the stage of trial. Currently, some online services authorize a third party to check the confidence of both sides of transactions. It seems to be able to relax the problem a little bit, but how to guarantee the reliability of the third party might become the next problem to be solved.

3) Sub-optimal problems

Sub-optimal problems in FIPA CNP, which is similar to the local optimum

[5] JADE provides JADE-S [27] as a plug-in for solving security problems.

problems in Artificial Intelligence, means that an initiator fails to set up a contract with the best available responder in the network.

Recall that FIPA CNP is a 1:N model, i.e., it aims at handling one initiator at a time. For this reason, once a responder receives a *CFP* message it will ignore all other CFP messages until it finishes a contract or receives a *reject-proposal* message. When more than one initiator exists in the agent system, this may cause sub-optimal problems.

Generally, the causes of sub-optimal problems can be:

A. delays of network communication,
B. the late appearances of some responders in the network,
C. and the race-condition caused by multiple initiators coexisting in the network.

For cause *A*, since the agents can appear anywhere and at any time in the network, there is a problem of balancing computational load throughout the network by distributing parallel tasks effectively to maximize the output. In a heterogeneous system which involves many interactions among agents, the initiator can hardly wait for answers from all the responders to make decisions, what it can do is to setup a deadline for collecting the information, then after the time out, the initiator has to make a decision of its offer. In CNP, the *deadline* needs to be set up at the point of collecting the proposals from the responder. It's like using the *Hill-Climbing* algorithm[6] toward finding a better solution than all the others at the moment, the final result can't guarantee to be best.

The following figure gives an example of what happens if messages are delayed by the network.

---

[6] *Hill-Climbing* is an algorithm for solving the search problem in AI [31].

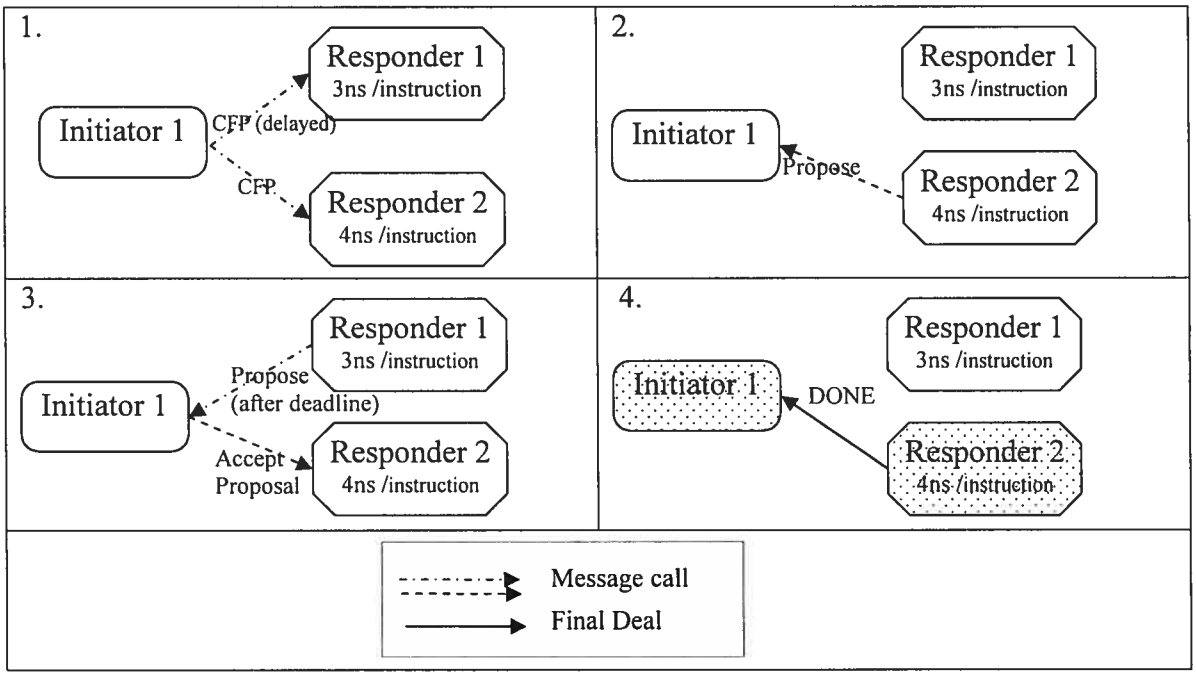


Figure 3-7 Example of sub-optimal situation

In Figure 3-7, the *CFP* message to **Responder1** is delayed by the network. **Initiator1** cannot make the decisions to maximize its own payoff without the information on all responders. Nevertheless, since the limitation of the deadline, when the deadline for collecting the proposals passes, **Initiator1** should make a decision immediately. It turns out that **Initiator1** is sending *accept_proposal* to **Responder2** when the *propose* message from **Responder1** arrives. The second proposal will be ignored, and the assigned contract for **Initiator1** is sub-optimal.

Cause *B* is in fact a variation of the cause A. Thus in the example above, the process of the interactions will be very similar if the **Responder1** comes into the network late. Then, no *CFP* messages may be sent to the late appearing responders.

Cause *C* has many variations. Since multi-initiator systems are very common in the real world, this issue will be discussed in the next section.

3.4.1 Sub-optimal Problems in multi-initiator systems

As mentioned above, CNP aims at handling one initiator at a time. If multiple initiators coexisting in a network, all responders only reply proposals to their first *CFP* messages received, and they need to commit themselves to do the jobs at the stage of sending proposals.

In the example used in this chapter, the responders are representing some exclusive network resources. That is to say, on one hand, each initiator wants to get deal with the BEST responder available in the network; on the other hand, once a responder has posted a propose to an initiator, it will not respond to any other initiator's request (CFP) until the contract is finished or receiving a *reject-proposal* message. In other words, whether or not an initiator can set up a contract with a desired responder will still depend on if its *CFP* is the first one received by the responder. This constraint generates a race-condition for the initiators in the network, and it leads to the sub-optimal problems in agent systems involving multiple initiators. In a large size multi-agent system, it's very common that multiple initiators and responders exist simultaneously, and the sub-optimal problem is going to be more serious.

Figure 3-8 and Figure 3-9 are two examples illustrating two possibilities of the resource allocation with race-condition problem. This time, each of the examples has two tasks and each task has different amount of work.

In Figure 3-8, **Responder1** receives **Initiator1**'s *CFP* messages, and **Responder2** receives **Initiator2**'s *CFP* message. Both of them ignore the other *CFP* message. Finally, **Responder1** sets up a contract with **Initiator1**, and **Responder2** sets up a contract with **Initiator2**. From **Initiator2's** point of view, the result is sub-optimal because it cannot set up a contract with **Responder1** which has the better bid.

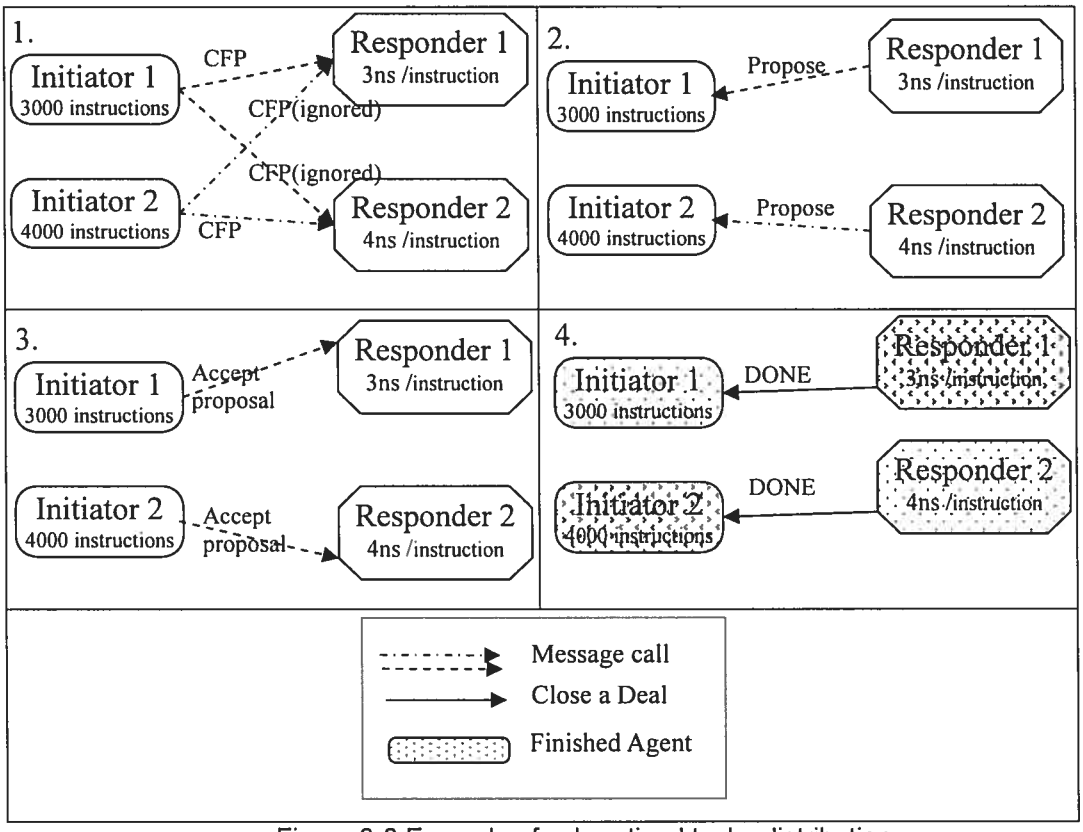


Figure 3-8 Example of sub-optimal tasks distribution

While one considers the four agents as a whole system, the result for the system is also sub-optimal. Simply calculating the contracting result for the system is like the following: assume there are *n* **Initiators** ($n \geq 1$), and *m* **Responders** ($m \geq 0$) in the system

$$T=\sum_{1}^{n} t(c(n)) \quad \text{if } n \leq m, \text{ , } c(n) \text{ denotes the cost for contract } n$$

$$T=\sum_{1}^{m} t(c(m)) \quad \text{if } n > m, \ c(m) \text{ denotes the cost for contract } m$$

Then in the situation above, $T_1=3000*3+4000*4=25000(ns)$, while if **Initiator1** assigns the contract to **Responder2** and **Initiator2** deals with **Responder1,** $T_2=3000*4+4000*3=24000(ns)$, $T_1 > T_2$.

In Figure 3-9, both **Responder1** and **Responder2** receive **Initiator1**'s *CFP* message, and ignore **Initiator2**'s *CFP* message. **Initiator2** has to wait until **Responder1** sets up a contract with **Initiator1** and finish the job or until **Responder2** receives *reject-proposal message,* then sends the *CFP* again. In the second round, most possibly, **Initiator2** can only interact with **Responder2**. This time, from **Initiator2's** point of view, the result is sub-optimal.

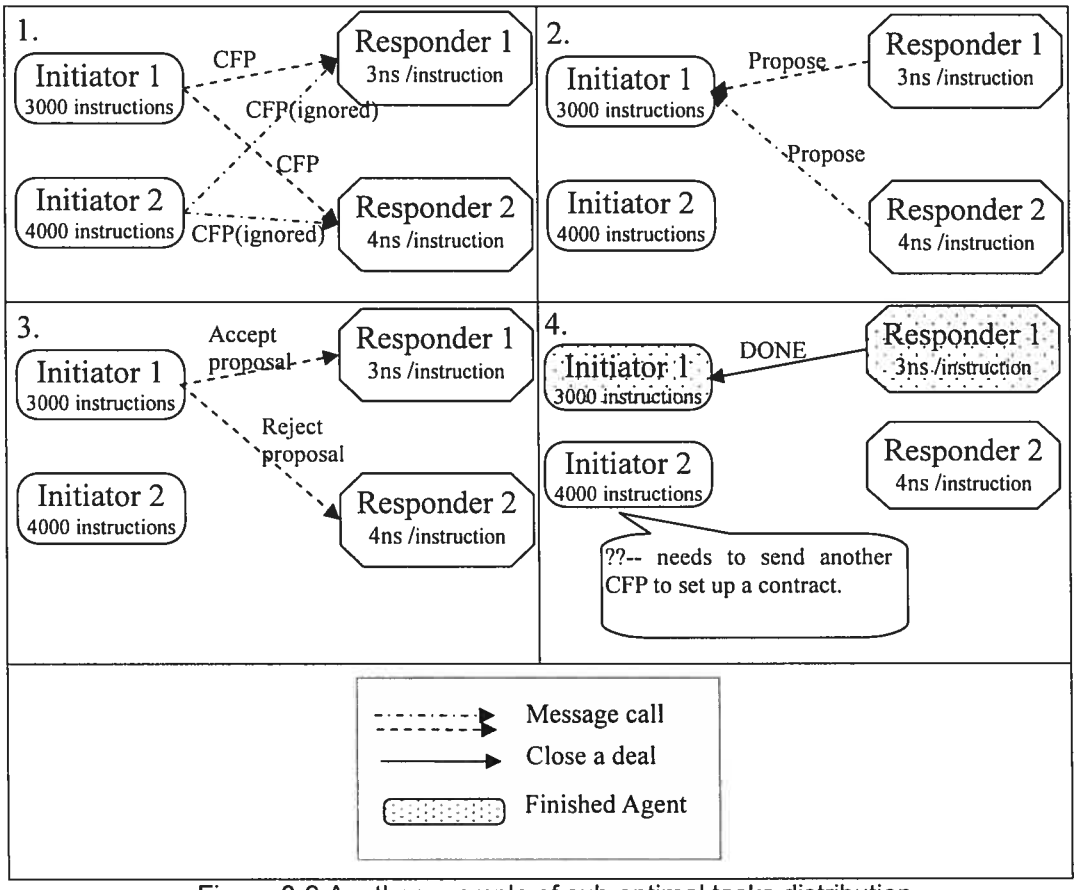


Figure 3-9 Another example of sub-optimal tasks distribution

One should understand that both the optimal and sub-optimal concepts are relative to a specific point of view. When applying the FIPA contract net protocol in agent system involving self-interested agents, the profit of individual agent is considered to be the most important. In this chapter, the sub-optimal problem is mainly focused on individual agents. The situations are also suitable for the collaborative

agents which just care about their own benefits. From the point of view of the whole system, the meaning of sub-optimal is very different. This topic will be discussed in chapter 4.

3.4.2 Solutions

As one can see, in spite of its advantages, FIPA CNP has the problems mentioned above that limit its application in MAS. For the problems caused by network delays, one may try to enhance the network transmission speed. For other causes, especially the race-condition caused by multi initiator coexisting at the same time, revising the protocol is necessary.

3.5 Other attempts to improve CNP

Over the time, some researchers have proposed a few extended versions of CNP or adaptations. Among them, Tore Knabe et al, proposed a revised version of FIPA CNP [28], and named Contract Net with Confirmation Protocol (CNCP).

In addition to CNCP, in recent, two Chinese researchers released a report [29] that presented a further extension of FIPA CNP, which is called CNP with threshold plus DoA[7]. They use a threshold for *responder* to limit the quantity of initiator with which communicates. At the same time, the initiator will evaluate the availability of responders. Since their model mainly aims at the problem of high unnecessary computational cost in CNP for a relatively large size agent system, it will not be discussed in detail here.

Generally, the extensions and adaptations can be divided into two types: one is to change the framework of original one, such as CNCP which adds a confirmation stage to postpone the commitment time; the other is to add a bidding strategy, such as the Levelled-commitment [14] which applies a bidding strategy to find a better

[7] DoA represents degree of availability [14].

offer.

The next chapter will discuss the detail of CNCP. In chapter five, a new combinatorial extension based on both concept of CNCP and Levelled-commitment CNP will be presented.

Chapter 4 Contract Net with Confirmation Protocol

The previous chapter has discussed the use of CNP in MAS in detail. This chapter will give a close look at one of its extensions, the Contract Net with Confirmation Protocol (CNCP), and analyze its advantages and disadvantages.

As mentioned in Chapter 3, despite some great advantages, FIPA CNP has its own drawbacks. Especially if an agent system contains multiple initiators, FIPA CNP will cause sub-optimal problems. To avoid these drawbacks, Tore Knabe et al, proposed a revised protocol [28], named Contract Net with Confirmation (CNCP).

4.1 The CNCP concept

Knabe et al. redesigned the original protocol and tried to postpone the time of commitment. In CNCP, a responder only commits to do the job after receiving a confirmation message from the initiator which owns the job saying that the task will really be assigned to it. Thus, even if the responder has sent out a bid to an initiator, before obtaining the confirmation message, it's still free to bid on the other job. As a result, other managers can get more proposals to compare and select.

Knabe's CNCP model adds the following characteristics to the FIPA CNP:

1) It is an $N{:}M$ ($M,N \geqslant 1$) model, i.e. it allows multiple initiators in the network working simultaneously.
2) Responders reply to all *CFP* messages so that each initiator can have a chance to collect the proposals from all responders.
3) The initiators now have two receiving deadlines.
4) After collecting all *propose* messages (or after the deadline for receiving *propose* has passed), each initiator will send a *request* message to the responder who offers the best *proposal*.
5) When a responder gets the first *request* message, it will reply with an

agree message to the corresponding initiator, and then send *refuse* messages to any other initiator which also sends it *request* message.

6) After sending a *request* message to the responder with the best proposal, if an initiator gets a *refuse* reply, or the deadline for receiving agree passes, it will send a *request* message to the responder that offers it secondary best proposal, and so on until it gets an *agree* message or no more proposal to be reply.
7) *reject_proposal* messages are sent out only after an initiator gets an *agree* message.

