

平成 22 年度 修士学位論文

Cooperation Protocol Design Method for Repository-based Agent Framework

東北大学大学院 情報科学研究科
情報基礎科学専攻
博士課程前期 2 年の課程
コミュニケーション論講座
木下研究室

A9IM4006
Wenpeng WEI

平成 23 年 2 月 16 日

Contents

Chapter1	Introduction	1
1.1	Background	1
1.2	Case Study: Comparison of Multi-Agent System Development in DASH and other Framework	3
1.3	Multi-Agent System Development Support Mechanism of DASH . .	5
1.4	Overview of This Research	6
1.5	Organization of the Thesis	7
Chapter2	Related Works	8
2.1	Existing Multi-Agent System Development Methodologies	8
2.2	Common Multi-Agent System Development in DASH	10
2.3	Summary of Chapter 2	11
Chapter3	Proposal of Cooperation Protocol Design Method for Repository-based Agent Framework	12
3.1	Protocol Template Mechanism	13
3.2	Protocol Design Workflow for DASH	14
3.3	Protocol Template Design Method	15
3.3.1	Step 1, Role and Sequence Design	16
3.3.2	Step 2, State Design	17
3.3.3	Step 3, Implementation	20
3.4	Summary of Chapter 3	23

Chapter4 Implementation, Experiment and Evaluation	24
4.1 Implementation 1: Protocol user (non-expert developer) support function	24
4.2 Implementation 2: Protocol designer (expert developer) support function	26
4.3 Experiment : Reduction of coding work	31
4.4 Summary of Chapter 4	35
Chapter5 Conclusion	36
5.1 Conclusion	36
Acknowledgements	38
References	39
Publication	42
Appendix A	44
Appendix B	50

List of Figures

1.1	Basic Model of Agent	2
1.2	Comparison of JADE and DASH	4
1.3	Software reuse at rule level in DASH	5
2.1	Comparison on development lifecycle coverages of multi-agent system methodologies	9
2.2	Comparison on agent nature requirements of multi-agent system methodologies	10
2.3	Multi-Agent System Development in DASH	11
3.1	Overview of the proposal	12
3.2	Protocol templates using repository	13
3.3	Protocol Design Workflow for DASH	14
3.4	Sequence Diagram	17
3.5	DASH State Diagram Component	18
3.6	DASH State Diagram of Role Participant of Contract Net Protocol Template	19
3.7	Meta Model of the DASH State Diagram	21
3.8	DASH State Diagram to Model Mapping	22
3.9	DASH State Diagram Model to DASH Rule Set Code Mapping	22
4.1	Protocol user (non-expert developer) support function	26

4.2	User interface of the graphical design tool for the DASH State Diagram design	28
4.3	Detailed images of Tool Palette and Properties Editor	29
4.4	“Generate ⇨ Dash Rule Set” menu	29
4.5	DASH rule set code file	30
4.6	Contract Net Protocol	32
4.7	DASH State Diagrams and DASH Rule Set files	33
4.8	MicroGrid Control System using the generated protocol templates .	34
4.9	Code comparisons between old implementations and proposed method	34
A.1	Illustrates the scale of the details within each development phase . .	44
A.2	Presents the measure of agent concept that each methodology support	44
A.3	Shows the scale of the modeling criteria within each methodology .	45
A.4	Compares the properties of the methodologies	45
A.5	Illustrates the available activities in each development phase	46
A.6	Illustrates the type of the system domain that each methodology is suitable for	46
A.7	Summarizes the toolkits that are available for each methodology . .	46
A.8	Comparison of Concept	47
A.9	Comparing a methodology ’s properties, attributes, process and pragmatics	48
A.10	Comparing methodology ’s properties, attributes, process and pragmatics	49

List of Tables

1.1	Comparisons of the status of development processes	5
3.1	Steps, products and supporting tools of proposed design method . .	15
4.1	The result of the comparison of lines of code between the old implementation and proposed design method	35

Chapter1 Introduction

In this chapter, the background of this research, the basic of the repository-based agent framework and the overview of this research will be discussed. The organization of this thesis will be described at last.

1.1 Background

Agent Technology

In the past few years, agent technology has been proved very effective for the development of distributed and complicated systems and there have been more and more software projects involve multi-agent technology. To support the development of multi-agent system (MAS), a number of agent-oriented methodologies and/or frameworks have been proposed. Nevertheless, the difficulty of development is still the big problem of multi-agent system. At the programming unit level, a software agent is much more complicated than a software object according to the characters of agent such as autonomous, reactivity and social ability. That point means the developers who are familiar with the Object-Oriented Programming would find very hard to get used to the development of agent system in a short time. In the other hand, at the system level, multi-agent

system is usually applied into very complicated application domain which make it even difficult to develop.

Repository-based Agent Framework

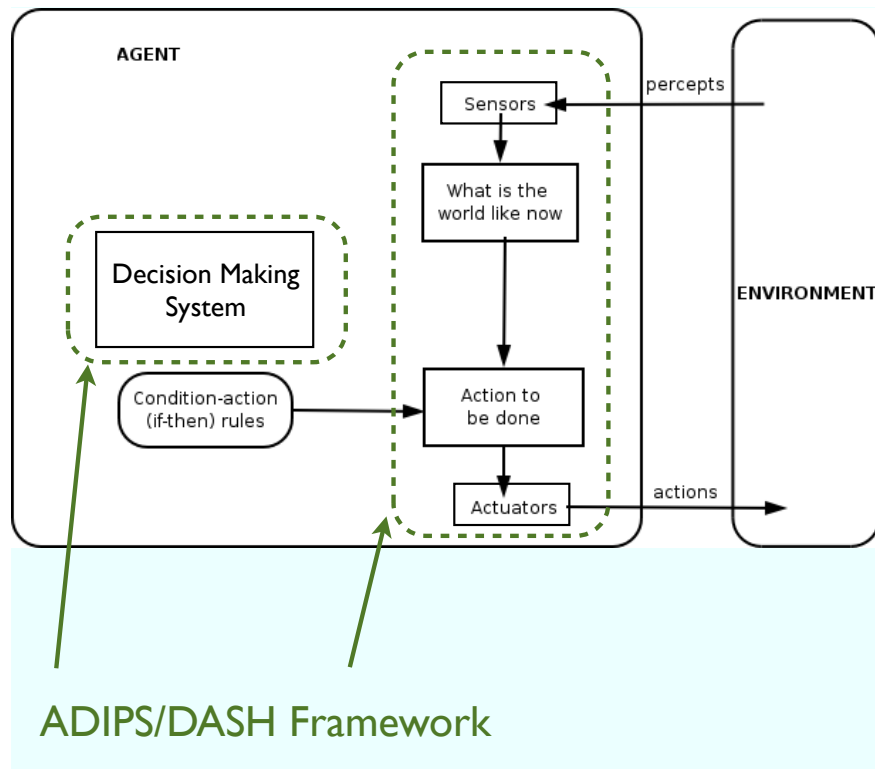


Figure 1.1: Basic Model of Agent

To improve that situation, repository-based agent framework (ADIPS/DASH)[1] has been proposed. *Figure 1.1* gives the basic model of agent and ADIPS/DASH framework. As a development framework designed just for agent, DASH just contains the essence functions of software agents. That keeps DASH simple and easy to learn and to use. Append to the simplicity, DASH also provides the repository mechanism to support the reuse of software components. Using the

repository mechanism practically, developers can reuse software agent and other software component effectively to avoid waste of time and resources spend on the re-implementation of the existing software components.