Figure 4-1 bellow is the UML diagram for Knabe's CNCP model. It can be seen from the diagram that CNCP is very similar to CNP. The part inside the circle shows the added confirmation stage.

Here, after collecting responses from all **Responder**s or after **deadlineA** having passed, the **Initiator** will choose the best proposal from a sorted proposal list, and send its *request* to the corresponding responder. The way of sorting the proposals depends on how the **Initiator** evaluates them. The **Initiator** will use its own strategy to arrange the proposals' order. On the other side, when a **Responder** receives the first *request* message, it will reply with *agree* message to bind to a commitment. The **Responder** then sends the *refuse* message to other **Initiator(s)** who provide(s) their offer later.

If a **Responder** refuses the offer or **deadlineB** has passed, the **Initiator** will choose the next one in the sorted list to send the request again. This process will be done repetitively, until one **Responder** gives a binding commitment to do the job or there is no more **Responder** available in the sequential list.

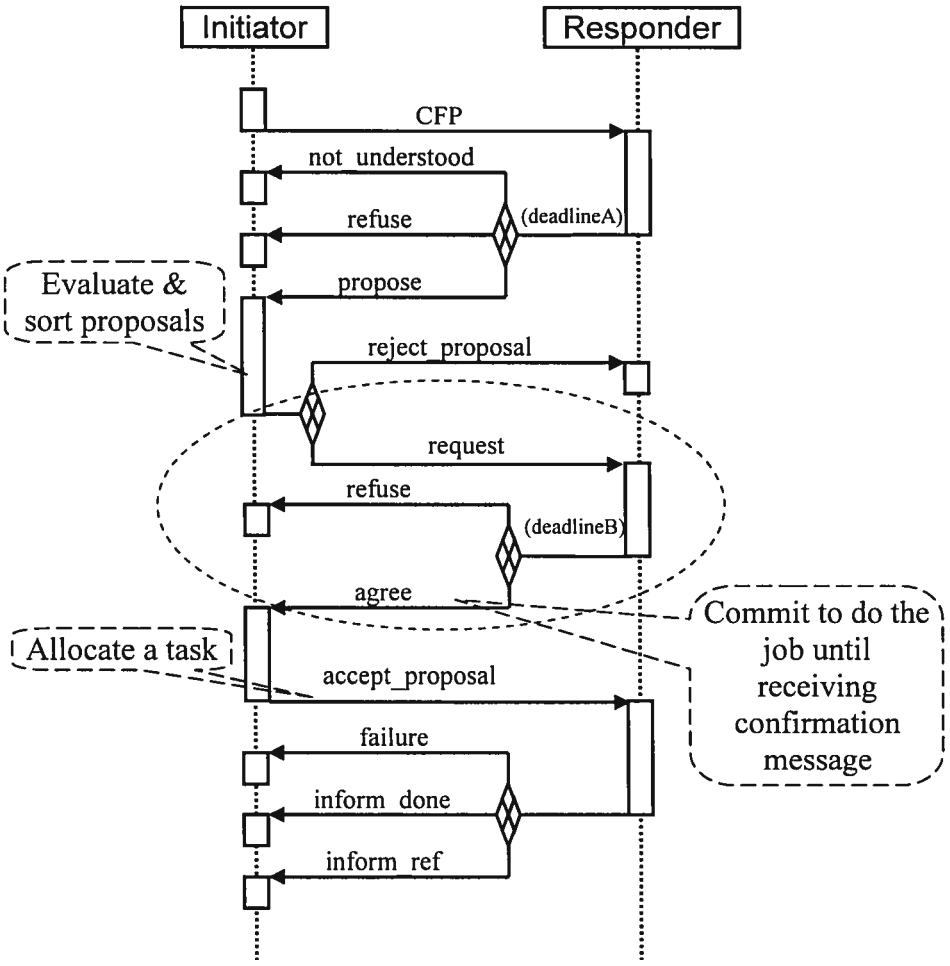


Figure 4-1 UML Diagram of CNCP

Figure 4-2 bellow shows the process of handling the sorted list of each initiator. One can see that when the responder refuses the initiator's offer or **deadlineB** passes, the initiator just chooses another proposal to send request without restart another interaction.

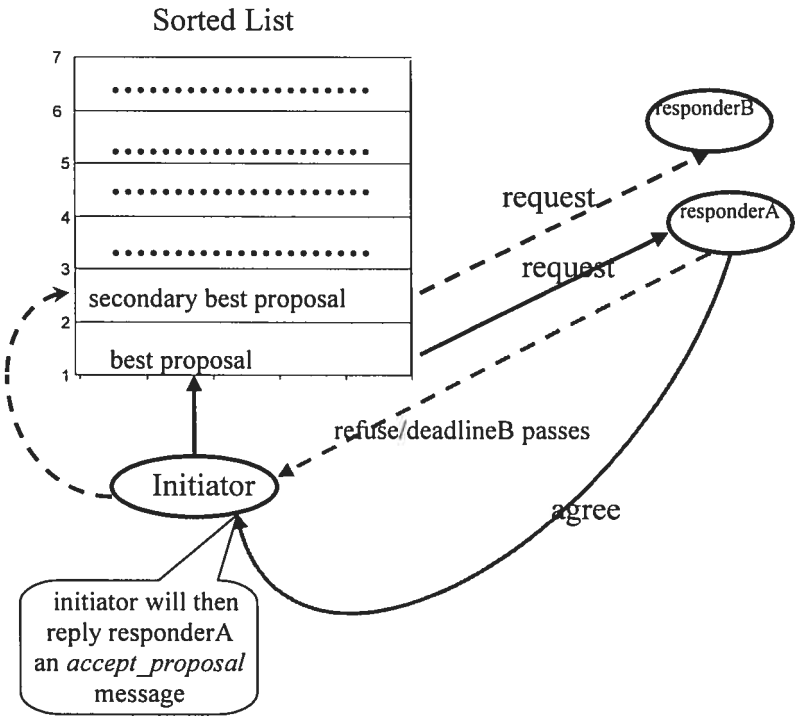


Figure 4-2 Handling the sorted proposal list in CNCP

Figure 4-3 bellow shows how the Knabe's CNCP handles the situation presented in figure 3-8. In step 2 of figure 4-3, each responder replies proposals to all initiators that send them *CFP* messages. In step 3, each initiator sends a *request* message to the desired responder (**Responder1**) that provides the best proposal. In step 4, **Responder1** sends an *agree* message to the owner (**Initiator2**) of its first received *request* message, and sends *refuse*s to the other initiator. In step 5, as soon as **Initiator1** gets the *refuse* message, it sends out another *request* message corresponding to **Responder2**'s proposal, and then continues the interaction until the task has been assigned to **Responder2**. Both of the initiators get their jobs done through one circle of interaction.

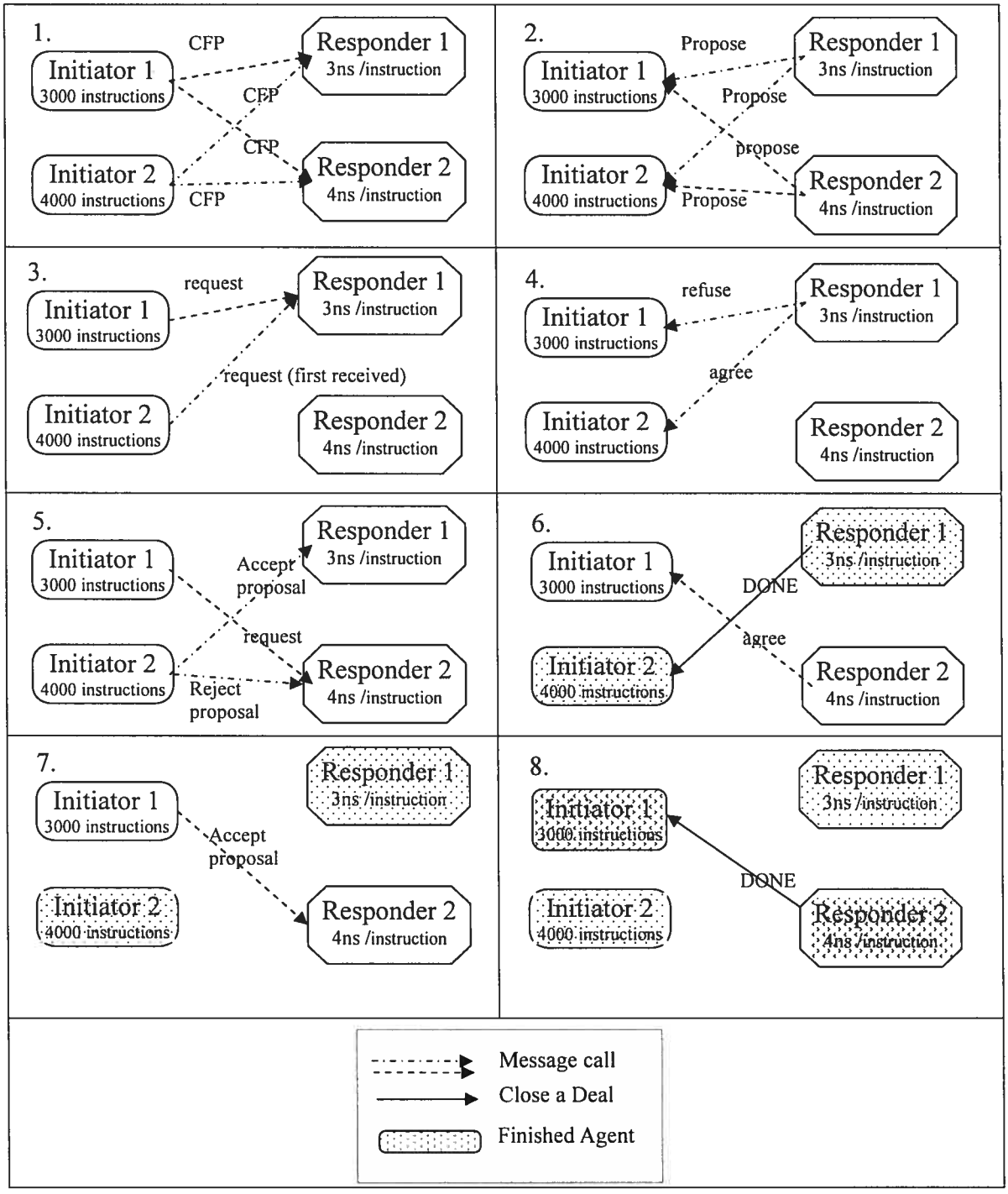


Figure 4-3 Multi Initiator work with CNCP

4.2 More on message types and deadlines

To understand better about how CNCP works, and make the further discussion about CNCP and its extension protocol (Levelled CNCP in chapter 5) easier, it will be helpful to give some more explanation about the message types and the deadlines.

4.2.1 Classification of message types

By reviewing the UML diagrams of FIPA CNP and Knabe's CNCP, one can see that the message types used in these protocols can be classified as "constructive" types and "destructive" types. Constructive types can help setting up a contract. These types in CNCP include *CFP*, *propose*, *request*, *agree* and *accept_proposal*. Destructive types are used for avoiding or terminating a contract. These types in CNCP include *refuse*, *reject_proposal*, *inform_done*, and *inform_ref*. Note that *not_understood* and *failure* can be used either for keeping a current interaction of a contract or for abandoning the current interaction. Therefore they can be either a constructive type or a destructive type, depending on the actual implementation.

After an agent sends out a constructive type messages, it needs the receiver(s) to send back responses so that it can decide what to do next. In other words, if any agent receives a constructive type message, it should reply, no matter the response is constructive or destructive. This is why in step 5 of figure 4-3, **Initiator2** sends a *reject_proposal* to **Responder2** as soon as it sends an *accept_proposal* to **Responder1**, so that **Responder2** can continue to make a decision for the next step (**Responder2** sends a *propose* to **Initiator2** in step 2).

Contrarily, if an agent receives a destructive message, it doesn't have to reply, because the sender doesn't expect its response for making further decisions. That is why after step 5, no more messages are sent to **Initiator2** from **Responder2**.

4.2.2 Understanding more about the deadlines

Whenever an agent is expecting some incoming messages to decide the next action, it needs to set up a deadline for the current receiving stage. For example in CNCP, at the beginning of an interaction, responders need to collect *CFP* messages from all initiators, but *CFP* messages won't arrive to the responders at exactly the same time. Thus each responder needs to set up a deadline and to keep receiving for a certain period so that it knows who send it *CFP*s and what proposals it should reply back. If a *CFP* arrives to a responder later than its deadline, the message will be ignored unless the responder is ready to receive more *CFP*s (in the next interaction cycle). Contrary in CNP, responders expect to receive only one *CFP*, so they don't need to set up a deadline for that.

The UML diagram of figure 4-3 and the related explanations are respecting to Knabe's original diagram which shows only two deadlines. In fact in actual implementations, all of the stages of receiving *CFP*, *request* and *accept_proposal* should also have deadline. The UML diagrams of the revised CNCP (figure 4-8) and the Levelled CNCP (figure 5-2) will show those deadlines to make the descriptions clearer.

4.3 Comparison between CNP and CNCP

As mentioned above, CNCP is an extended version of CNP, and it is very similar to CNP. In an agent system involving just one initiator at one time, CNCP works the same way as CNP except that it adds a *request* message and an *agree* message to the interaction framework. Since most open systems in the real world are MASs, the real objective of CNCP is aiming at solving the sub-optimal problem of applying CNP in those systems which can have a large number of initiators and responders working simultaneously.

Unlike CNP in which responders only pick up the first coming *CFP* and have no

way to interact with other initiators, by adding a confirmation stage to postpone the time of commitment and allow responders to send responses to multiple *CFP* messages, Knabe's CNCP gives every initiator a chance to collect the information of all responders and choose the "best" responder to contact with. However, responders still have no control to the allocation processes in this protocol. The description later in 4.3.2.2 will give more explanation on this issue.

Knabe's paper [28] compared the differences between CNCP and CNP by their allocation policy and the result of allocation. Unfortunately, comparing allocation policy doesn't properly distinguish the performance differences between the protocols; while comparing the result of allocation, Knabe's paper didn't compare the results from the whole system point of view, and again, cannot clearly tell distinguish the performance differences.

In fact, to compare CNCP with CNP, one should look at the following two aspects: the system message traffic and the system resource utilization efficiency. It's obvious that improving these two issues may help the overall system performance. Since CNCP and the Levelled CNCP (mentioned in chapter 5) do not concern about the enhancement of security, the trust issue will not be discussed in the rest of the thesis.

4.3.1 Comparing the system message traffic

Let's assume that in a multi-agent system, there are n initiators and m responders, and $n \leqslant m$.

1) Messages needed in the interactions using CNP

If all initiators can get at least one proposal back after they send *CFP* messages, then all initiators will finish setting up their contracts at the same time. In such case, the total message traffic in the system is **minimum**. Figure 4-4 illustrates the

messages traffic in CNP.

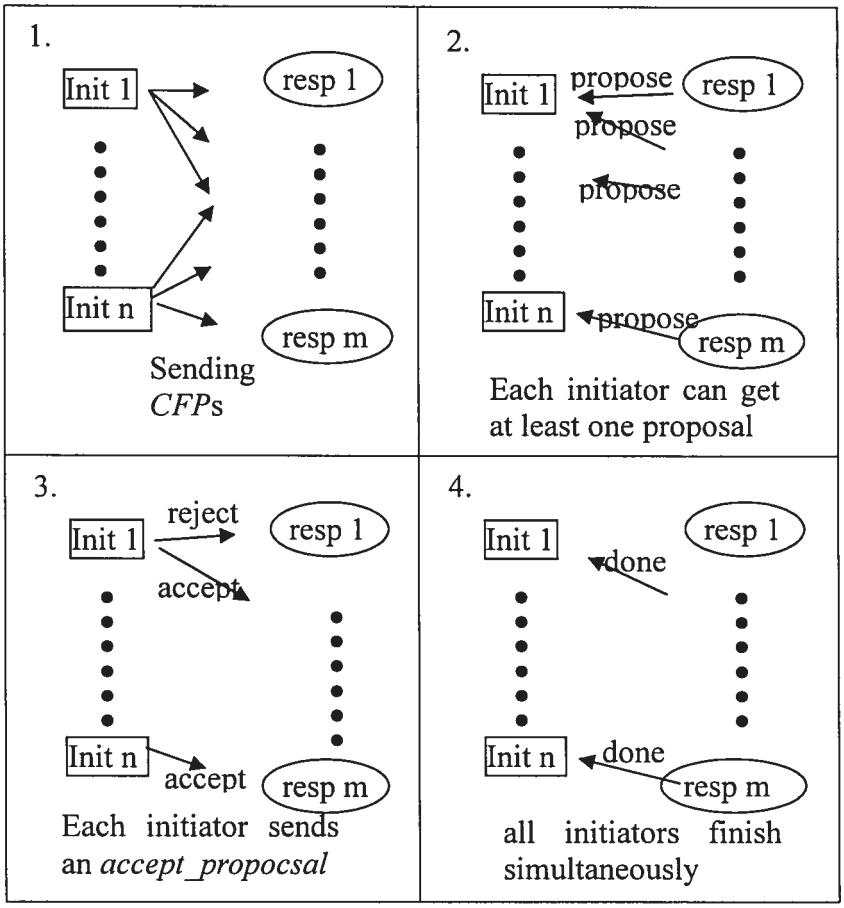


Figure 4-4 Message traffic in CNP (best case)

In this best case, the total messages

$$M_{min} = (nm)_{\underline{cfps}} + m_{\underline{proposals}} + n_{\underline{accepts\ \&\ rejects}} + n_{\underline{inform\ done}}$$
$$= nm + 2n + m \qquad (4.1)$$

If in each turn there is only one initiator getting proposals from all available responders, then ***n*** initiators need ***n*** runs to setting up all the contracts, and in each run all remaining initiator(s) need to send new *CFP* messages to all remaining responders. In such case, the total message traffic in the system is **maximum**, and figure 4-5 shows the system message traffic in such situation.

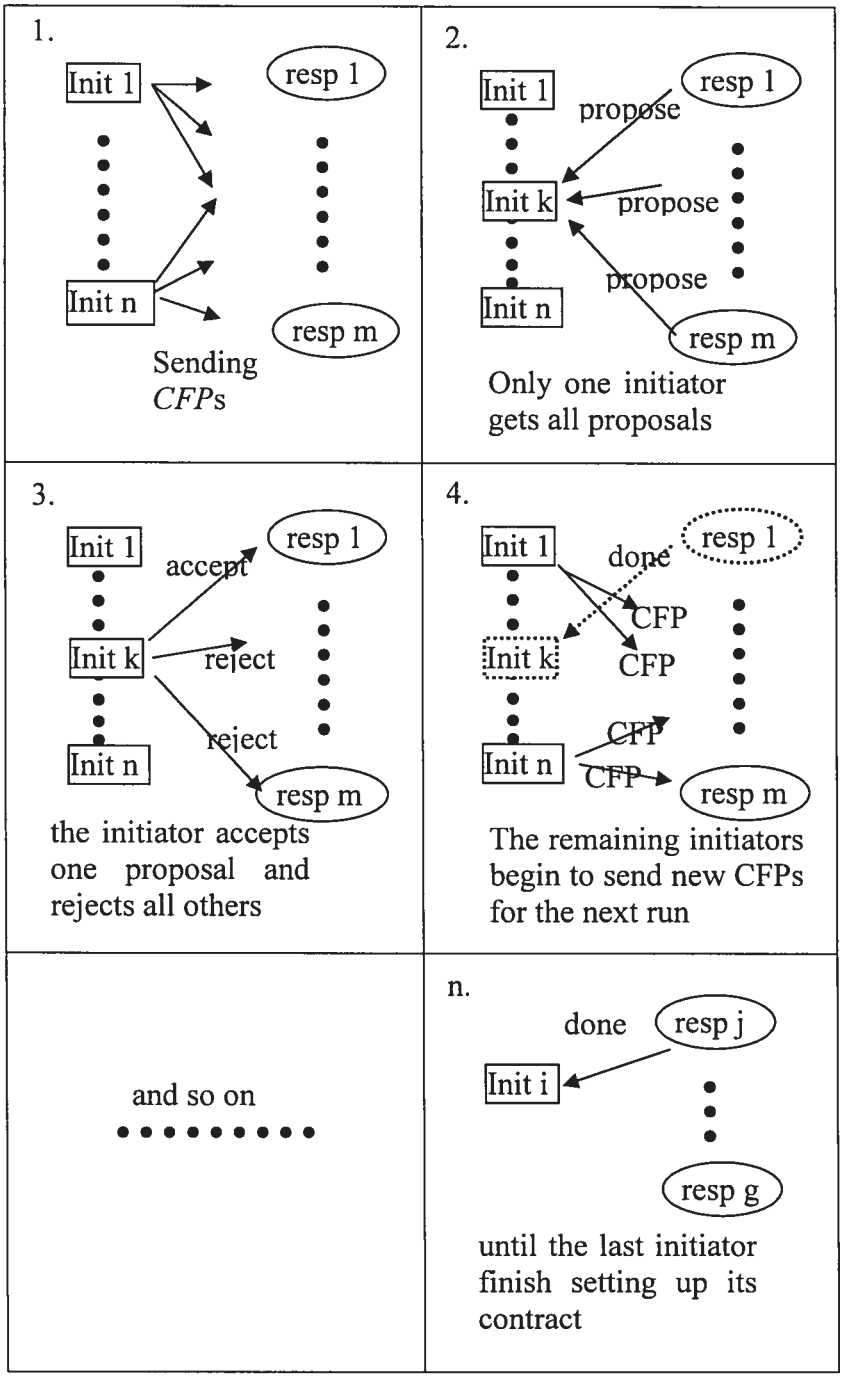


Figure 4-5 Message traffic in CNP (worst case)

And then, the maximum total of messages

$$M_{max} \cong (mn)_{\underline{cfps}} + n_{\underline{proposals}} + n_{\underline{accept\ \&\ rejects}} + 1_{\underline{inform\ done}}$$

$$+(m-1)(n-1)_{\underline{(new)\ cfps}} + n-1_{\underline{proposals}} + n-1_{\underline{accept\ \&\ rejects}} + 1_{\underline{inform\ done}}$$

$$+ \ldots\ldots$$

$$+ (n-m+1)_{\underline{(new)\ cfps}} + n-m+1_{\underline{proposals}} + n-m+1_{\underline{accept\ \&\ rejects}} + 1_{\underline{inform\ done}}$$

$$\cong n^2m + 2mn + 3n - 2n^2 + [1^2+2^2+\ldots+(n-1)^2]$$

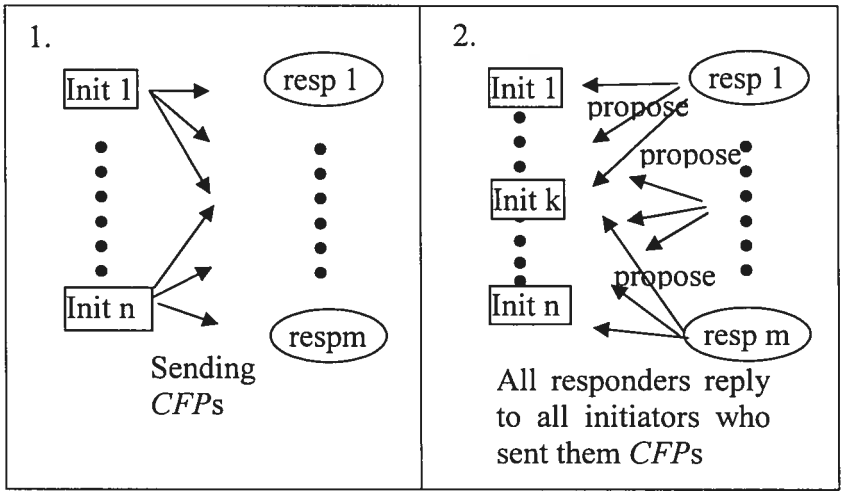
$$= n^2m + 2mn + 3n - 2n^2 + n(n+1)(2n+1)/6$$

$$= n^2m + 2mn + n^3/3 - 3n^2/2 + 19n/6 \qquad (4.2)$$

Equation 4.1 and 4.2 show that when the number of initiators (m) is not greatly larger than the number of responders (n) – which is often the case, the total number of messages of in a CNP system can vary significantly from order of nm to order of n^2m+n^3, i.e. approximately from $O(n^2)$ to $O(n^3)$ when $m \cong n$.

2) Messages needed in CNCP interaction

For CNCP, if a MAS has no new coming agent(s) after it starts, initiators only need to send *CFP*s to all responders one time; and responders only need to reply to all of the *CFP*s one time too! Now if all initiators can get the contracts assigned after they send *request* messages, then all initiators will set up their contracts at the same time. In this best case, the total message traffic in the system is **minimum**, and the system working process is illustrated in figure 4-6.



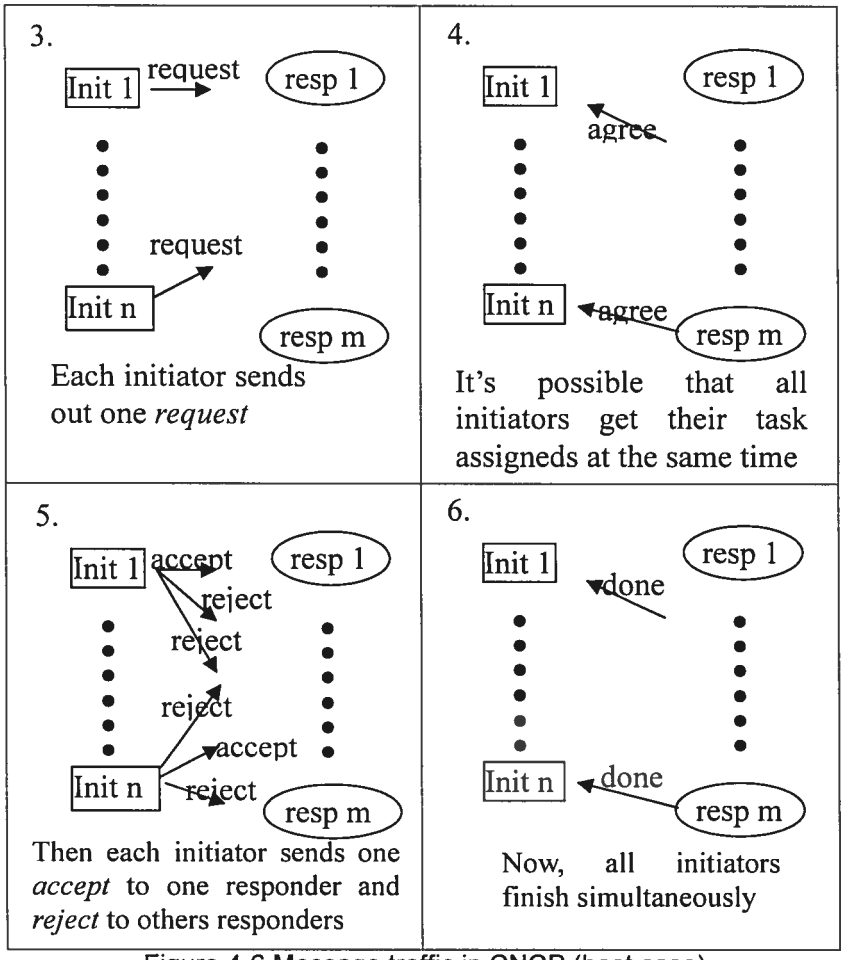


Figure 4-6 Message traffic in CNCP (best case)

Here, the communication requires messages T_{min},

$$T_{min} = nm_{\underline{cfps}} + nm_{\underline{proposals}} + n_{\underline{request}} + n_{\underline{agree}} + nm_{\underline{accepts\ \&\ rejects}} + n_{\underline{inform\ done}}$$
$$= 3nm + 3n \qquad (4.3)$$

Note that in step 5 of figure 4-6, since each initiator who sends out an *accept_proposal* message to a responder also needs to send out *reject_proposal* messages to other **m-1** responders, as stated in section 4.2.1, the total messages sent in this step is **nm**.

In fact, the best case rarely happens. It requires that the initiators send *request* messages at almost the same time, and moreover, any responder must not receive

more than one *request* message. The more possible situation is that some responders receive multiple *request* messages and some have no *request* message received at all. In the worst case, which will generate the maximum number of message, (when the network transmission delay can be ignored) all initiators tend to send *request* messages to only one responder. In such case, there is only one "lucky" initiator getting an *agree* message back and others will get a *refuse* message. Then the initiators gotten the *refuse* message will send *request* messages to the second best responder, and so on. The most "unlucky" initiator might need to contact all responders to find out a contractor. Figure 4-7 (starts from step 3 of figure 4-6) shows this situation.

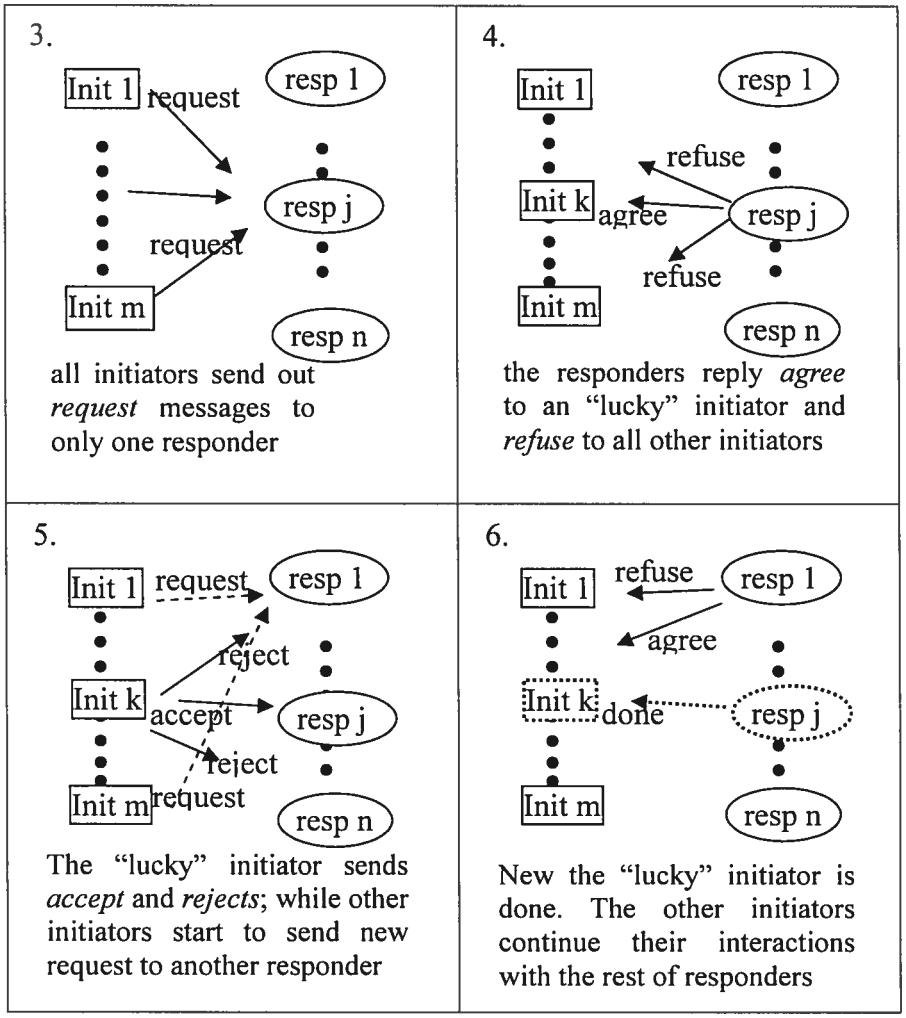


Figure 4-7 Message traffic in CNCP (worst case)

Here, in general, the maximum messages needed is T_{max},

$$T_{max} \cong nm_{\underline{cfps}} + mn_{\underline{proposals}} + n_{\underline{request}} + m_{\underline{agree\ \&\ refuse}} + n_{\underline{accept\ \&\ rejects}} + 1_{\underline{inform\_done}}$$
$$+(n-1)_{\underline{(new)\ request}} + (n-1)_{\underline{(new)\ agree\ \&\ refuse}} + m\text{-}1_{\underline{accept\ and\ rejects}} + 1_{\underline{inform\ done}}$$
$$+ \ldots\ldots$$
$$+ 1_{\underline{(new)\ request}} + 1_{\underline{(new)\ agree}} + m\text{-}n+1_{\underline{accept}} + 1_{\underline{inform\ done}}$$
$$= 2m*n + 2n + (n^2 + 5n)/2 \qquad (4.4)$$

Comparing (4.1) with (4.3) and (4.2) with (4.4), we can see that

- The minimum message requirement (best cases) of CNCP is about three times as many that of CNP;
- The maximum message requirement (worst case) of CNP is much greater than that of CNCP, especially when the number of agents is large. This is because in CNCP, *CFP* messages and *propose* messages are sent once only; while in CNP, these messages need to be sent repeatedly.

Thus for a large agent system involving multiple initiators, CNCP often has better message traffic efficiency in the average case.

4.3.2 Comparing the system resource utilization

The concept of system resource utilization efficiency directly indicates how well the system resources are allocated to the requesting tasks. Increasing the system resource utilization efficiency is the main objective of interaction protocols. Before comparing CNP with CNCP on this issue, one should understand how to evaluate the system resource utilization efficiency.

4.3.2.1 Sub-optimal problems regarding to the system level

As described so far, one should understand that in a multi-initiator system, since

the resources which we concern about are exclusive, it is impossible that two or more tasks can share one resource simultaneously, i.e., only one initiator can contract with a responder; if other initiator want to contract with this responder, they have to wait until the current contract is finished. The concept of "sub-optimal" in the previous chapter only applies to individual agent. From the individual agent's point of view, at any time, if one initiator "occupies" the "best" responder, then all other initiators will get "sub-optimal" result. While from the whole system's point of view, one or more agents getting sub-optimal results can't represent the resource allocation status of the system. To evaluate the performance of a protocol, one should consider the whole system, instead of individual agent. Thus, in the rest of the thesis, if not indicated specially, the term "sub-optimal" will always apply to the whole system.

To understand this idea, let's review the analysis of Figure 3-8 and Figure 3-9. We can summarize the system time consumptions calculated in section 3.4.1 into the following two tables so that we can easily compare.