1.2 Case Study: Comparison of Multi-Agent System Development in DASH and other Framework

To compare DASH and the other agent frameworks, we took a small case study with DASH and JADE[3], a popular Java-based agent framework. A multi-agent web service composing system[2] has been taken as an example in the case study. The system could receive web service requirements from users and choose appropriate web services. By composing the chosen web services, the example system would provide the web services user required. Different kinds of agents are there in the system and as an example, let us consider the broker agents. A broker agent in the system is an agent that holds the knowledge about a specific web service. The information of the web service could be retrieved by querying to the broker agent. And the broker agent also could invoke the web service depend on the requirement received from other agents or users and send back the result of the invocation. The broker agents are the basic but important part of the web service composing system. The difficulty and the amount of coding work of the broker agents would decide the difficulty of the whole system implementation.

The code snippets of broker agents which are implemented in DASH and JADE are shown in *Figure 1.2*. As we can see the differences are pretty obvious. In JADE, as a Java-based agent framework, all the agents are implemented in Java,

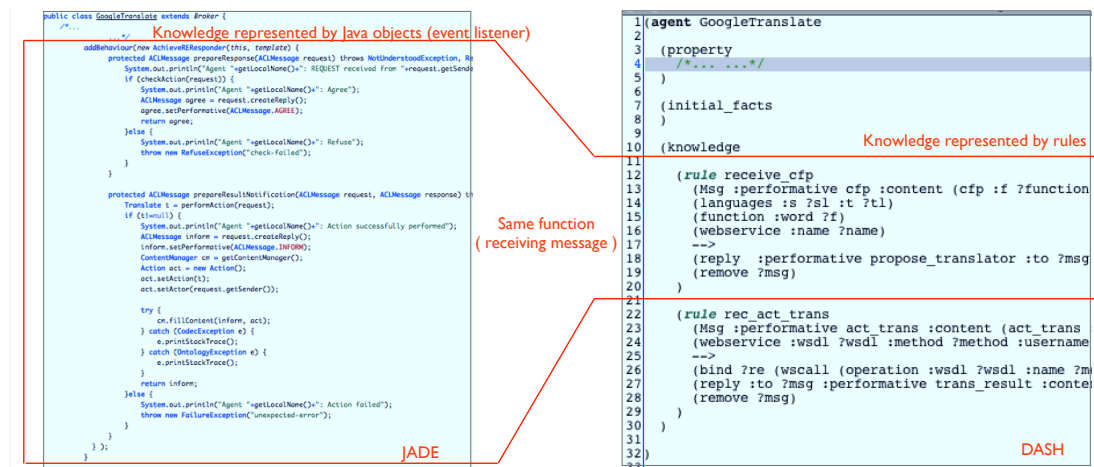


Figure 1.2: Comparison of JADE and DASH

including the knowledge that agents hold. Agents as Java objects would have rich functions, but remain complicated due to all the knowledge are represented by Java object and method such as event listener. In the other hand, DASH agents are implemented by DASH agent programming language which is designed for agent programming with the essence of a software agents. With DASH, agent could be implemented very simple and effectively. A reasoning engine is contained in every DASH agent by default to keep the knowledge of the agent could be represented by simple reasoning rules. In result to the same agent functions, a DASH agent could be more simple than a JADE agent in programming codes. As shown in *Figure 1.2*, for the same function of response to a cfp (call-for-propose) message, the codes of DASH agent is more simpler than the ones of JADE agent. The comparison of codes are surrounded by the red lines in the figure which we can clearly see. The whole comparisons of the status of development processes are shown in *Table 1.1*. Average lines of codes for one agent and the time spent in developing are compared. From the results we can obviously see that for the system like the one

mentioned, developing in DASH is much more easier and faster than in JADE.

	DASH	JADE
Average lines of codes for one agent	67 lines	426 lines
Time spent on developing	about 20 hours	about 80 hours

Table 1.1: Comparisons of the status of development processes

1.3 Multi-Agent System Development Support Mechanism of DASH

Not only the simple DASH agent programming language, but also to support multi-agent system development, DASH provide the repository to support software reuse. And the mechanism called "Rule Set" is introduced into DASH to improve the software reuse at rule level with the repository.

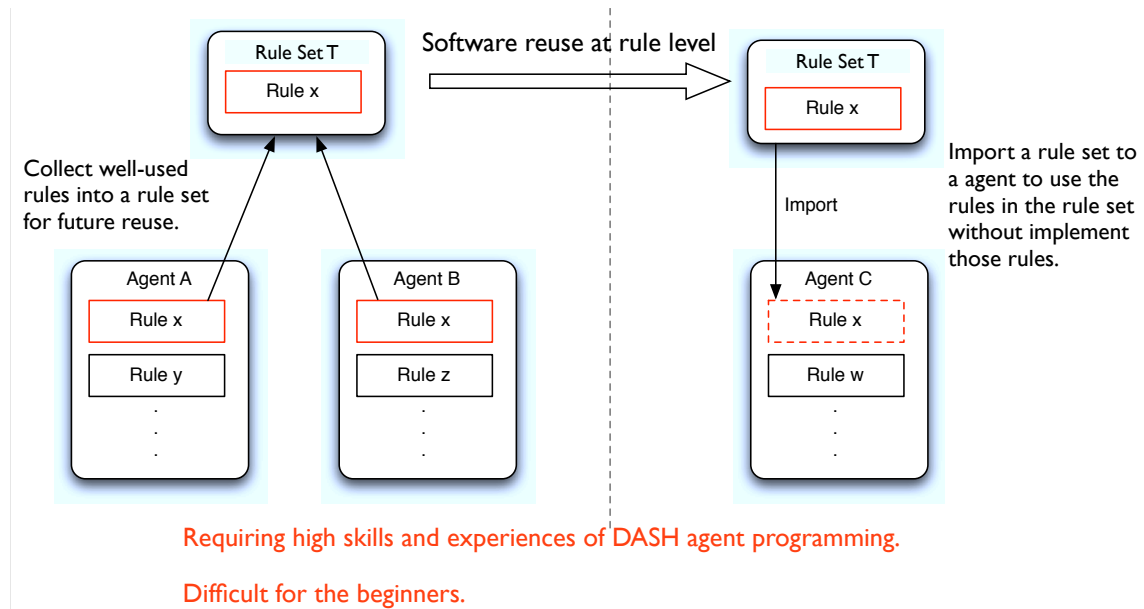


Figure 1.3: Software reuse at rule level in DASH

A rule set is a reusable rule library for agents while agent designers could collect

well-used rules into a rule set for future reuse. As the example shown in *Figure 1.3*, Agent A and Agent B contain a same Rule X, which could be collected into a Rule Set T. And for an other Agent C, developer can just import the Rule Set T to use the rules in it without implement them. With the rule level software reuse of the rule set mechanism, developers of agents could reuse the existing rule codes to develop new agents. But those rule level reuse require very high skills and experiences of DASH agent programming and difficult for the non-expert developers. That problem will be addressed to in this research.

1.4 Overview of This Research

Research Purpose

This research has been done to support the multi-agent system development in DASH.

Problem

The problem of DASH is the lack of easy-to-use, effective development methodology.

1. Difficult for non-expert developers
2. Heavy coding work due to the low level reusability.

Proposal

To solve that problem of DASH, the proposal of cooperation protocol design method for repository-based agent framework has been proposed and would be described in details in the following parts of this paper.

1.5 Organization of the Thesis

In Chapter 2, the related works about the existing multi-agent system development methodologies and the common multi-agent system development in DASH would be discussed. The proposed protocol design method would be described in Chapter 3. Chapter 4 contains the implementation, experiment and evaluation of this research. And the conclusion and future work would be discussed at least in Chapter 5, respectively.

Chapter2 Related Works

In this chapter, the related works about the existing multi-agent system development methodologies would be discussed. And the common multi-agent system development in DASH would be described.

2.1 Existing Multi-Agent System Development Methodologies

There are many different multi-agent system development methodologies have been proposed in order to support the development of agent systems. In the common sense of software engineering, multi-agent system development contains six different states in the whole lifecycle: Requirement Analysis, Design, Implementation, Testing, Deployment and Maintenance. As shown in *Figure 2.1*, supported development lifecycle coverages are different depend on different multi-agent methodologies. While GAIA[4], MaSE[7], Prometheus[8] and Troops[9] are supporting Requirement Analysis and Design phases, ADELFE[10], INGENIAS and PASSI[13] are supporting one more phase to Implementation. However, the other phases of Testing, Deployment and Maintenance are out of support by those methodologies. There are also researches[11][12][13] of comparing agent-oriented methodologies have been done. The detailed result of the comparisons from those

researches could be found in Appendix A.

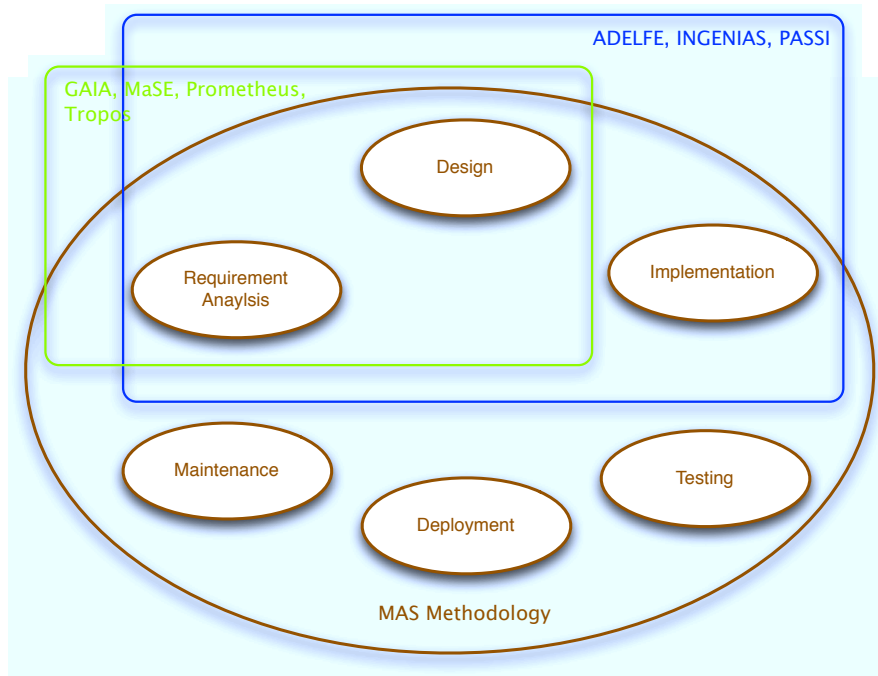



Figure 2.1: Comparison on development lifecycle coverages of multi-agent system methodologies

One of the problems of those methodologies is the limitation to the specific agent implementation frameworks. Some methodologies are requiring specific agent natures to adapt. The simple comparison of agent nature requirements are shown in *Figure 2.2*. The cells with red background are the requiring agent natures that DASH dose not support. Those requirements make it impossible to adapt those methodologies to DASH to support multi-agent system development. Another problem is the lack of Implementation phase coverage of the other methodologies. The multi-agent development in DASH could not be fully supported without the Implementation phase supporting methodology.

<i>Methodology</i>	GAIA ^[4]	ROADMAP ^[5]	MESSAGE ^[6]	MaSE ^[7]	Prometheus ^[8]	Tropus ^[9]	ADELFE ^[10]
<i>Requiring agent nature</i>	Heterogeneous	Uses knowledge schema	Heterogeneous	Not specified but possibly heterogeneous	BDI-like agents	BDI-like agents	Adaptive



 Requiring agent natures that DASH doesn't support.

 Cannot be adopted to DASH.

Figure 2.2: Comparison on agent nature requirements of multi-agent system methodologies

2.2 Common Multi-Agent System Development in DASH

To analyze the multi-agent system development in DASH, let us consider the common multi-agent system development behaviors of DASH. The development of software is usually started by developers with software ideas. And the final products of the development activities are the running software codes. The process from ideas to codes are often supported by software development methodologies. In common multi-agent system development situation of DASH, developers usually perform three different activities to archive the final agent codes. Those three activities are Knowledge Design, Protocol Design and Organization Design, as shown in *Figure 2.3*, respectively. Currently, the existing multi-agent development support function of DASH supports the software reuse at rule level and requires high skills and experiences of DASH agent programming which is difficult for the non-expert developers. And there is no available methodology to support the development. Address to those problems, this research focus on the frequently performed Protocol Design to provide support function for software reuse at design level for both expert and non-expert developers. The details of the proposal would

be described in the next chapter.

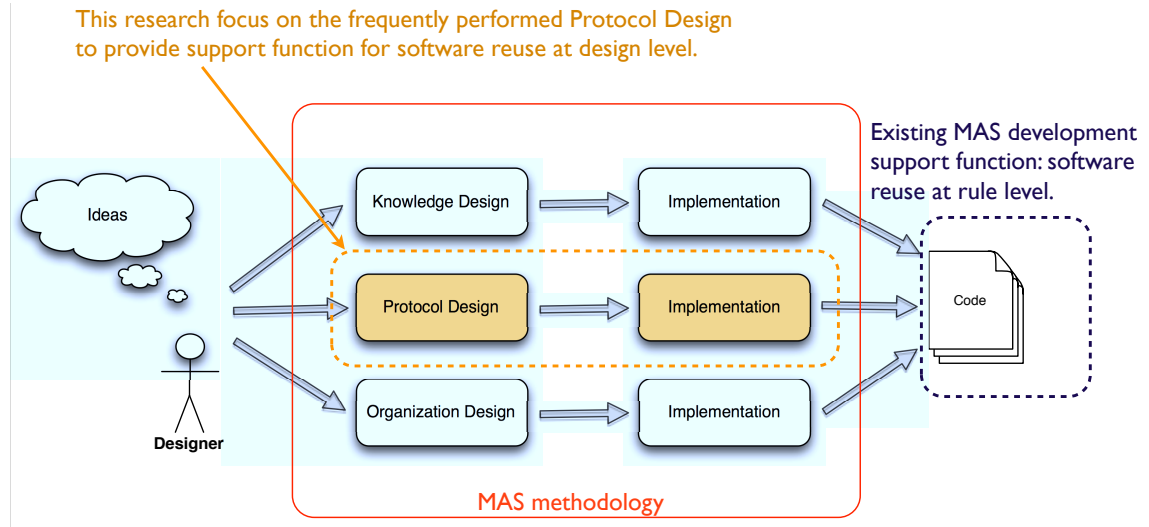


Figure 2.3: Multi-Agent System Development in DASH

2.3 Summary of Chapter 2

This chapter discussed the comparisons of existing multi-agent system development methodologies on the perspectives of software engineering lifecycle coverages and agent nature requirements. And the problems of adapting existing methodologies to DASH has been discussed. Further more, the common multi-agent system development in DASH has been described to introduce the research focus of this thesis.