Final Deals	Analytical result	Total time (ns)
Contract a	Initiator1 ↔ Reponder1	3000* 3 = 9000
Contract b	Initiator2 ↔ Reponder2	4000* 4 = 16000
Total		25000

Table 4-1 System time consumption for case of Figure 3-8

Final Deals	Analytical result	Total time (ns)
Contract a	Initiator1 ↔ Reponder2	3000* 4 = 12000
Contract b	Initiator2 ↔ Reponder1	4000* 3 = 12000
Total		24000

Table 4-2 System time consumption for another possible result of Figure 3-8

It's clear from the tables that the system resource utilization efficiency in Figure

3-8 is higher than that in Figure 3-9, i.e., the system can consume less time to handle the same tasks.

From now on, for the whole system, it's preferred to say: for a certain way of system resource allocation, if there is no other way that gives out better resource utilization efficiency, then this way of task and resource allocation is **optimal**; otherwise, it is **sub-optimal**.

4.3.2.2 The sub-optimal problem of Knabe's CNCP model

A difficulty with CNP is that the agents may not always disclose truthful information, which is necessary for optimal global task allocation. To make decisions by accurately compute the value of an accepting task, a potential contractor (responder) needs a view of all possible tasks – those available at the moment and those that might arrive in the near future. The problem is that the relationship between an initiator and a responder is not symmetrical. The initiator knows all of the bids proposed to it, but a responder doesn't. Thus a responder cannot predict the outcome of its bidding. For the whole system including all initiators and all responders, the outcome also becomes unpredictable. This is the uncertainty problem of CNP.

As mentioned so far, in Knabe's CNCP model, the mechanism is like this: when a responder receives the first *request* message, it replies an *agree* message; and after that, all *request* messages are replied with *refuse* messages. In other words, in the decision making stage, the responder just follows the first come first serve rule, no matter how good the later offers are. In such case, the responder still has the problem of knowing the information of all initiators.

In fact, although Knabe's CNCP mechanism gives responders a chance to interact with more than one initiator at the same time, responders still have no chance on choosing initiators, and thus the uncertainty problem still remains. For example, in step 4 of figure 4-7, **responderJ** sends an *agree* to **initiatorK** only because it

receives **initiatorK**'s *request* first. It doesn't mean that **initiatorK** is the best initiator to contract with.

4.4 A revision to Knabe's CNCP model for asynchronous MAS

As mentioned in section 4.3, Knabe's CNCP doesn't allow responders to consider more than one *request*. To solve this problem, this thesis proposes a revision to this protocol:

- Same as the initiator(s) receiving *propose* messages, each responder will try to collect all *request* messages up to the receiving deadline (**deadline2** in figure 4-8) passes, and sort them up according to some rules regarding to the content of the messages. The "best" *request* (at the top of the sorted list) will be replied with *agree* message.
- After an *agree* message is sent, if the responder cannot receive the expected *accept_proposal* message before its receiving deadline, the secondary "best" *request* will be replied with *agree* message, and so on, until an *accept_proposal* message is received or the sorted list is empty.

The revised CNCP UML diagram is shown in figure 4-8. As stated in section 4.2.2, figure 4-8 shows the other deadlines that should be used in the implementations. To distinguish them clearly from the ones originally in figure 4-1, they are named a little bit differently.

To see more clearer what kind of MAS that CNCP is suitable to, here I separate MASs into two types of systems: **asynchronous** systems and **synchronous** systems. If all agents are existing since a system starts, and all initiators sending *CFP* messages at the same time[8], i.e. there will be no agent becoming "available" (appearing or being activated) later than other agents, the system is **synchronous**; otherwise, the system is **asynchronous**. For example, the elevators in a building

[8] Here, "all initiators send *CFP*s at the same time" means that all *CFP*s can reach the responders before their receiving deadlines for *CFP* (deadline1) have passed.

can be treated as a synchronous agent system because all elevators are available to work since the beginning; an E-commerce system can be treated as an asynchronous agent system because customers generally won't appear at the same time.

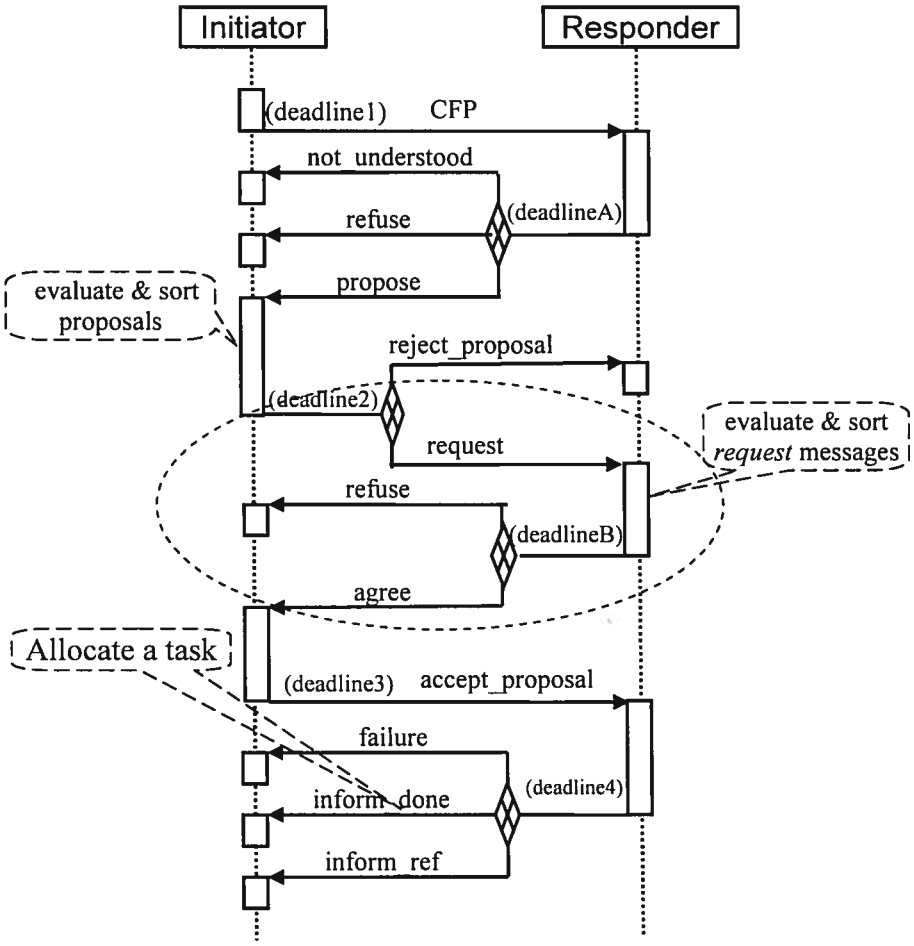


Figure 4-8 Revised CNCP UML diagram

The implementation of this revised CNCP shows that for synchronous systems in which all initiators exist simultaneously, the revised CNCP model can almost completely solve the sub-optimal problem of Knabe's model. This implementation result will be discussed in more details in chapter 7.

4.5 The sub-optimal problems in asynchronous MAS with CNCP

We know that in the real world, most agent systems are asynchronous. In asynchronous systems, agents can be available at any time.

In a multi-initiator agent system applying the revised CNCP, for example, if a better initiator becomes available after a responder sends out *propose* messages, the responder has no way to receive the *CFP* message from the new initiator. Then the result of the system may become sub-optimal.

In the next chapter, a new protocol that combines the revised CNCP with a levelled-commitment concept will be proposed to allow a better solution to the sub-optimal problem in asynchronous systems.

Chapter 5 CNCP with Levelled Commitment

CNCP with Levelled Commitment, namely Levelled CNCP, is built on top of the concept of CNP of Levelled Commitment Protocol and CNCP. The CNCP concept was already introduced in the last chapter. Before describing the details about CNCP with Levelled Commitment, it is necessary to look at the CNP of Levelled Commitment Protocol first.

5.1 The concept of CNP with Levelled Commitment

For the uncertainty problem of the CNP mentioned in 4.3.2.2, Sandholm et al proposed another solution [14, 15]. They suggested a number of extensions to the contract net protocol including more fluid interactions to allow the agents to adapt to various levels of commitment. This adaptation allows a more complete searching because the agents are free to make contingent commitments [15], and may get back if a better offer arrives. In other words, any commitment is temporary and can be replaced by a "better" one until a final decision is made later.

In this protocol, an agent may try an offer with many agents, allocate risk according to its own risk tolerance, and enter contingent contracts [15]. By permitting both sides of the interaction to decommit, i.e. to cancel the existing commitment(s), from the awarded contract according to new information, this model allows agents to profit from the future event such as new tasks arriving or new source becoming available. If the new events make the old contracts unbeneficial or infeasible, one can decommit from the old contract.

The CNP with Levelled Commitment Protocol doesn't change the structure of the original protocol too much. Instead, it adds strategy when bidding for a contract: even after a commitment is set up, an agent can still have the opportunity to find a better partner. Figure 5-1 shows the UML diagram of the protocol.

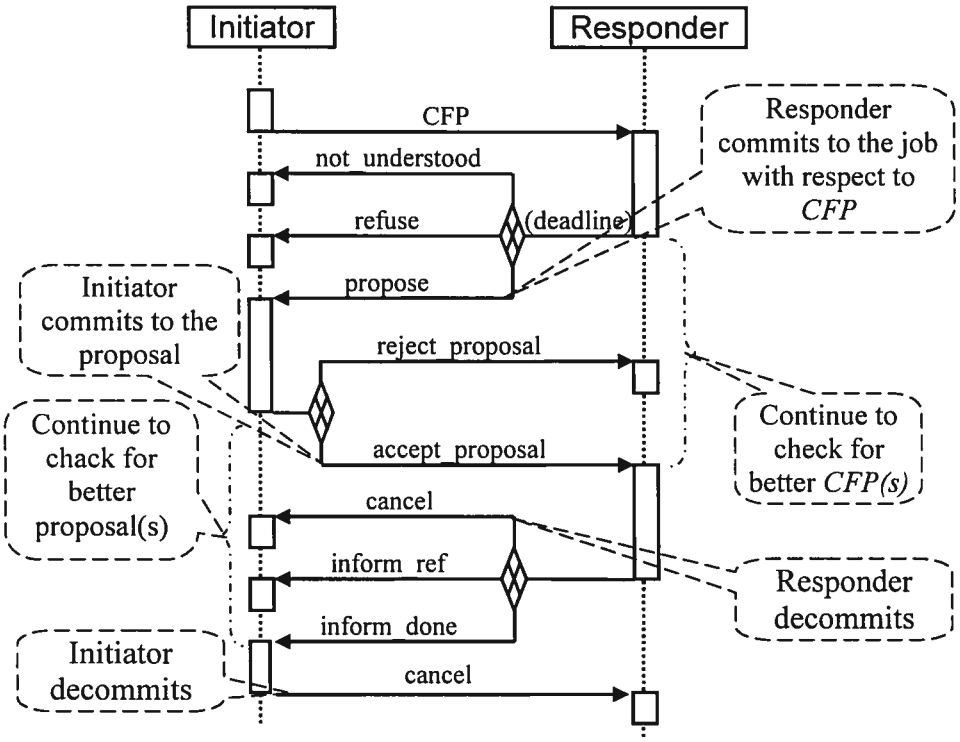


Figure 5-1 Diagram of CNP with leveled commitment

In figure 5-1, the structure of the protocol is basically the same as that of FIPA CNP, except that at the end of the diagram, there's a *cancel* message for **Initiator** to decommit the **Responder** in case that the former has a better alternative to commit. A responder commits to an initiator when it sends a proposal to the latter. In turn, an initiator commits to a responder when it sends out *accept_proposal*. A responder can decommit the task after it receives an *accept_proposal* message.

Notice that there's no need for a responder to send *failure* message after it receives an acceptance. It can just send *cancel* message to decommit the task [15].

An issue that must be noticed here is the compensation rate for decommitment. When talking about the problems of CNP, we have to notice the difference between the collaborative agents and self-interested agents. The loss of an awarded task by cancelling the contract will bring the corresponding compensation burden

for self-interested agents. If the compensation rate is too high to achieve a better outcome, this protocol will become meaningless.

5.2 The advantages of CNP with Levelled Commitment

The first advantage of CNP with Levelled Commitment protocol is that it allows all participants (initiators and responders) to give considerations and reactions to the late coming information of the system. Secondly, this protocol can increase the welfare for both parties of the contract by reallocating tasks.

This protocol can be applied in agent systems involving either self-interested agents or collaborative agents. For the self-interested agents, the agents will decommit from the previous contract only when the new events can increase their own immediate or expected payoff. The cooperative agents will consider the summed payoff of all the agents in the system when deciding whether to decommit from an awarded contract.

In addition, the agents with look-ahead [15], i.e. estimating the payoff if a decommitment happens, will need much more computation cost than that of without look-ahead. If the agents perform full look-ahead, they compute the payoff of all possible future events and decommit from the previous contract only if the expected payoff is better than the former contract. This is similar with Game theory [30]. The agents without look-ahead, namely the myopic agents [15], only consider the immediate payoff under the communication. Myopic agents decide to decommit when a new event has happened, and the individual agent can get more expectable payoff from the new contract. The decision making myopic agents seems simpler than that of the agents with look-ahead, but in complex MAS, with the increasing of new events, the computational cost will be still be enormous.

Although the CNP with Levelled Commitment Protocol has the above benefits, it is not obviously superior to the original Contract Net Protocol. The reasons are:

- The penalty sometimes means that the decommitting may decrease the overall profit from the contract;
- Even for an experienced human being, how to make decision for future event is very difficult, a software agent has to face more uncertain situations.
- To get maximum profit, the self-interested agent might decommit strategically and thus require large amount of computation.

5.3 Combining CNCP together with Levelled Commitment CNP

Both the CNCP and CNP with Levelled Commitment have their own advantages as well as drawbacks. Comparing with CNCP, CNP with Levelled Commitment has more flexibility with considering the effect of future events; but for a complicated system, it is not so easy to predict future events. On the other hand, CNCP, even for the revised CNCP, may lead to sub-optimal results in asynchronous agent systems, but the computation is much simpler than that of CNP with Levelled Commitment, and the time efficiency of setting up contracts may also be higher. To get a balance between flexibility and time efficiency, an attempt of properly combining the CNP with Levelled Commitment and the revised CNCP is made in order to enhance both the efficiency and the flexibility of the interaction protocol.

The combinatorial version of the CNP adaptation is called CNCP with Levelled commitment (Levelled CNCP). It adds the bidding strategy to improve the flexibility of choosing better "partner" and tries to avoid the sub-optimal problem that may occur in CNCP. At the same time, the repeated computation on future events' possibility in CNP with Levelled Commitment is tried to be partially simplified by CNCP's confirmation process which postpones the commitment time to evaluate and accept the new potential offers.

With the advantages of both extensions of CNP, Levelled CNCP is expected to be more robust and more flexible than the revised CNCP and CNP with Levelled Commitment when working in asynchronous MAS. In fact Levelled CNCP is aiming at being able to handle new agents up to the last stage of interactions.

The UML diagram of the Levelled CNCP is given in Figure 5-2. Again, as in figure 4-8, all deadlines are shown in this diagram.

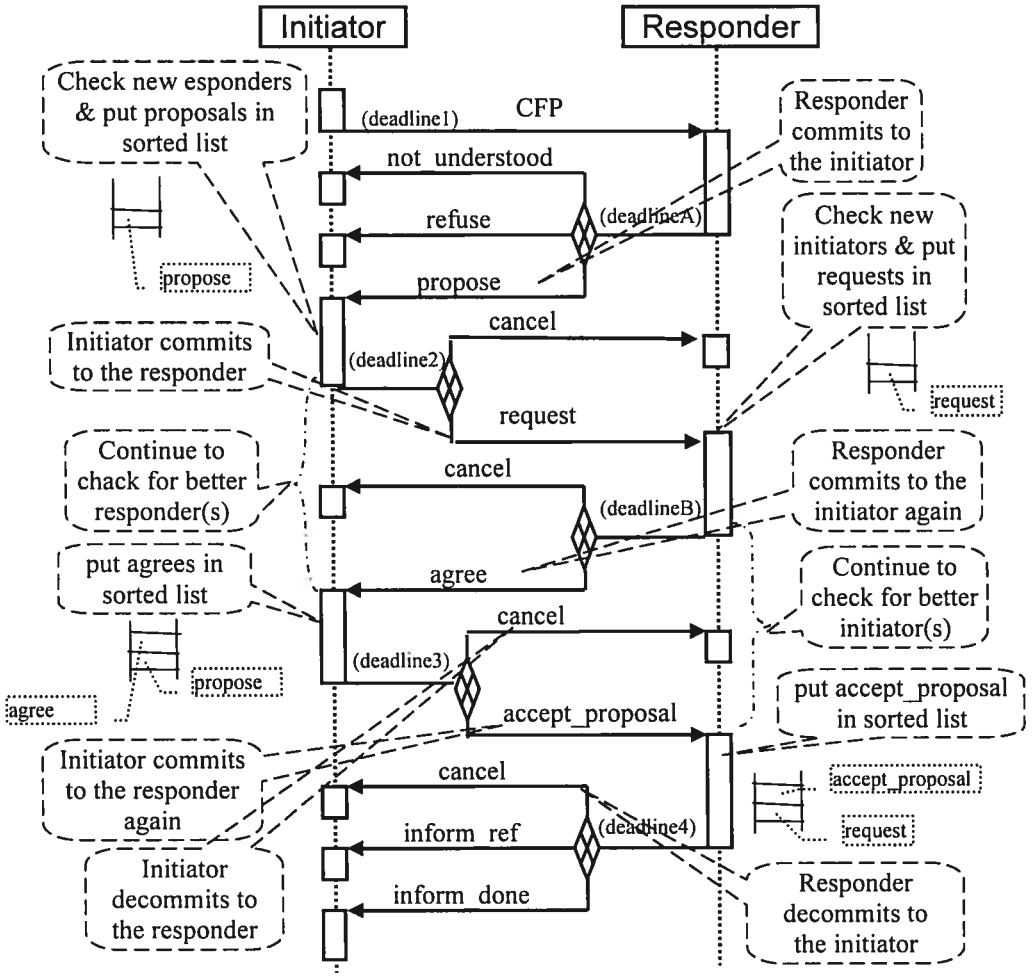


Figure 5-2 UML diagram of CNCP with Leveled Commitment

Like CNP with Levelled commitment, which doesn't change the structure of CNP much, in figure 5-2, the UML structure of Levelled CNCP is basically the same as that of the revised CNCP. The main difference is the strategy.

To understand more about the algorithm of levelled CNCP, figure 5-3 in the next page shows the flow diagram of an initiator and a responder.

In figure 5-3, after receiving an *agree* message, an initiator will put the message into the sorted list together with the *propose* messages. Then when it is ready to send, the initiator will check the top of the sorted list, if the top one is a proposal, it will send a corresponding *request*; if the top one is an *agree* message, it will send a corresponding *accept_proposal* message.

Similarly, after receiving an *accept_proposal* message, a responder will put the message into the sorted list with the *request* message. Then when it is ready to send, it checks the top of the sorted list, if the top one is a *request*, it sends an *agree* message; if the top one is an *accept_proposal* it replies with an *inform_done*.

Please note that both initiator and responder use a repetitive way to check for newcomers in the network. Whenever an initiator wants to draw a received message from the top of the sorted-list, it will check to see if there is any new responder coming into the system. If yes, it will send *CFP* to the new responder and temporarily keep the top of the sorted-list untouched. On the other hand if a responder sees a new initiator coming into the system at the beginning of any receiving stage, it will prepare for receiving a new *CFP*. If it really receives a new *CFP*, it will reply a *propose* message and temporarily keep the top of its sorted-list untouched. This way ensures both initiators and responders to be able to get the information of any late coming agents in an asynchronous system, and compare this information with that of the old agents.

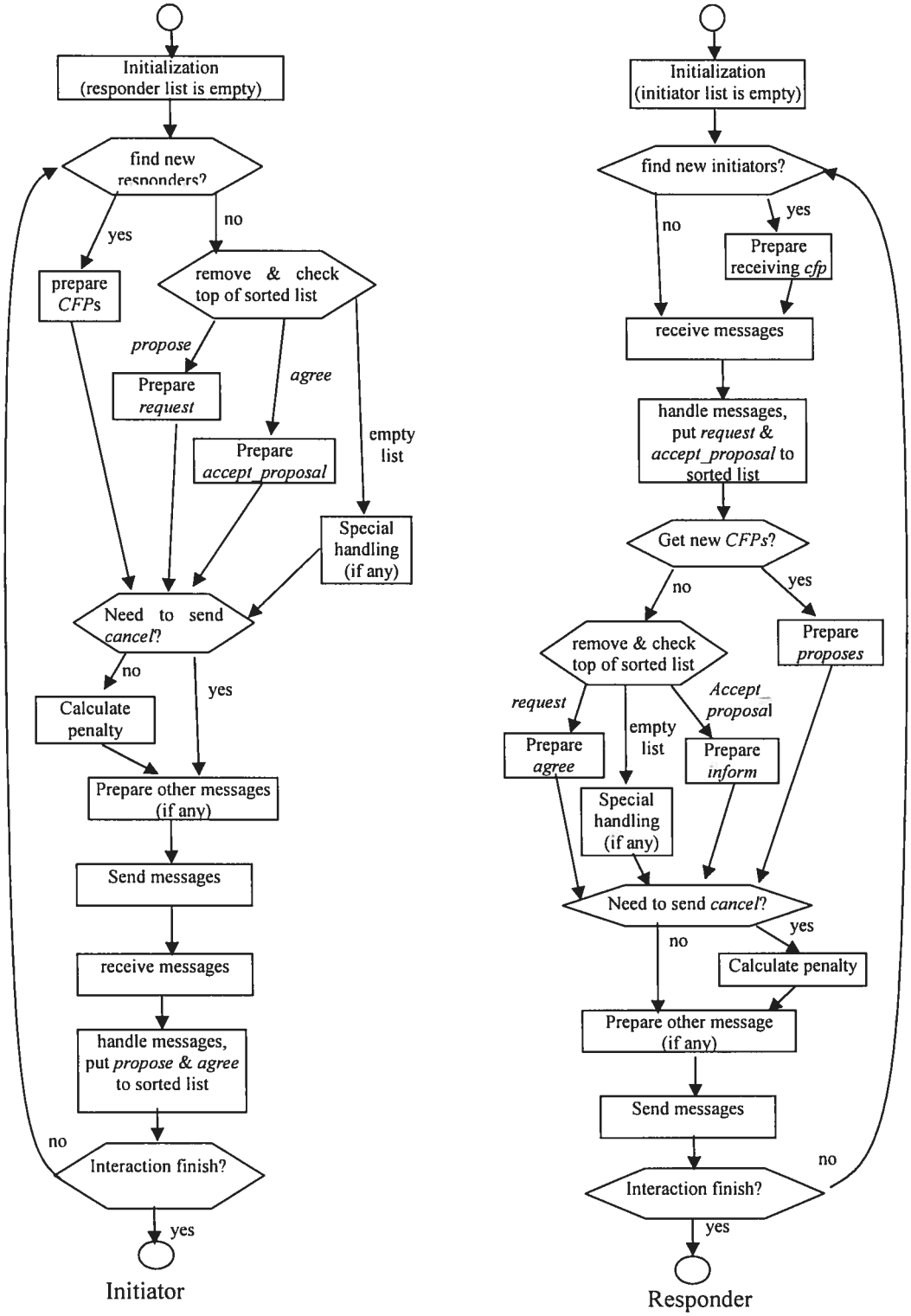
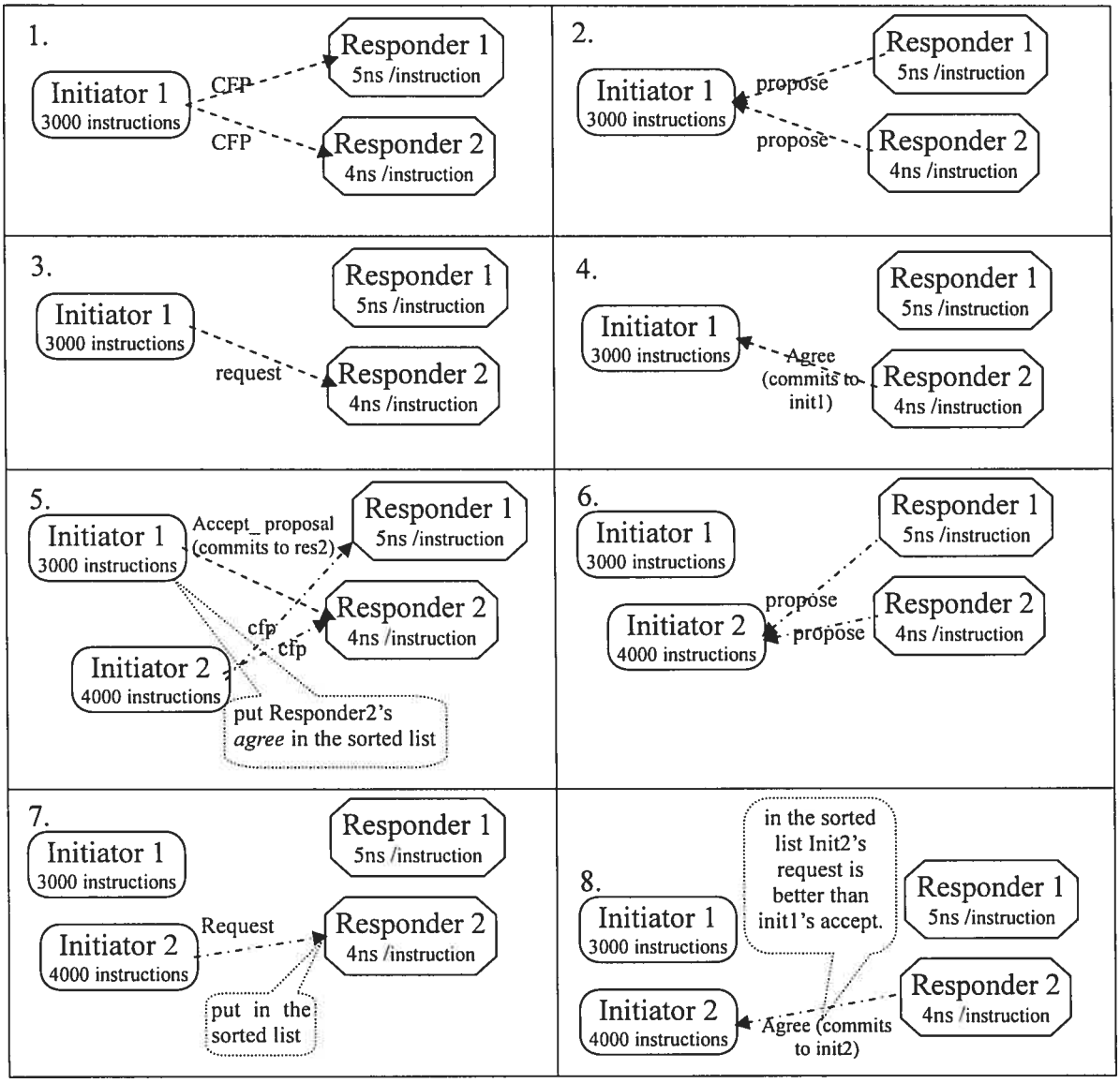


Figure 5-3 Flow diagram of CNCP with Leveled Commitment

Figure 5-4 below is an example which is similar to the one in figure 4-3 to demonstrate how Levelled CNCP works in such situation. In step 5 of figure 5-4, an agent **Initiator2** joints into the system and sends out *CFP* message when **Initiator1** sends out an *accept_proposal* message. Since in **Responder2**'s sorted list, **Initiator2**'s *request* and acceptance are better than **Initiator1**'s acceptance, **Responder2** only interacts with **Initiator2**, until it receives **Initiator2**'s acceptance, and then in step 10, it decides to set up a deal with **Initiator2**, and decommits to **Initiator1**.



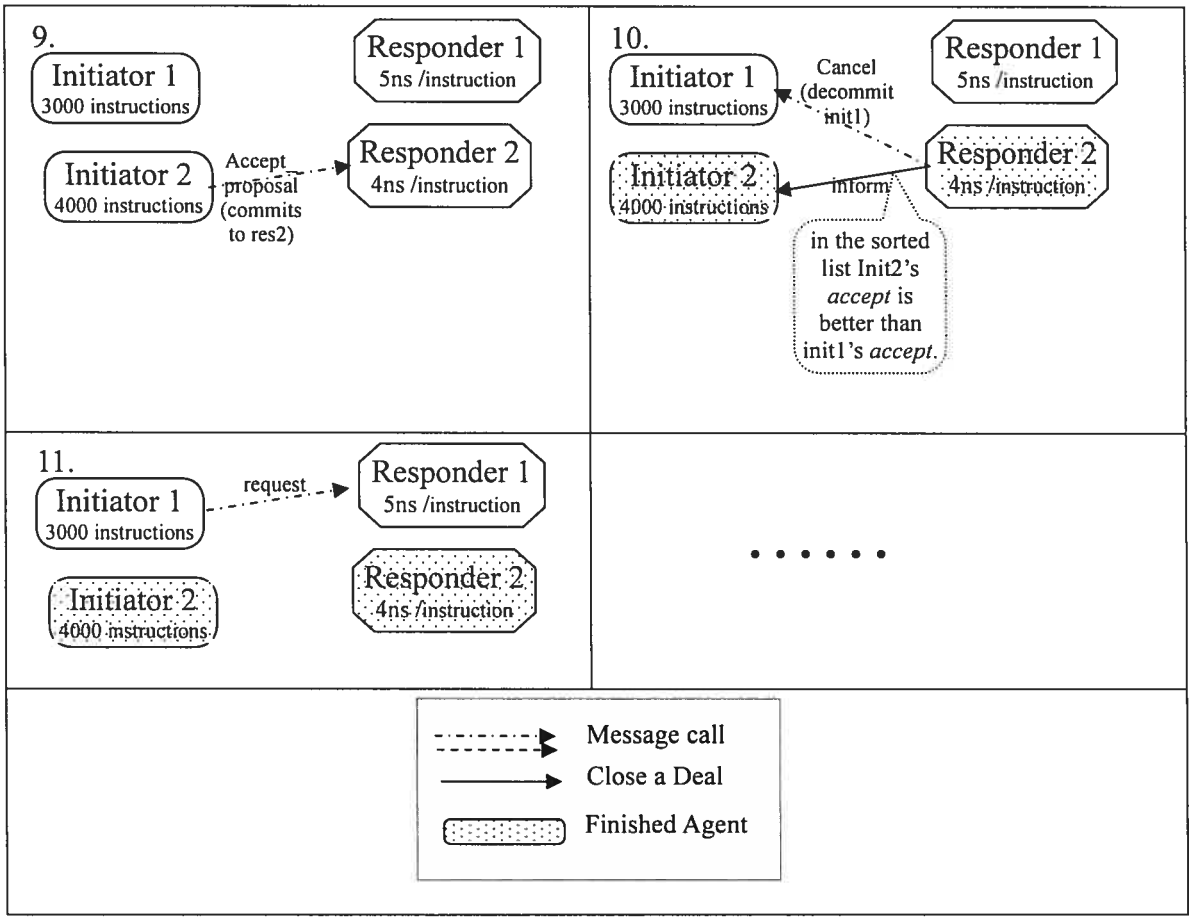
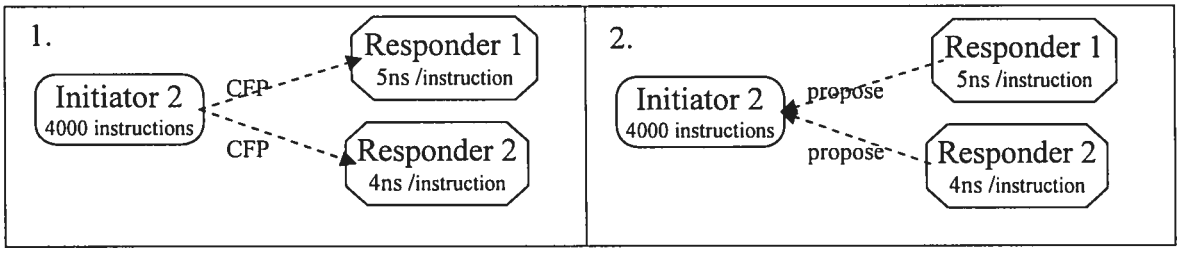


Figure 5-4 An example of applying CNCP with Leveled Commitment

Figure 5-5 below shows an alternative situation in which **Initiator1** joins to the system late. In step 8 of figure 5-5, since **Initiator2**'s acceptance is better than **Initiator1**'s *request*, **Responder2** decides to set up a deal with **Initiator2**, and refuses **Initiator1**.