Chapter3 Proposal of Cooperation Protocol Design Method for Repository-based Agent Framework

In this chapter, we would discuss our proposal of cooperation design method for repository-based agent framework in details. The proposed method would be described below by three different aspects which are related closely. *Figure 3.1* gives the overview of the proposal.

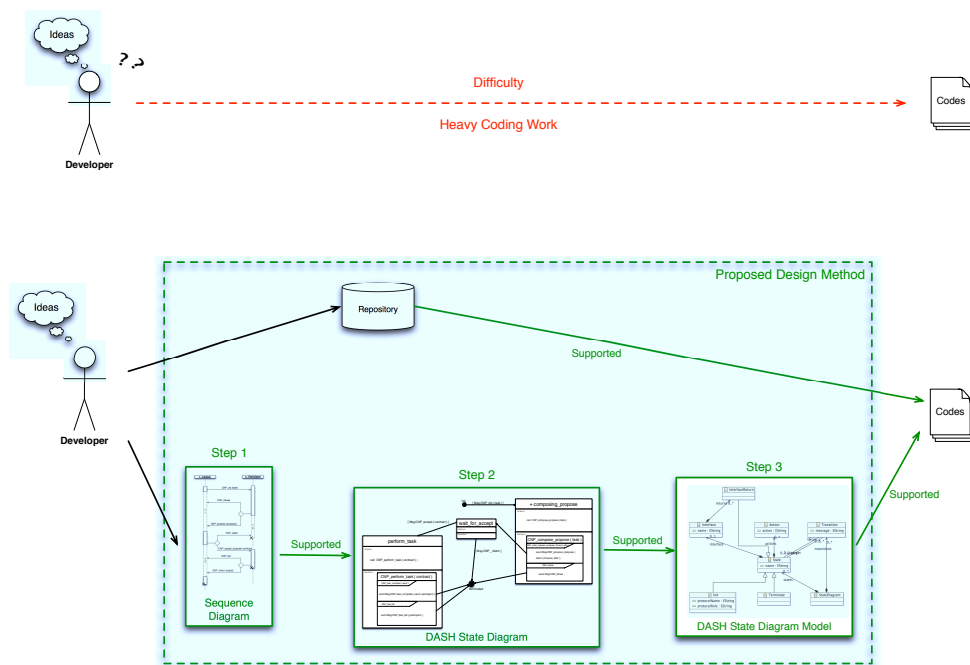


Figure 3.1: Overview of the proposal

3.1 Protocol Template Mechanism

For practical using the repository mechanism to improve the high level software reuse, we would like to introduce the protocol template mechanism. The protocol templates are the communication pattern abstract to well-used agent cooperation protocols. Developers of the protocol templates could make different protocol templates for different cooperation protocols and submit them to the repository to form a protocol library. The other multi-agent system developers who want to use the functions of the protocols just need to simply import the protocol templates they would like to use from the repository and implement the interfaces of the templates to use them, as shown in *Figure 3.2*, respectively.

Support for the expert developers.

3.2 Protocol design workflow for DASH

3.3 Protocol template design method

4.2 Protocol designer (expert developer) support function.

Support for the non-expert developers.

4.1 Protocol user (non-expert developer) support function.

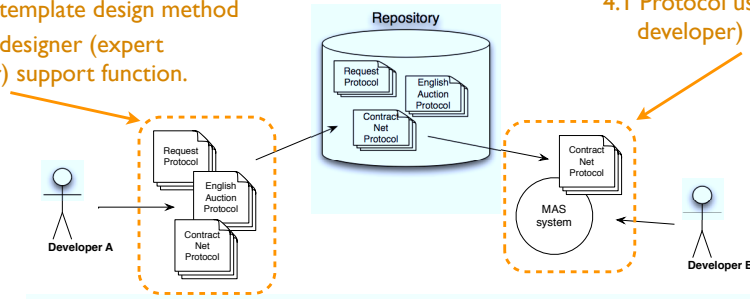


Figure 3.2: Protocol templates using repository

In *Figure 3.2*, developer A (expert developer) designed and implemented three different protocol templates and submitted them to the repository for future reuse. In the other side, developer B (non-expert developer) who hopes use the functions of a cooperation protocol just imported desired protocol template to his multi-agent system project without re-design and re-implementation of it. To support the expert developers, we would like to introduce protocol design workflow

for DASH, protocol template design method and supporting tools with protocol designer (expert developer) support functions which would be described separately in the following section 3.2, 3.3, and 4.2, respectively. While for the non-expert developers, we also prepared supporting functions for the protocol users which would be described in section 4.1 later.

3.2 Protocol Design Workflow for DASH

The protocol design workflow is the guideline and instructions for design protocol templates. The proposed workflow is described in *Figure 3.3*.

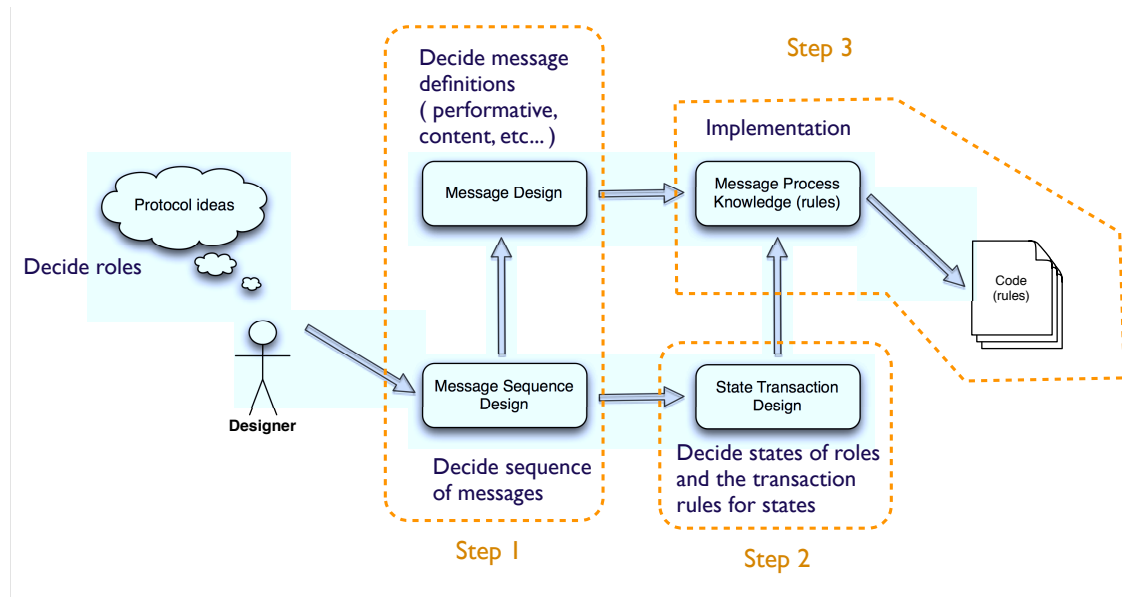


Figure 3.3: Protocol Design Workflow for DASH

As shown in *Figure 3.3*, the protocol template design activity would be started from the **Protocol Ideas** of protocol designer, where the roles of the protocol should be decided. After that, the sequence of the messages which would be used

in this protocol should be decided in the next **Message Sequence Design** phase. Once the sequence of the messages are defined, the developer should give the detail definitions of the messages such as the performative, content, etc... in the **Message Design** phase. And also the states of each roles of the protocol and the transition rules between states should be decided in the **State Transition Design** phase following. While all those design phase finished, the developer would implement the designed protocol template in the phase of **Message Process Knowledge (rule)** to get the final code (rules). The proposed workflow has been divided into three different step as shown in *Figure 3.2*. Phase **Message Sequence Design** and **Message Design** are contained by **Step 1**, while **State Transition Design** phase in **Step 2** with the rest in **Step 3**. We would propose the design method follow those design steps in the following section.