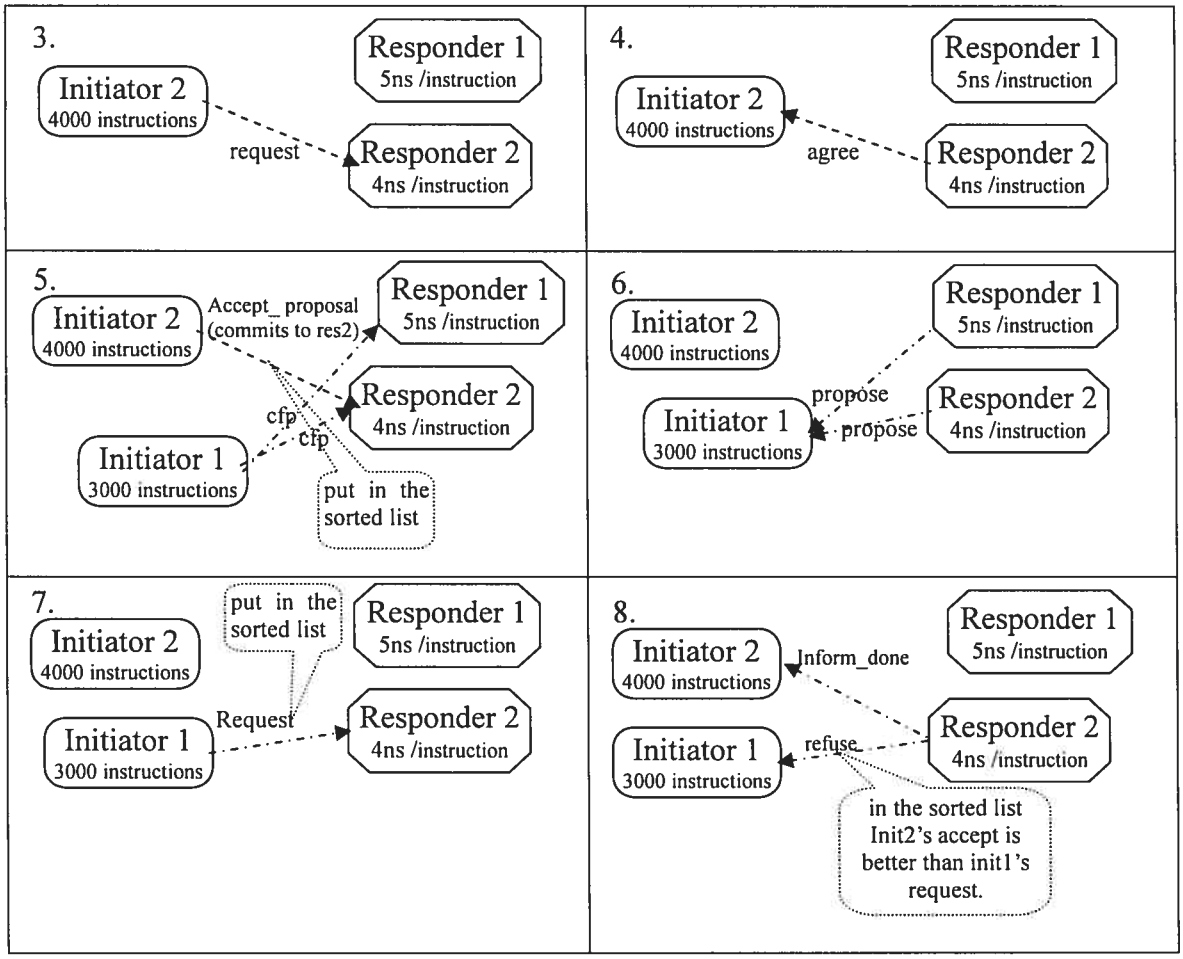


Figure 5-5 Another example of applying CNCP with Leveled Commitment

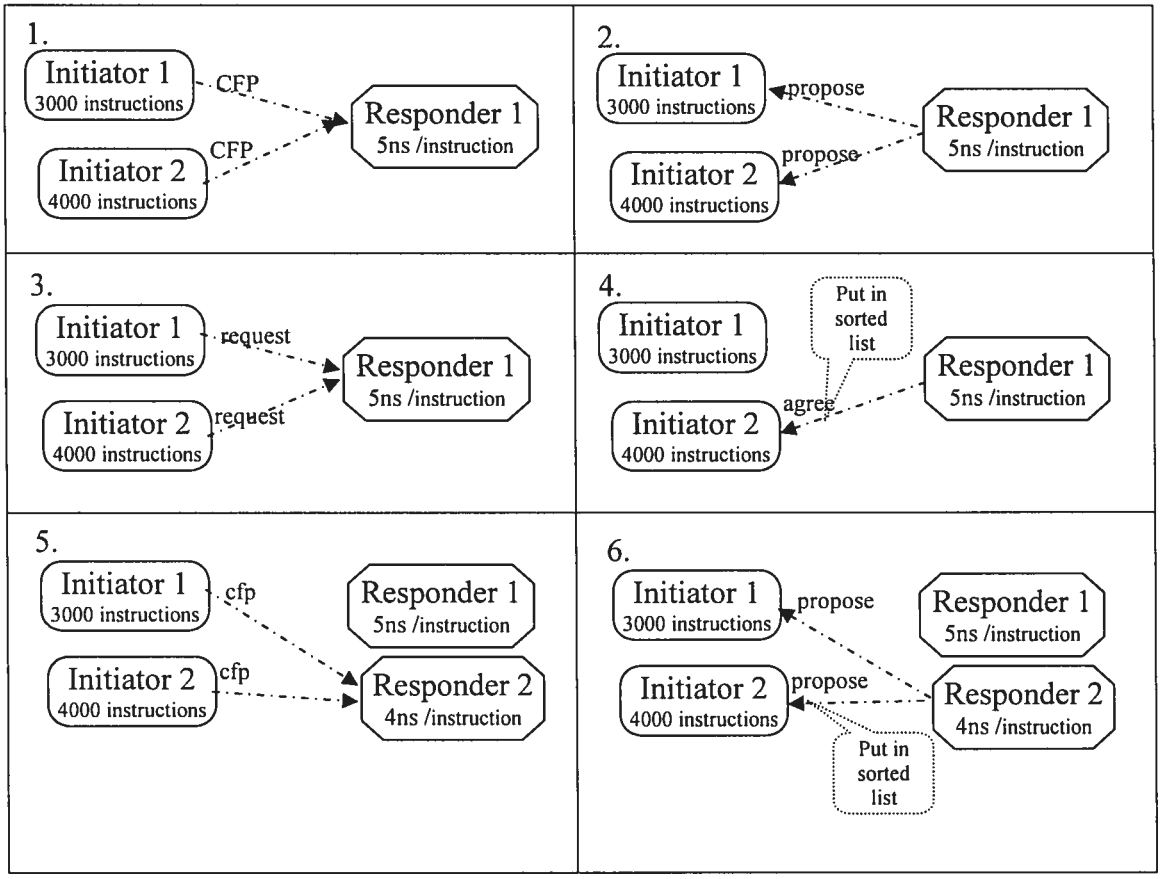
From the results of figure 5-4 and figure 5-5, one can see that both cases can lead to optimal allocation. It's very important for an initiator to put *agree* message and *propose* message in the same sorted list, so that it can decide according to the top of the sorted list what should be done next. Similarly it is very important for a responder to put *request* message and *accept_proposal* message in the same sorted list.

The implementation later shows that since Levelled CNCP can handle either new initiator(s) or new responder(s) up to the last stage of an interaction, it will solve the sub-optimal problems in asynchronous MAS as long as no more new coming

agents join into the allocation process after some contracts have been set up and finished.

For comparison, figure 5-6 shows a situation in which **Responder2** is coming to the system late. In step 5 of figure 5-6, both initiators realize that **Responder2** has joined into the system and then send *CFP*s to it. In such case, **Initiator2** keeps the *agree* message (sent by **Responder1**) in the sorted-list untouched.

Since **Responder2** has better computing speed than **Responder1**, in **Initiator2**'s sorted-list, **Responder2's** proposal and agreement are always on top of **Responder1**'s agreement. This is why **Reponder2** can finish first and can deal with the best initiator.



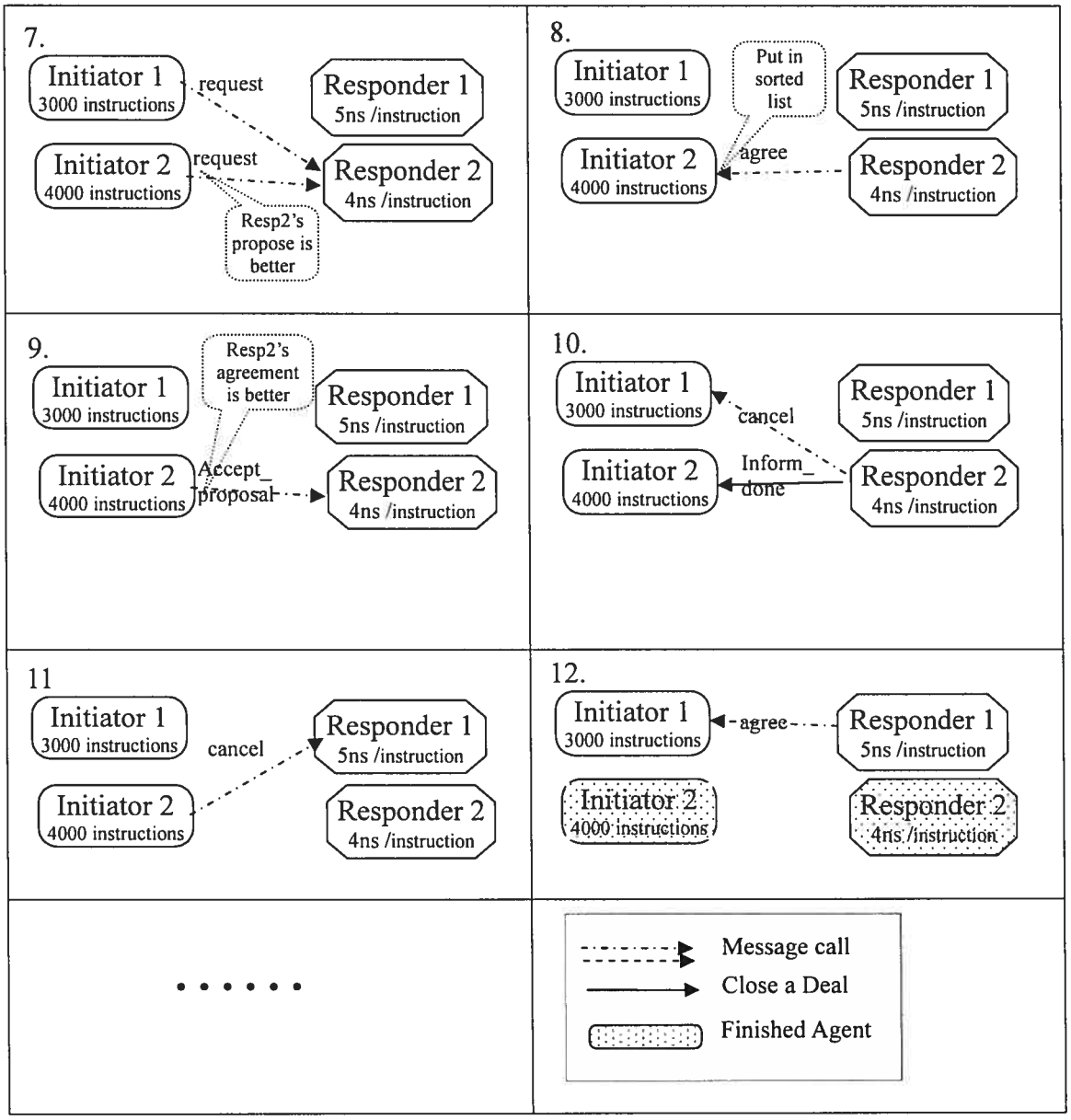


Figure 5-6 An example of applying CNCP with Leveled Commitment for newcomers

5.4 Further Discussion on the Levelled CNCP

In complicated situations where the future events will affect the result of CNCP, it's pretty difficult for the initiator to determine proper deadlines for collecting the proposals and the agreement or refuse from the responder in a large size dynamic system.

According to Sandholm's theory [14], the probability of future events should be calculated. This seems to be difficult to achieve for a complex open system because there are too many uncertain factors in the system and the actions of agents are hard to predict. So, even in the author's own implementation of the protocol, he just used the agents which will not consider any event that has not happened yet. Then, in my implementation, the experiments are also based on the performance of this kind of agent, so called myopic agent.

Even so, the implementation of the combinatorial protocol is still very complicated. The complexity lies not only in the requirement (for the agents) of being able to handle and response to all possible types (*CFP*, *REQUEST*, *AGREE*, *PROPOSE*, etc.) of messages at all receiving stages, but also in the requirements of tolerance to exceptional situations such as late coming messages (causing the receiving actions "time-out") and out of order messages. Those exceptional situations are mainly related to the specific network condition in which the experiments are running.

The implementation of the Levelled CNCP will be described in detail in the next chapter. To focus on describing the main features of the Levelled CNCP, the implementations of handling specific exceptional situations will not be described in detail.

Chapter 6 Implementation of protocols

The purpose of the implementation is to verify the theoretical analysis of the three protocols discussed above, and try to find out and fix any problems in handling exceptional situations that are not seen in the UML diagrams.

6.1 Implementing with JADE

As mentioned previously, the implementation is done with JADE. JADE has provided a set of behaviour templates for handling initiator and responder activities in the CNP, so the implementation of CNP becomes a matter of applying the class templates into the experiments.

Both CNCP and Levelled CNCP are implemented based on JADE's provided behaviour template ***FSMBehavior***, which allows users to define Finite State Machine (FSM) to control complicated agent behaviours. The benefit of using FSM for implementing the protocols is that the FSM has very good flexibility, extendibility and consistency, and is easier to be adapted to handle the complex requirements of protocols.

A FSM can be defined as the following [32]:

- A **FSM** $M \in S$ is a 3-tuple (Q, Σ, δ).
- Q is a finite set of states.
- Σ is a finite set of input/output symbols.
- δ: $Q \times B \rightarrow Q \times B$, here B is a set of zero or more symbols from Σ, it is the transition function.
- Empty input $\varepsilon \in \Sigma$ is a special input symbol, which is usually used for indicating a transition from one state to another without any input; but in the applications of JADE, this symbol means that a transition can occur with any input EXCEPT those having

been defined in other transitions.

To be able to use ***FSMBehavior***, it is necessary to transfer the protocols presented by UML diagrams to FSM State diagrams first. The main ideas in transferring each UML are:

i) Each UML diagram is separated into initiator behaviour part and responder behaviour part.
ii) Each behaviour is a FSM which repetitively send out a number of messages first, then continuously receive messages for a certain time period (up to deadline). All of the received messages will be treated according to the type and content individually. The result of treating those messages will be used for deciding the transition to the next state (i.e. loop back, abort, or finish, etc.).
iii) Each behaviour has a "NoMessage handling" state for handling the situation of no message received exception before the receiving deadline passes.

The complete FSM state diagrams for those protocols are too complicated to be presented, I attach them in appendix. Instead, a simplified version, shown in figure 6-1, roughly represents the internal state structure of a behaviour (either initiator behaviour or responder behaviour) of all protocols (CNP, CNCP, and Levelled CNCP). Note that figure 6-1 doesn't show any message communication with other agent(s). Message passing relations can be referred to the UML diagrams.

In figure 6-1, state "abort" and "finish" are the final states. If there is no expected message received before its corresponding deadline has passed, the state machine will transit into state "*NoMessage handling*". In this state the agent can either re-try to receive messages or simply abort. If the state machine ends up with state "abort", the agent may manage to start behaviour again by going through state "reset".

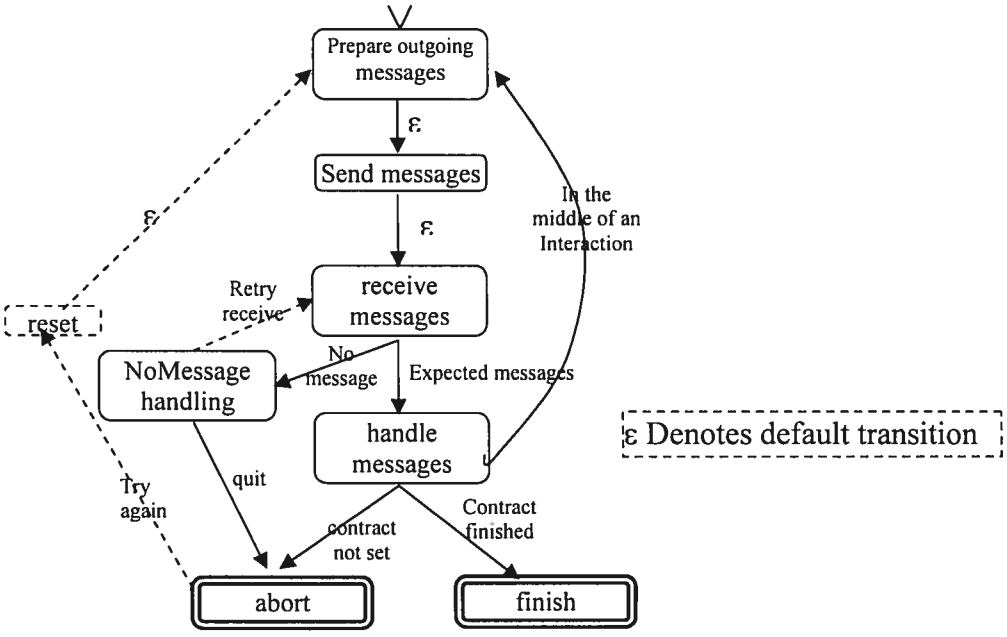


Figure 6-1 FSM State diagram for initiator/responder behaviour

Suppose that this FSM is representing a CNCP initiator behaviour, the procedures for the states to handle an interaction will be:

i) If the status of the FSM is at the beginning state, check for existing responder(s) through the *DF*[9] in the system and prepare *CFP* messages; otherwise just prepare other outgoing messages.
ii) Send out messages (*CFP*, *request*, etc.).
iii) Wait for receiving messages.
iv) Handle the received messages. If there's any proposal message, put the message into a sorted list.
v) Loop back to the first state.

For Levelled CNCP, the first state also checks new coming agents so that the FSM can be used in asynchronous agent systems.

[9]The Directory Facilitator (*DF*) is the agent that provides a yellow page service to the agent platform.[22]

6.2 Experiments Design

The experiments are designed to solve an example task allocation problem in which a number of computational tasks are looking for some servers to perform their calculations. The tasks can be considered as a set of self-interested agents. Each agent tries to maximize its own local resource utility. Typically, the experiments are separated into 10 groups. In each group, a number of initiators, representing the computational tasks with certain number of instructions, try to set up contract with an equivalent number of responders, which represent the servers with certain computing resource.

The number of initiators in each group varies from one to ten respectively, i.e. group one has one initiator and one responder; group two has two initiators and two responders; and so on. Each group has twenty experiments, ten for testing synchronous system and another ten for testing asynchronous system. When testing asynchronous systems, there are up to eight initiators or responder entering the runtime system late.

In each experiment, all three protocols are tested separately based on the same set of input data. A set of input data include data for responders (in nanoseconds per instruction), data for initiators (in number of instructions), and the delay time (in milliseconds) for each agent entering into the runtime system. The delay time is used for simulating asynchronous multi-agent systems.

In each test, a number of data including the time period consumed by each agent, the quantity of communication messages sent and received by each agent, information of the final deal of each agent, and etc. are recorded. These data are used for statistically analyzing and comparing the performance of the protocols:

1. **Average system time consumption vs. number of initiators (number of task) in synchronous systems** – the average time

consuming by each initiator is denoted by T_{avg}. It is supposed to increase together with the number of agents in synchronous systems. For asynchronous systems, the system time cost will be interfered by the joining of new members. In my experiments, the number of initiator is equivalent to the number of responders, and it's easier to calculate the average consumption of each initiator only.

Let us assume we are in the j^{th} experiment, if n is the number of initiators used in the experiment; t_{ij} is the time consumed by i^{th} initiator, $i \le n$; T_j is the average time consumed for this experiment. Then the average of consumed average time by n initiators is calculated by:

$$T_{avg} = \frac{\sum_{j=1}^{10} T_j}{10}, \quad \text{and} \quad T_j = \frac{\sum_{i=1}^{n} t_{ij}}{n}. \qquad (6.1)$$

2. **The message traffic vs. number of initiators** – the message traffic, S_{total}, is considering the total number of messages transmitted throughout the interactions. It is mainly used for comparing the network traffic load when applying different protocols. S_{total}, is calculated for both synchronous system and asynchronous system.

   Let M_{ij} be the total number of (both *in* and *out*) messages of i^{th} initiator; S_j be the sum of messages exchanged in j^{th} experiment; n be the number of initiators used in the experiment, then the average of total messages for n initiators is calculated as:

$$S_{tatal} = \frac{\sum_{j=1}^{10} S_j}{10}, \text{ and } S_j = \sum_{i=1}^{n} M_{ij}. \qquad (6.2)$$

3. **The resource utilization efficiency vs. number of initiators** – the definition of resource utilization efficiency is described in chapter 4. It

is the most important property for comparing the three protocols. The calculating for each protocol will be made in each testing group respectively.

Since the resource allocation result of each experiment can be either optimal or sub-optimal, the resource utilization efficiency is the percentage of the number of optimal allocation result versus the number of experiments in the group.

To see whether or not an allocation result is optimal, assume in an experiment with n initiators, in any contract with i^{th} initiator and j^{th} responder ($i,j \leq n$), Q_i is the number of instructions of the initiator, S_j is the computation speed of responder; then the sum of the time cost (number of instructions of the initiator times the speed of the responder) of all contracts is $T_{sum} = \sum_{i,j=1}^{n} Q_i \times S_j$; and then check if there is any alternative way of allocation that can give out a T'_{sum} that is lower than T_{sum}. If T_{sum} is the lowest one, then the experiment result is optimal.

Since Knabe's CNCP may lead to sub-optimal results pretty often, as described in chapter 4, my experiments will use the revised CNCP (in chapter 4) only for testing.

As mentioned previously, estimation of the decommitment penalties for Levelled CNCP is various according to the situations applying the protocol. To concentrate on the performance of the protocol itself, it's assumed that the penalties are zero.

The actual experiments and their results are described in the next chapter.

Chapter 7 Experiment results

This chapter describes and analyzes the results of a number of experiments for evaluating the performance of the three protocols. The experiments are done in a public laboratory (in the computer science department of University of Montreal) which has eleven PC computers. Those PCs are connected together with a computer network. When doing the experiments, initiators and responders are running in pairs, and each pair occupies one computer. In other word, every initiator is running with one responder on one JADE platform.

7.1 The average system time consumption vs. number of initiators

The time cost of each individual initiator, denoted by t_{ij} in formula 6.1, is recorded after every experiment. The result of T_{avg} is plotted as in figure 7.1 below.

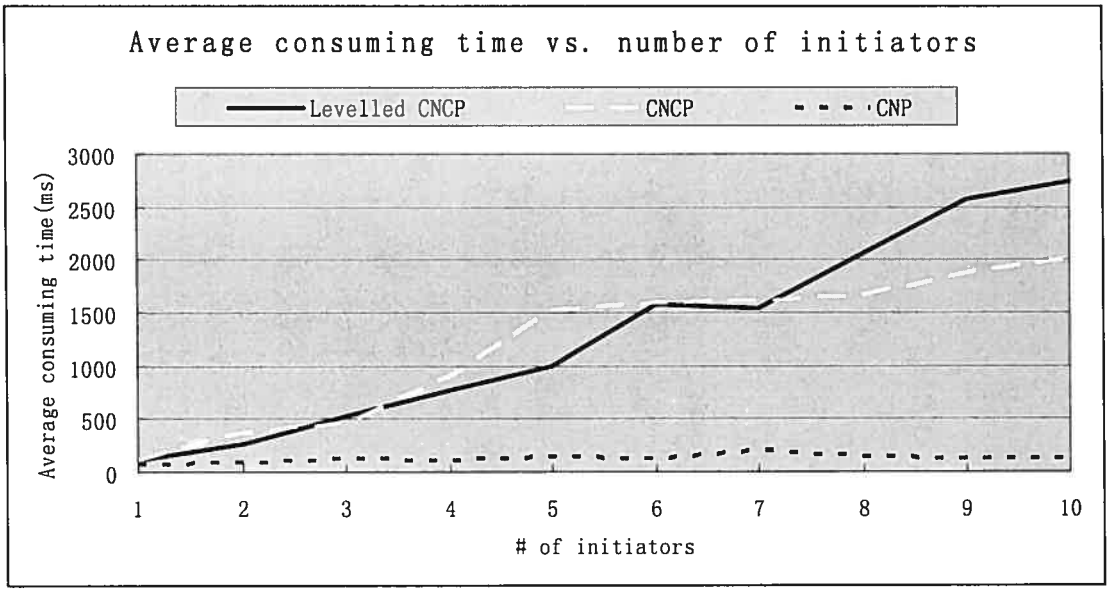


Figure 7-1 Results of Average system time cost vs. number of initiators

Figure 7.1 shows that the average system time cost (millisecond) in both CNCP and Levelled CNCP is increasing proportionately with the number of initiators. The increasing rate of Levelled-CNCP is about the same as that of CNCP. The average time consumption of each initiator in CNP is almost constant.

Analysis: For experiments of CNCP and Levelled CNCP, observation of the interaction processes shows that agents are interacting in the worst case, which is described in section 4.3.1 and figure 4-7. T_{avg} increasing linearly with the number of initiators in CNCP and Levelled CNCP is because the initiators are interacting with responders one by one. The more responders in the system, the more time they take to finish. Another factor is that with the increasing number of agents, the number of messages an agent can handle will also increase. The consequent result is that the time consumption for the initiator to finish the task assignment will increase accordingly.

Comparatively, for CNP, the result is the different. T_{avg} is close to a constant. Careful look at the result of resource allocation tells that all initiators set up contracts with the responders in their own computer. The mechanism of CNP determines that a responder just can receive one *CFP* message. In the experiments, a responder always receives the *CFP* coming from the initiator running on its own platform. Other *CFP* messages transmitted through the network won't have the chance to be picked up by the responder. This outcome is not very typical, but it tells that the performance of CNP is strongly affected by the physical condition of the network and the locations of resources. In fact we had tried by putting a few initiators and responders on one platform, and the results varied greatly and were hard to predict.

7.2 The total message traffic vs. number of initiators

The experimental result of the total number of messages for each initiator is substituted into M_{ij} of formula 6.2. The result of S_{total} for synchronous system is plotted as in figure 7-2 below, and the result for asynchronous system is plotted as in figure 7-3.

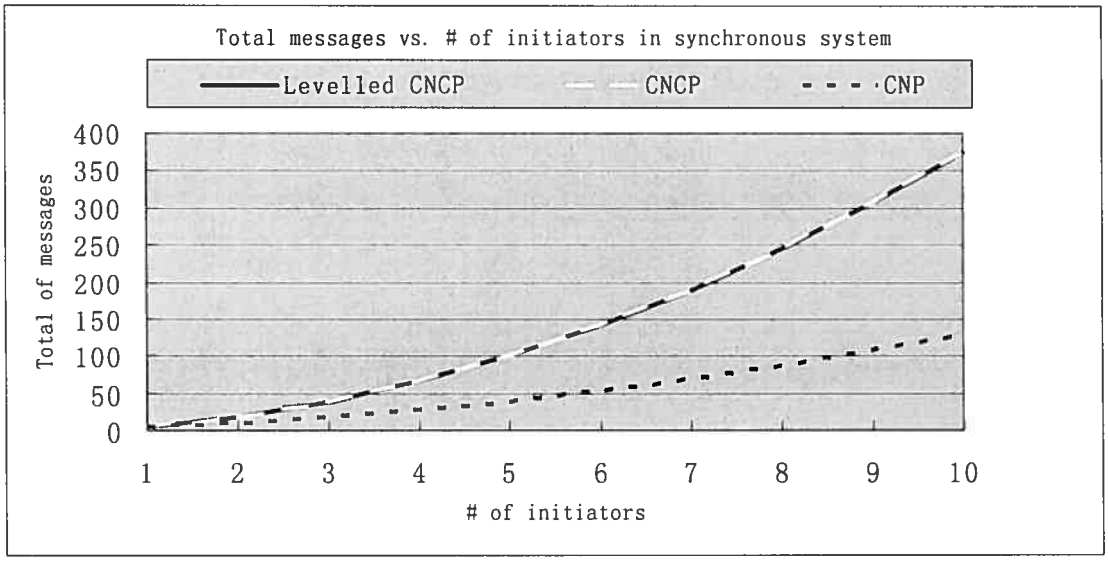


Figure 7-2 Results of message traffic vs. number of initiators in synchronous system

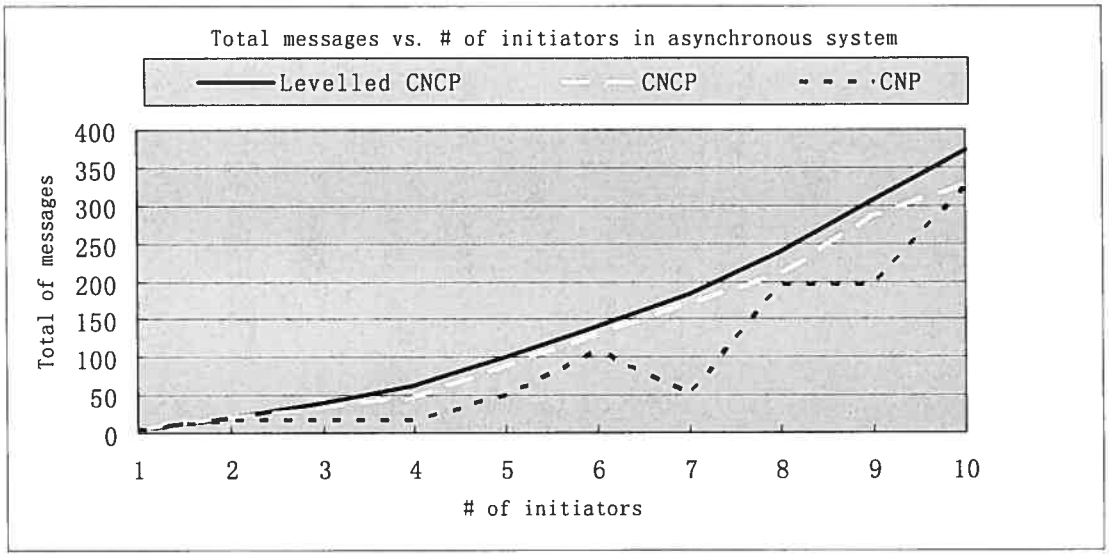


Figure 7-3 Results of message traffic vs number of initiators in asynchronous system

Figure 7-2 shows that the values of S_{total} of CNCP and Levelled CNCP are exactly the same and are proportionally increased with a polynomial function the number of initiators; while the value of S_{total} increases proportion to a polynomial function of the number of initiators.

In figure 7-3, the S_{total} of Levelled CNCP is almost the same as in figure 7-2, except that some places in the line drop a little bit. The S_{total} of CNCP drops more, but is still proportion to a polynomial function of the number of initiators. The S_{total} of CNP varies a lot and again approximately proportional to a polynomial function with a higher order.

Analysis: for each experiment in synchronous systems, the outcome is in fact reflecting the analytical results of formula 4.1 (CNP) and formula 4.4 (CNCP) in chapter four. As mentioned in the analysis of section 7.1, CNP is doing the case that gives the minimum number of traffics; and CNCP is doing the case that gives the maximum number of traffics. Note that in the experiments, the number of initiators equals the number of responders, and formula 4.1 to 4.4 can now be simplified to single-variable polynomials. Because there's no new agent coming into the system, the processes of Levelled CNCP is exactly as CNCP.

For experiments in asynchronous systems, the values of S_{total} of CNP vary very much. Since most of the agents join into the system after the system starts, as mentioned in section 4.3.1, these agents interact in the way that generates the maximum number of messages. For CNCP, since some agents have passed the stage of handling new members, they will not interact with the newcomers. In such cases S_{total} will drop down. In Levelled CNCP, the case of drop-down of S_{total} also happens a few times, but not as many as in CNCP. This is because agents in Levelled CNCP can handle late newcomers up to the last stage of the interactions.

Table 8-1 gives a clearer summarization on the relationships between the message traffic in different protocols and the number of agents.

7.3 The resource utilization efficiency vs. number of initiators

In an experiment, testing whether or not a resource allocation result is optimal takes a lot of work, but careful selecting the set of input data can save a lot. Table 7.1 shows the results in synchronous systems and table 7.2 shows the results in asynchronous systems (notice here, 10% means one optimal result out ten experiments, 20% means two optimal results out of ten experiments, and so on).

1) In synchronous systems:

# of initiators	1	2	3	4	5	6	7	8	9	10
CNP	100%	50%	50%	0	0	0	0	0	0	0
CNCP	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Levelled CNCP	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 7-1 Results of resource utilization efficiency vs. number of initiators in synchronous system

Table 7.1 shows that in synchronous systems, the resource utilization efficiencies of the CNCP and Levelled CNCP systems are all optimal. But in CNP systems, the resource allocation results are usually sub-optimal.

2) In asynchronous systems:

# of initiators	1	2	3	4	5	6	7	8	9	10
CNP	100%	50%	0	0	0	0	0	0	0	0
CNCP	100%	70%	40%	30%	10%	0	10%	0	0	0
Levelled CNCP	100%	100%	100%	90%	100%	100%	90%	80%	100%	90%

Table 7-2 Results of resource utilization efficiency vs. number of initiators in asynchronous system

Table 7.2 shows that the resource utilization efficiency of CNP in asynchronous systems is as bad as in synchronous systems; the efficiency of Levelled CNCP is

the best; the efficiency of CNCP drops significantly but still better than that of CNP. Another observation tells that the utilization efficiencies of CNCP and Levelled CNCP decrease as the number of initiators increases.

Analysis: the resource utilization efficiencies of CNCP and Levelled CNCP are optimal in all synchronous agent system experiments. This is because during interactions, all initiators always send *request*s to the only responder that has the best speed. When this responder has closed the deal with the "best" initiator, all of the rest initiators will send request to the second best responder, and so on. Both initiator and responder have the right to choose the best one to set up a contract with, and then the both side of a contract are always the best available one in the current system. It is not always the case in other synchronous systems, but if the network is good, it is very likely that they are also optimal.

Experiments of Levelled CNCP in asynchronous system have very high utilization efficiencies. This result is depending on the delay time setting, when the newcomer enters the system too late, some responders may have finished the task already, and then the newcomer will just have the little chance to communicate to others. Anyhow, on average Levelled CNCP is much better than CNCP.

Table 8-1 also gives a clearer summarization on the relationships between the resource utilization efficiency in different protocols and the number of agents.

7.4 Discussion

Through the experiment results, one can see that: among the three protocols, Levelled CNCP is the best from the resource utilization efficiency point of view, but it generates the largest message traffic in the network and, from my personal

experience, is very hard to implement – to accommodate the flexible systems, many things need to be considered. While CNP is very simple to implement and is perfect for the situation that single initiator existing at one time, it is not very practical for the real world's complex open systems. Finally, CNCP's performance in open systems is in between the previous two.

7.4.1 The effect of deadlines

The length of the deadlines will affect the experiment result greatly. Basically if the number of agents in the network is fixed, then the average consuming time of an agent will increase linearly with the increase of the deadlines. But the values deadlines cannot be too small. The experiment results show that if receiving deadlines are too small, the average time will be mainly determined be network conditions including the program processing time and network transmission time.

For example, when the deadline is set to be 1 millisecond, to finish a Levelled CNCP interaction is about 60 milliseconds in a two initiators and two responders system; and if the receiving deadline is set to less than 60 millisecond, the results of the experiments for Levelled CNCP are hard to be controlled, i.e. for the same set of input data, the results are always different.

Similar situation happens in implementation of the other two protocols. Therefore, the deadlines are usually set to large enough to get relatively stable results.

Chapter 8 Conclusion

This thesis studies the Contract Net protocol and some of its extensions. Based on the two existing extensions, Contract Net with Confirmation Protocol (CNCP) and CNP with Levelled Commitment (Levelled CNP), a new combinatorial extension is proposed here as CNCP with Levelled Commitment (Levelled CNCP).

8.1 Summary

The Contract Net protocol (CNP), a widely used interaction protocol, is originally designed to be used for one task to be allocated among multiple agents. If there are multiple agents interacting in this way to have their tasks assigned concurrently, the limitation of this protocol's mechanism leads the interactions to suboptimal outcomes.