3.3 Protocol Template Design Method

In this section, we would like to describe the proposed protocol template design method for the steps of the proposed protocol design workflow which has been introduced in last section. The products and supporting tools for each step is listed in *Table 3.1*.

Step	Product	Tool Support
1. Role and Sequence Design	Sequence Diagram	General UML Design Tool
2. State Design	DASH State Diagram	Proposed Graphical Design Tool
3. Implementation	Code (*.rset files)	Code Generation Function

Table 3.1: Steps, products and supporting tools of proposed design method

The **Sequence Diagram** would be the product of **Role and Sequence Design**

step. And There are a number of general UML design tools to support the design of it. While for the **State Design** step, **DASH State Diagram** should be generated. We would introduce a graphical design tool which is designed for this purpose to support the design of this step in section 4.2. And for the **Implementation** step in which final codes are the product, the code generation function of the graphical design tool would definitely help the developers. We would like to introduce each step in details in the following subsections.

3.3.1 Step 1, Role and Sequence Design

The roles and message sequences of the protocol template would be decided by developer in this step using common UML sequence diagrams with specific notations for the template characters. An example of Contract Net Protocol template has been shown in *Figure 3.4*.

The specific nations for DASH protocol templates in the sequence diagram are surrounded with dashed lines in *Figure 3.4* which are called **interface**. The interface is just like the definition of functions, which are decided by the protocol designers in ways of the names and the parameters. The body of the interfaces would be left blank to the protocol users to finish the inner actions to meet their application domain specific requirements.

In the example of Contract Net Protocol template above, two different roles of Initiator and Participant communicate using seven different kind of type messages to archive the goals of the protocol. Five interfaces with parameters have been also defined by the protocol developer for the protocol users to archive the

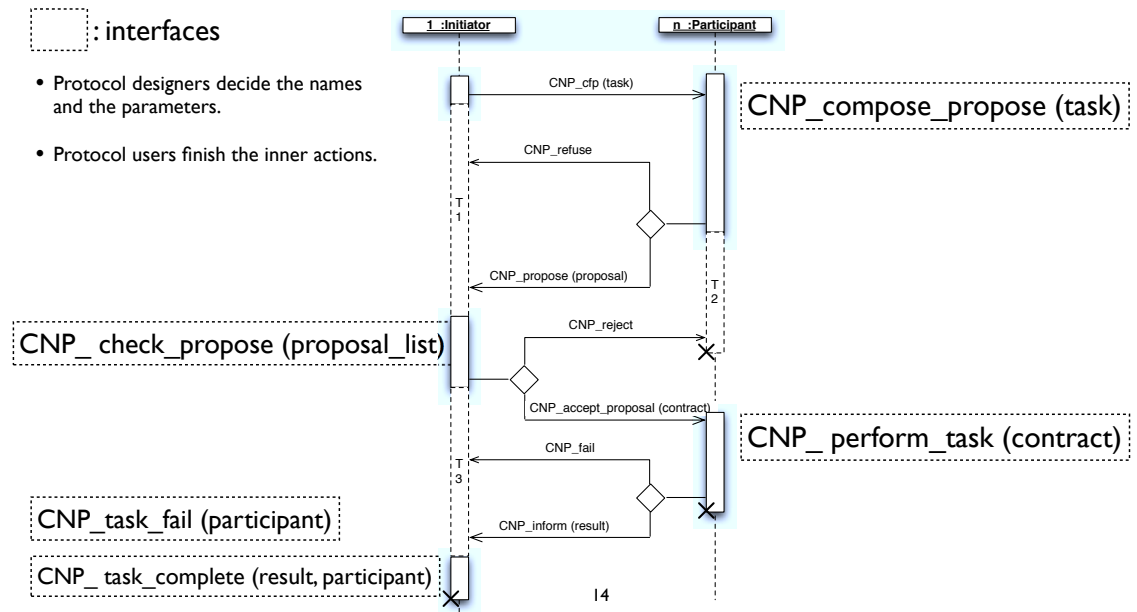


Figure 3.4: Sequence Diagram

application domain specific requirements. Once the sequence diagram is defined, the development could entry the next step of State Design which we will describe in next subsection.

3.3.2 Step 2, State Design

For each role of the protocol template, the DASH State Diagram should be designed in this step by the designer depend on the sequence diagram which has been designed in the last step. The DASH State Diagram is formed by six different basic components which are shown in *Figure 3.5*.

The DASH State Diagram would be always started from the **Initiation** state and ended to the **Termination** state. The else states of the roles would be described by the **State** part with the specific state names. The “+” in front of the state name

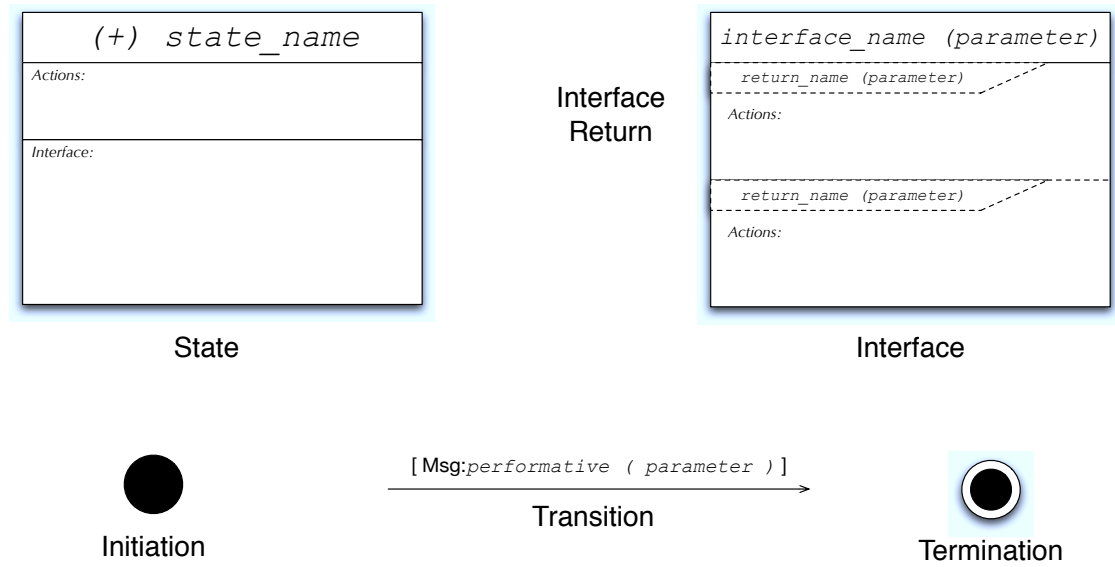


Figure 3.5: DASH State Diagram Component

decide if a new thread would be created for a particular state. And the Actions part of the state describe the actual actions would be performed in the state. The state would contain one optional **Interface** which must be defined with a name and none or more parameters. The interface would be always invoked with the parameters after all the actions of the state has been performed by the system. One or more **Interface Return** components would be contained by an interface. The interface return components must be defined by interface return names and optional parameters and actions. Finally, all the states must be connected by the **Transition** with the optional message labeled.

With the basic components which have been described above, developers could easily compose them to construct complex DASH State Diagram. One example of the Role Participant of Contract Net Protocol template has been shown in *Figure 3.6*.

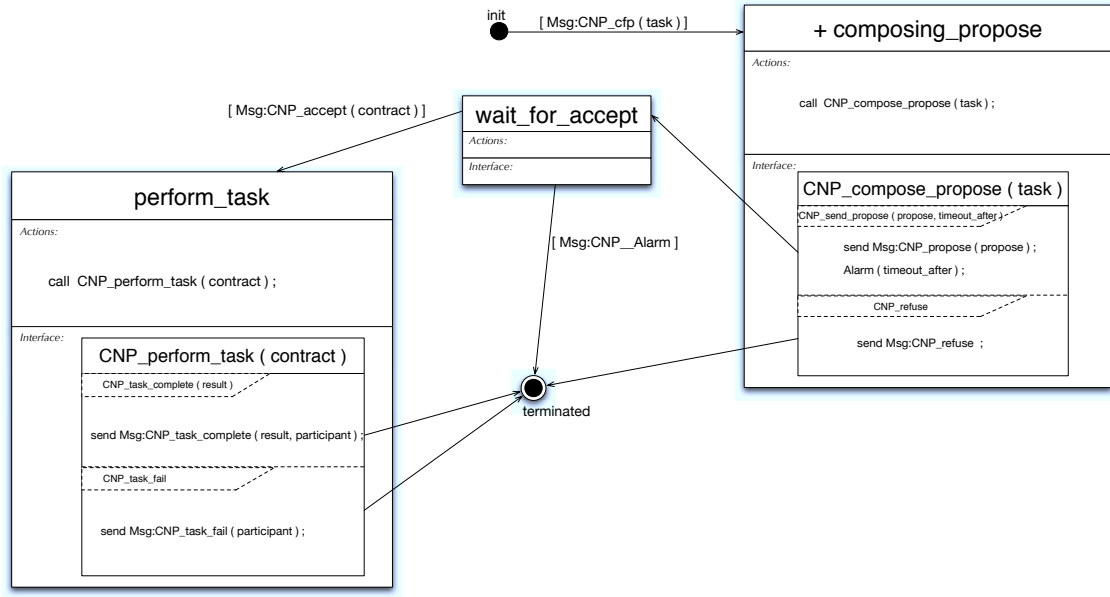


Figure 3.6: DASH State Diagram of Role Participant of Contract Net Protocol Template

In the example there are three different states besides the initiation and the termination states. The initiation state would transfer to the *composing_propose* state in a new created thread on receiving a message with the performative of *CNP_cfp* and parameter of *task*. The interface *CNP_compose_propose* would be invoked in the state *composing_propose* to give the protocol template users chance to decide whether propose or not using interface return *CNP_send_propose* or *CNP_refuse*. If user chooses to refuse, the message with performative of *CNP_refuse* would be sent and the thread turns into termination state. While the user chooses to propose, the message with performative of *CNP_propose* would be sent and an alarm would be set and the thread turns into the state of *waiting_for_accept*. If the alarm goes time out without receiving the accepted message, the thread goes to terminate. On receiving message with performative of *CNP_accept*, the thread transfers to the state of *perform_task* to perform the

contracted tasks by invoking the interface of *CNP_perform_task*. Finally, after performing the tasks, the thread turns into termination.

As the example we have been explained above, the developers could use the components to construct any kind of DASH State Diagram to describe the state transitions of the roles of the protocol templates. Once this step has been finished, the developer could implement the designed transitions in the next step of Implementation which would be described in next subsection.

3.3.3 Step 3, Implementation

In this step developers should convert the designed DASH State Diagram into DASH rule set codes. To archive that goal, we would like to introduce a meta model of the DASH State Diagram so that the implementation activity could be described by the mapping between models and codes. The meta model of the DASH State Diagram has been shown in *Figure 3.7*.

According to the meta model, the **StateDiagram** class contains the **State** class and the **Transition** class as components. And the Transition class has State class type properties named “source” and “target” which points to the source state and the target state of the transition. **Interface** class with **InterfaceReturn** class in it with a component could be contained by the State class, which could also contain **Action** class as components.

Using the proposed meta model, the DASH State Diagram could be modeled and mapping to the DASH rule set codes. The mapping keeps one Transition object