Contact Net Protocol with Confirmation (CNCP) is a variation of Contract Net Protocol. It opens the door for multiple initiators and multiple responders interacting to each other simultaneously. By adding a Request-Agree interaction stage, the problem for allocating resources to multiple initiators is solved. Despite the advantages of the original CNCP proposed by Knabe et al, their model cannot allocate resources to multiple initiators optimally. This problem is solved by allowing responders to sort the *request* messages and choose the best one for commitment. This revision of the protocol is able to well solve the sub-optimal problems in synchronous multi-agent systems. Yet in asynchronous systems, even the revised CNCP cannot solve the sub-optimal problem.

The author then tries to find a proper way to combine the idea of Levelled-Commitment contract together with CNCP protocol, so that the new protocol, namely Levelled-CNCP, can solve sub-optimal problem in asynchronous systems as long as new agent(s) do not appear too late.

The implementation section of the thesis describes the procedures to build the CNP, CNCP and Levelled-CNCP finite state machines (FSM) behaviour models based on UML diagrams. This section also roughly explains the state structures in those models. Simplified state diagrams are drawn to demonstrate the complicated relationships between states, transition functions and the messages flowing among the states of initiators and responders.

Despite the existing of concepts for those models, the actual structures built in the implementations may have different variations.

The experiments done for the thesis try to observe and compare the characteristics of CNP, CNCP and Levelled-CNCP in the following properties:

- average consumed time vs. number of initiators in synchronous systems;
- the message traffic vs. number of initiators in synchronous systems;
- the message traffic vs. number of initiators in asynchronous systems;
- the resource utilization efficiency vs. number of initiators.

The experiment results are carefully analyzed and compared with theoretical calculations. Overall, the conclusions about the protocols are summarized in the following table.

Protocol	**CNP**	**CNCP**	**Levelled CNCP**
Suitable MAS	1:M (single initiator)	N:M (multi-initiator synchronous)	N:M (multi-initiator asynchronous)
Average consuming time in synchronous systems	Network conditions and resource location dependant	$\propto N$	$\propto N$
Message traffic in synchronous systems	$\propto N^2$	$\propto N^2$	$\propto N^2$
Message traffic in asynchronous systems	$\propto (N^2 \sim N^3)$	less than N^2	approximately $\propto N^2$

Resource utilization efficiency in synchronous system	ok in 1:M systems, very low in N:M systems	High	High
Resource utilization efficiency in asynchronous system	Very low	Low	High
Flexibility	low	Medium	High
Difficulty of implementation	Easy	Normal	difficult

Table 8-1 Summarization of the differences of the three protocols

8.2 Future works

While many studies are still exploiting the variations of CNP, future studies about Levelled-CNCP can focus on understanding various structures of the model and their preferable applications. There is still some future works needed to do to make the Levelled-CNCP model proposed in this thesis to become a mature practical protocol. Features such as adding proper strategy on evaluating the effect of penalty of cancellations can be considered to the future implementation.

To further improve the flexibility of the protocol, the calculations on possibility of the future events and the risk probability of withdrawing (cancelling) a contact can be also include this model. While this kind of complicated calculation depends on the computer system's capability and the algorithms to be used.

Bibliography

[1] Jack Krupansky, http://www.agtivity.com/.

[2] The RoboCup Federation, http://www.robocup.org/.

[3] Reid G. Smith. *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*. IEEE Transactions on Computers, Volume C-29, page 1104-1113, 1980.

[4] B. Chaib-draa, B. Moulin, R. Mandiau, and P. Millot. "Chapter 1 - Trends in Distributed Artificial Intelligence", in *Foundations of Distributed Artificial Intelligence*, G. M. P. O'Hare and N. R. Jennings, John Wiley & Sons Inc, page 3-55, 1996.

[5] Guofei Jianga, Wayne Chungb, and George Cybenkob. *Semantic Agent Technologies for Tactical Sensor Networks*. SPIE, Volume 5090, page 311-320, 2003.

[6] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[7] Michael R. Genesereth, Steven P. Ketchpel. *Software Agents*. Communications of the ACM, v37, Issue 7, page 48-53, 1994.

[8] Cristiano Castelfranchi. *Guarantees for autonomy in cognitive agent architecture*. Intelligent Agents: Theories, Architectures, and Languages (LNAI), v890, page 56-70, 1995.

[9] Bryan Horling, Victor Lesser, Régis Vincent, Ana Bazzan, and Ping Xuan. *Diagnosis as an Integral Part of Multi-Agent Adaptability*. DARPA Information Survivability Conference and Exposition, page 211-219, 1999.

[10] Les Gasser. *Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems*. Artificial Intelligence, v47, Issue 1-3, page 107-138, 1991.

[11] Edmund H. Durfee and Thomas A. Montgomery. *MICE: A Flexible Testbed for Intelligent Coordination Experiments*. Ninth Workshop on Distributed Artificial Intelligence, page 25-40, 1989.

[12] Parunak, H.Van Dyke, "Chapter 4 - Applications of Distributed Artificial Intelligence in Industry", in *Foundations of Distributed Artificial Intelligence*, G. M. P. O'Hare and N. R. Jennings, John Wiley & Sons, page 139-163, 1996.

[13] N. R. Jennings, "Chapter 6 - Coordination Techniques for Distributed AI", in *Foundations of Distributed Artificial Intelligence*. G. M. P. O'Hare and N. R. Jennings, John Wiley & Sons, page 187-210, 1996.

[14] Tuomas W Sandholm, Sandeep Sikka, and Samphel Norden. *Algorithms for optimizing Levelled commitment contracts*. International Joint Conference on Artificial Intelligence, 1999.

[15] Martin R. Andersson, and Tuomas W. Sandholm. *Levelled Commitment Contracts with Myopic and Strategic Agents*. Third International Conference on Multi-Agent Systems, page 26-33, 1998.

[16] Burt Wilsker. *A Study of Multi-Agent Collaboration Theories*. ISI Research Report, ISI/RR-96-449, 1996.

[17] P. Cohen and H. Levesque. *Confirmation and Joint Action*. International Joint Conferences on Artificial Intelligence (IJCAI'91), page 951-957, 1991.

[18] Barbara J. Grosz, and Sarit Kraus. *Collaborative plans for complex group action*. Artificial Intelligence, v86, issue 2, page 269-357, 1996.

[19] D. Kinny, M. Ljungberg, A. Rao, E. Sonenberg, G. Tidhar, and E.Werner.

Planned Team Activity. Fourth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI Volume 830, 1992.

[20] Martha E. Pollack. *Plan as Complex Mental Attitudes.* Intentions in Communication, MIT Press, 1990.

[21] Bernhard Bauer, Jorg P. Müller, and James Odell. *Agent UML: A Formalism for Specifying Multiagent Interaction.* International Journal of Software Engineering and Knowledge Engineering, v11, issue 3, page 207-230, 2000.

[22] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. *JADE – A FIPA-compliant agent framework.* Practical Application of Intelligent Agents and MultiAgents, pages 97-108, 1999.

[23] Zoltan Juhasz and Prasenjit Paul. *Scalability Analysis of the Contract Net Protocol.* 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), page 319-320, 2002.

[24] Saris Kraus, Jonathan Wilkenfeld and Gilad Zlotkin. *Multiagent Negotiation Under Time Constraints*. Artificial Intelligence, v75, issue 2, page 297-345, 1995.

[25] Frank Teuteberg, and Karl Kurbel. *Anticipating Agents' Negotiation Strategies in an E-marketplace*. 5th International Conference on Business Informations Systems, page 91-100, 2002.

[26] Dongmin Kim, and Izak Benbasat. *Trust-Related Arguments in Internet Stores: A Framework for Evaluation*, Journal of Electronic Commerce Research, v4, issue 2, page 49-64, 2003.

[27] JADE Board. *JADE Security Add-On GUIDE*. http://jade.tilab.com/doc/SecurityAdminGuide.pdf

[28] Tore Knabe, Michael Schillo, and Klaus Fischer. *Improvements to the FIPA*

Contract Net Protocol for Performance Increase and Cascading Applications. The 25th German Conference on Artificial Intelligence (KI-2002), 2002.

[29] Chen Xueguang, and Song Haigang. *Further Extensions of FIPA Contract Net Protocol: Threshold plus DoA.* ACM Symposium on Applied Computing, SESSION: Agents, interactions, mobility, and systems (AIMS), Page 45-51 2004.

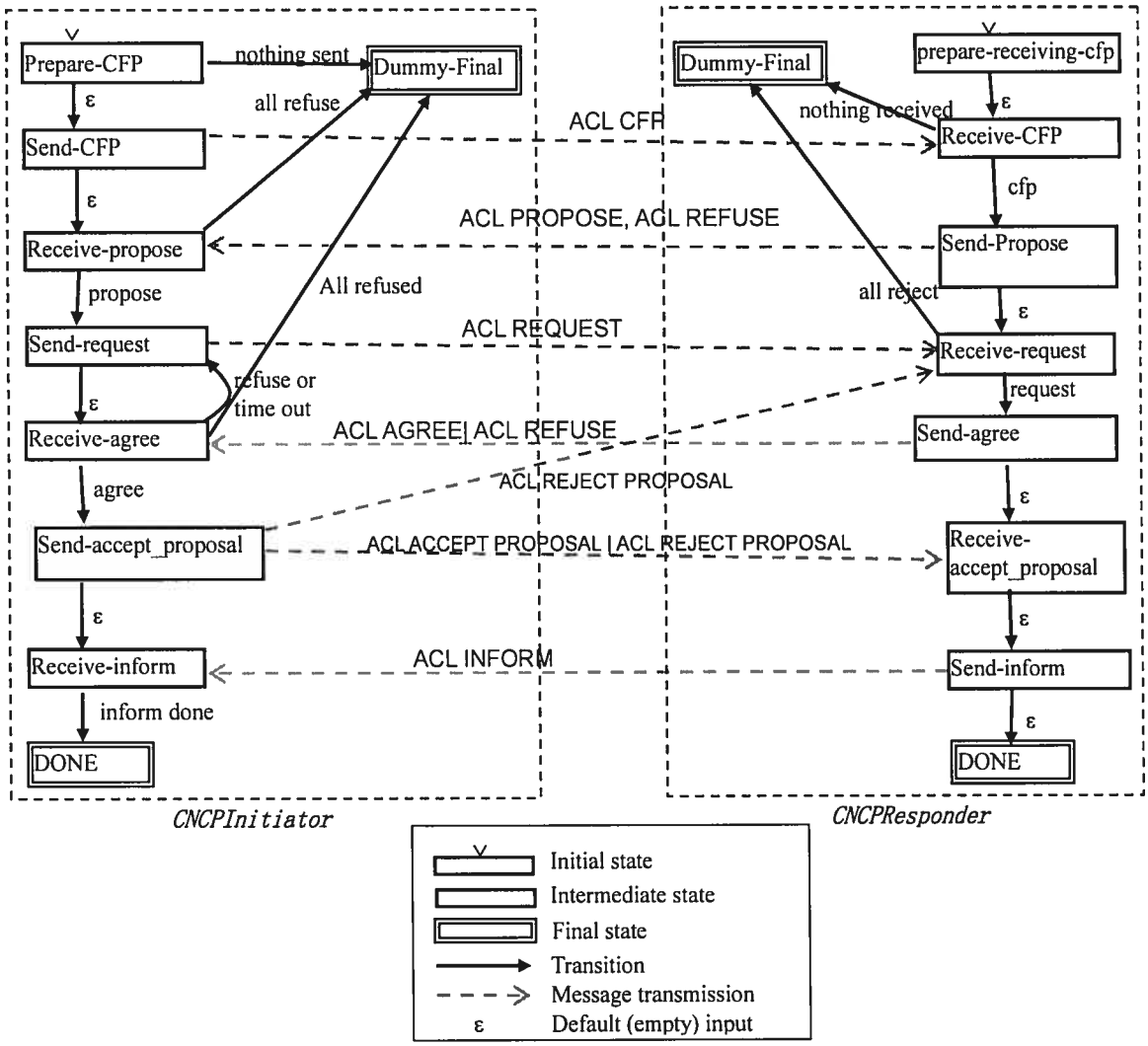
[30] David K. Levine, *What is Game Theory?* http://levine.sscnet.ucla.edu/general/whatis.htm, 1999

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

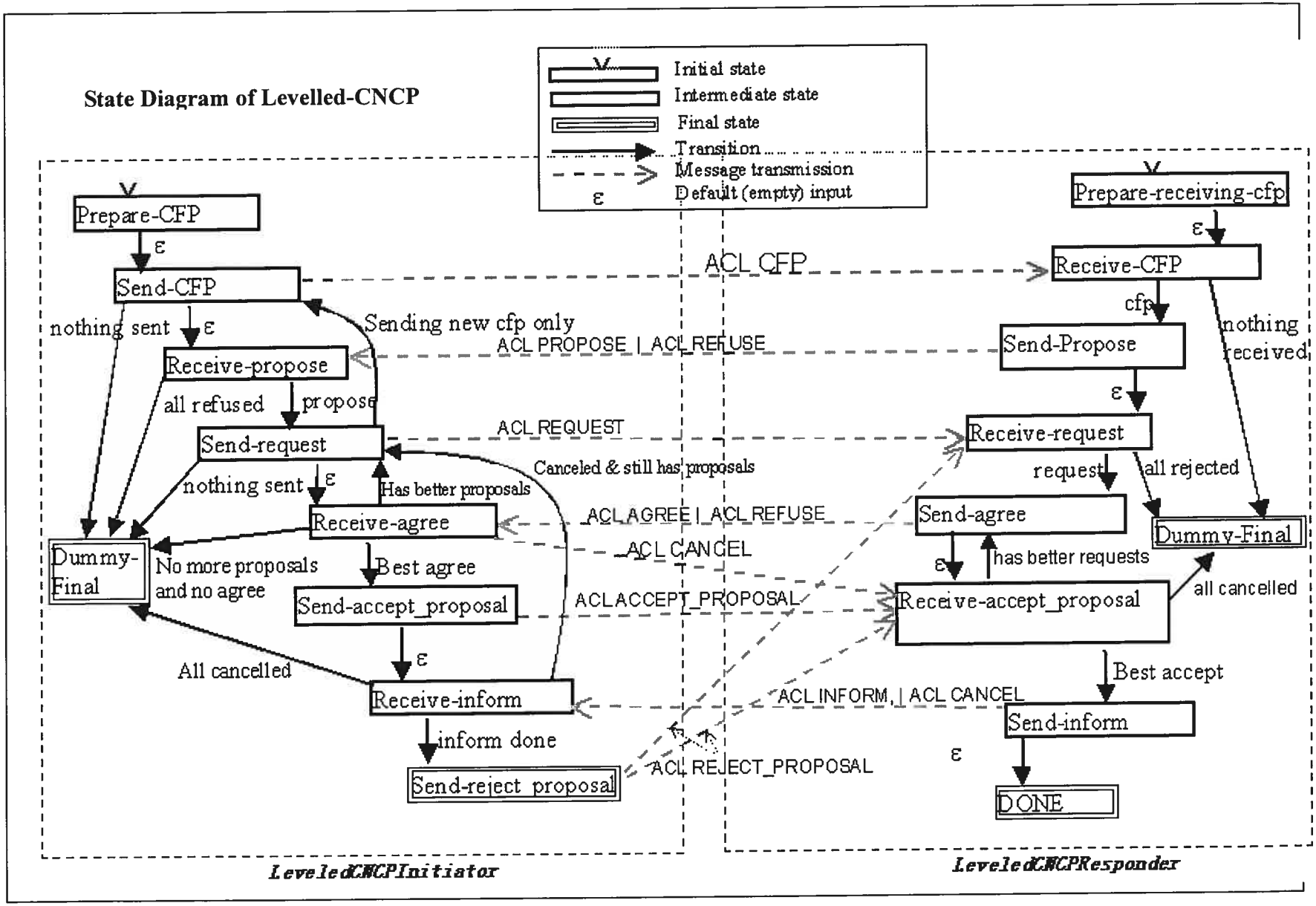
[32] David Owen et.al. *What Makes Finite-State Models more (or less) Testable?* 17th IEEE International Conference on Automated Software Engineering, page 237-240, 2002.

[33] R. S.Cost, Y. Chen, T. Finin, Y.Labrou, and Y. Peng. *Using Coloured Petri Nets for a Conversation Modeling*, in *Issues in Agent Communications*, F. Dignum and M. Greaves, page 178-192, Springer-Verlag, 2000.

Appendix



State diagram of CNCP


State Diagram of Levelled-CNCP
Initial state
Intermediate state
Final state
Transition
Message transmission
Default (empty) input
ε
Prepare-CFP
Send-CFP
nothing sent
Receive-propose
all refused
propose
Send-request
nothing sent
Has better proposals
Receive-agree
Dummy-Final
No more proposals and no agree
Best agree
Send-accept_proposal
All cancelled
Receive-inform
inform done
Send-reject_proposal
Sending new cfp only
Canceled & still has proposals
ACL CFP
ACL PROPOSE | ACL REFUSE
ACL REQUEST
ACL AGREE | ACL REFUSE
ACL CANCEL
ACL ACCEPT_PROPOSAL
ACL INFORM, | ACL CANCEL
ACL REJECT_PROPOSAL
Prepare-receiving-cfp
Receive-CFP
cfp
nothing received
Send-Propose
Receive-request
request
all rejected
Send-agree
has better requests
Dummy-Final
Receive-accept_proposal
all cancelled
Best accept
Send-inform
DONE
LeveledCNCPInitiator
LeveledCNCPResponder