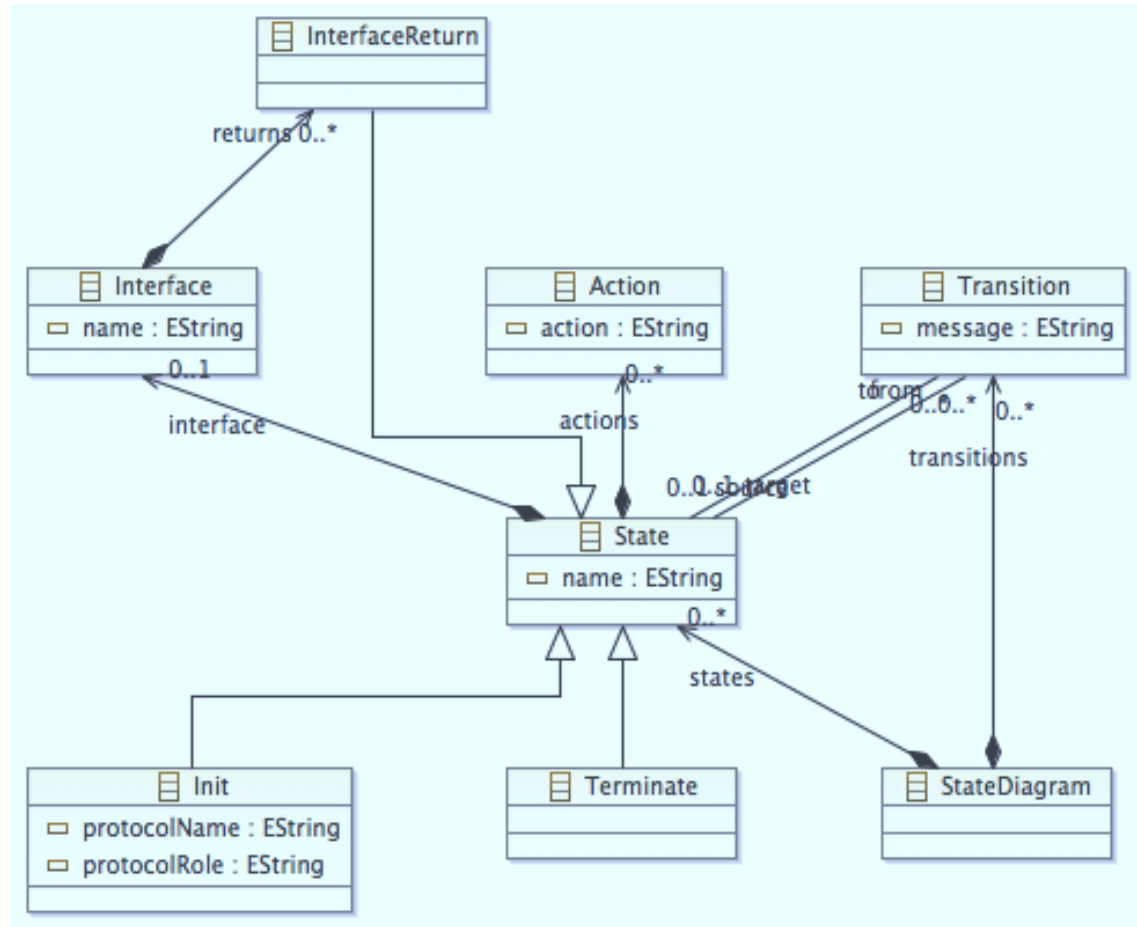


Figure 3.7: Meta Model of the DASH State Diagram

to one rule which has the conditions from the message property of the Transition object and the name of State object of source property. The actions of the rule are constructed by the Action properties of the target State object of the Transition object. And example of the State Diagram-model-code mapping has been shown in *Figure 3.8* and *Figure 3.9*.

The diagram-model mapping has been shown in *Figure 3.8* with the example of the part of Contract Net Protocol template. The transition from initiation to the *composing_propose* state has been mapped to the transition object with the source

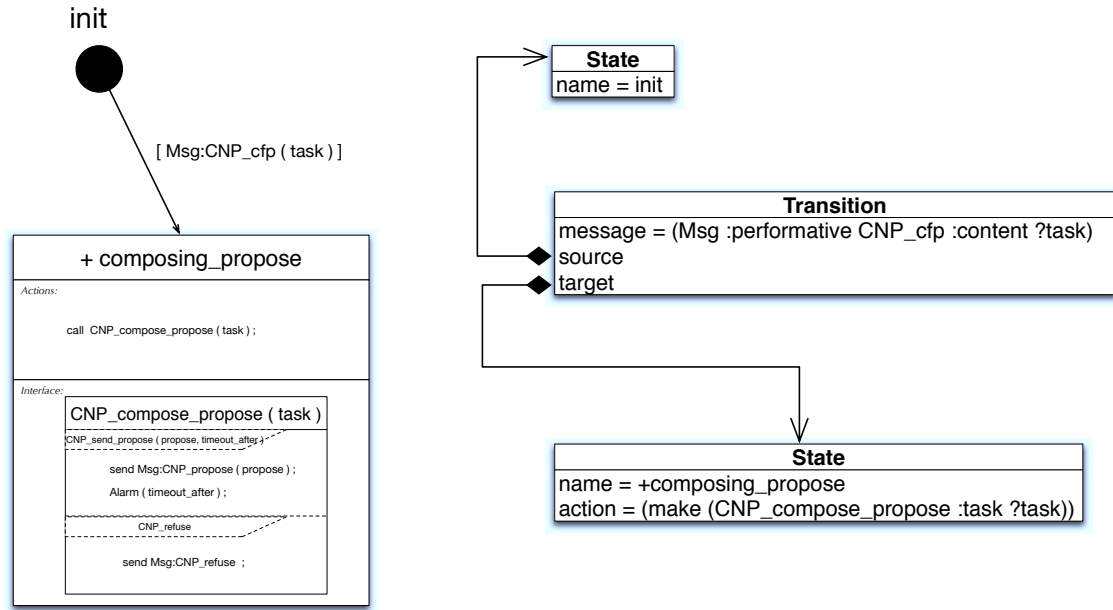


Figure 3.8: DASH State Diagram to Model Mapping

property which pointed to the state object with the name property of “init” and the target property which pointed to the state object with the name property of “composing_propose”.

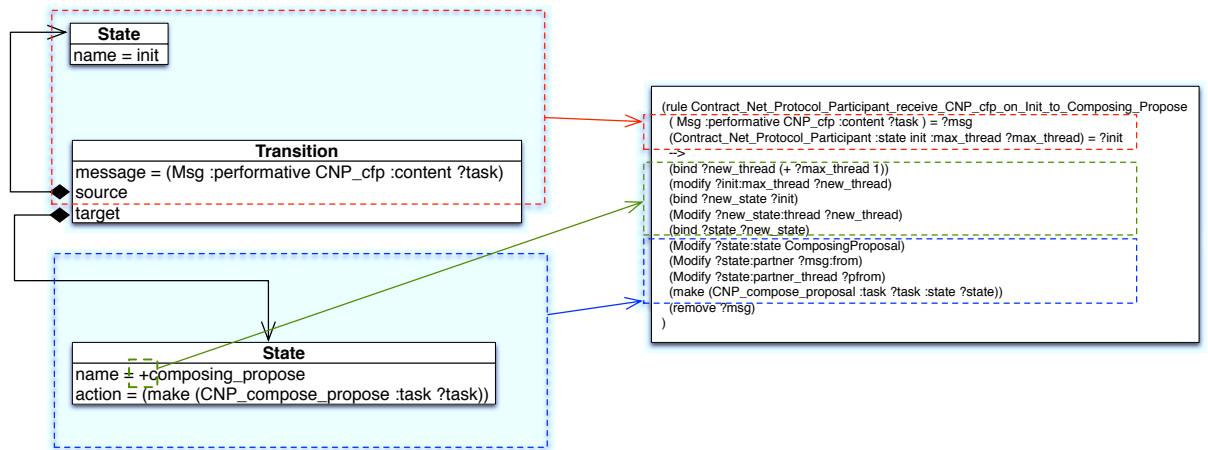


Figure 3.9: DASH State Diagram Model to DASH Rule Set Code Mapping

The model-code mapping has been shown in *Figure 3.9* with the same example.

The conditions of the rule would be constructed by the transition object and the source state of it. And the action part of the rule would be constructed by the action properties of the target state of the transition. The “+” notation in front of the name of the state would construct the codes of creating new thread.

With the proposed diagram-model-code mapping in this step, the developers could easily implement the designed DASH State Diagram into reusable DASH rule set codes. Further more, we also provide the graphical design tool for the diagram design with the function of code generation to support the development of protocol templates which would be introduced in the next chapter.

3.4 Summary of Chapter 3

In this chapter, the proposal of cooperation protocol design method for repository-based agent framework has been described in the perspectives of protocol template mechanism, protocol design workflow and protocol template design method. The DASH State Diagram and the meta model of the diagram has been proposed to help developers to design and implement the protocol template. The supporting tools would be introduced in next chapter.

Chapter4 Implementation, Experiment and Evaluation

The implementations of the supporting functions of the proposed design method would be described in this chapter. Further more, the experiment of the reduction of the coding work and the evaluation of it would also be introduced.

4.1 Implementation 1: Protocol user (non-expert developer) support function

Purpose

This function would be used to support the non-expert developers who use the protocol template in their works.

Method

To support the non-expert developers, the automatic code generation functions for the implementation of protocol template interfaces should be appended to the development environment of DASH which is called IDEA.

Running Environment

The supporting functions should be running under the environments described below.

- Operating System: Windows, Mac OS, Linux
- Java Virtual Machine: Version 1.5 or higher
- DASH: Version 1.9.7h
- IDEA: Version 2.0 beta

Result

The example result using the Role Participant of the Contract Net Protocol template has been shown in *Figure 4.1*.

To use the non-export developer supporting function, the developer could first choose a protocol template to import from the “PROTOCOL” menu of the menu bar. Then the codes which are necessary for using the chosen protocol template are automatically generated by the function with the user instructions in the form of comments. The only thing that developers should do is to complete the rules with their own application simian specific actions. In the example above, 56 lines of codes in 3 rules with 30 lines of instructions are generated by the support function.

With the support function described above, developers could easily use the protocol templates in their own multi-agent system without heavy and confusing

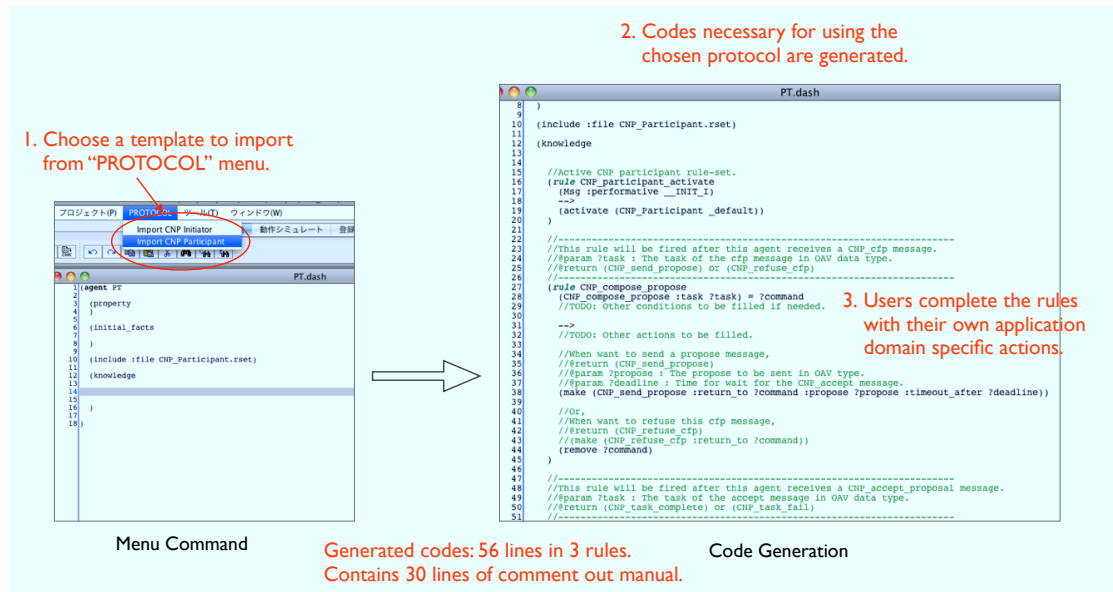


Figure 4.1: Protocol user (non-expert developer) support function

coding work. The problem of difficulty for the non-expert developers has been solved.

4.2 Implementation 2: Protocol designer (expert developer) support function

Purpose

This function would be used to support the expert developers who design protocol templates for future reuse.

Method

- To support the Diagram Design step of the proposed design method which has been described in last chapter, a graphical design tool has been created.
- The automatic code generation function has been appended to the graphical design tool to support the Implementation step of the proposed design method.

Running Environment

The supporting functions should be running under the environments described below.

- Operating System: Windows, Mac OS, Linux
- Java Virtual Machine: Version 1.5 or higher
- DASH: Version 1.9.7h
- IDEA: Version 2.0 beta

Result

The snap shot of the running graphical design tool has been shown in *Figure 4.2* with the DASH State Diagram of Role Participant of the Contract Net Protocol template as an example.

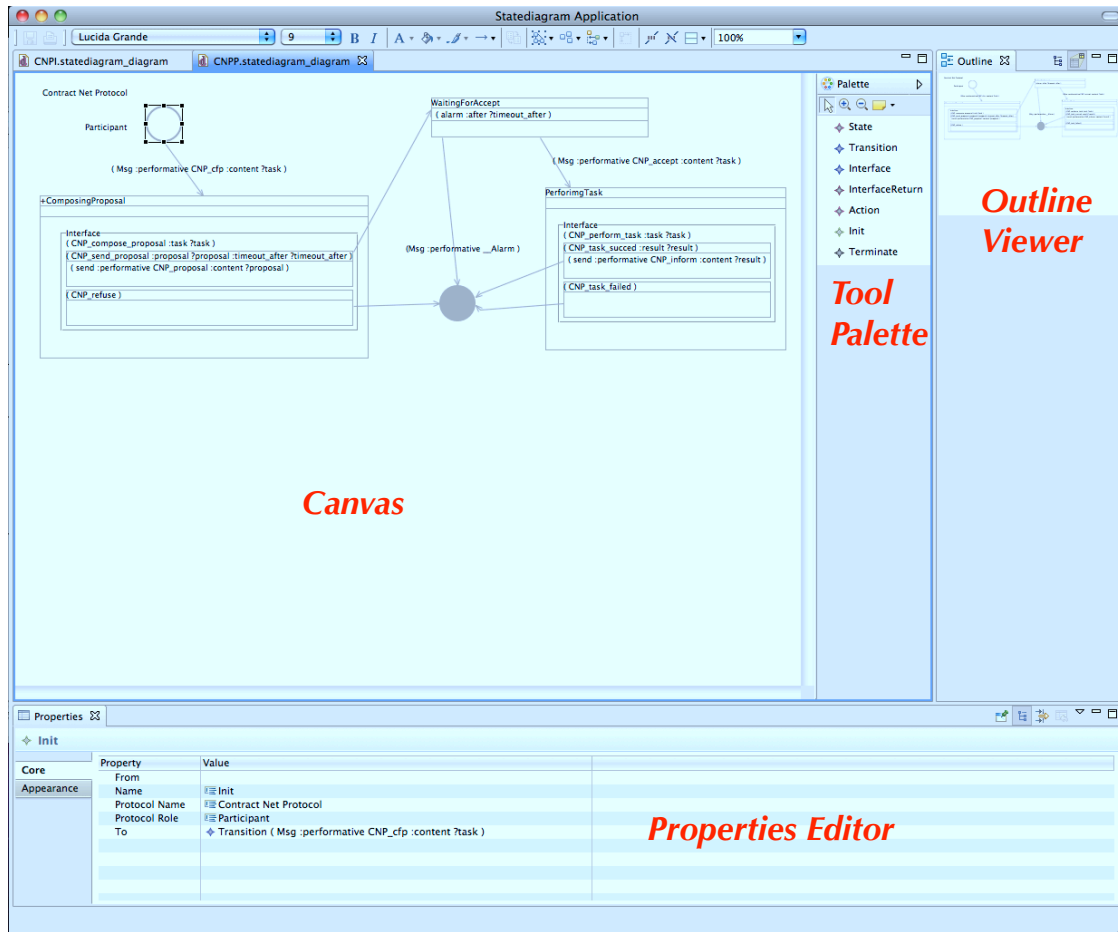


Figure 4.2: User interface of the graphical design tool for the DASH State Diagram design

The tabbed **Canvas** has been placed in the main place of the tool in which user could design the diagram by choose different component from the **Tool Palette** besides the canvas. At the bottom of the tool the **Properties Editor** could be used to easily edit the properties of the components. Finally, the right side **Outline Viewer** could give the developer a full perspective of the whole diagram. The detailed images of Tool Palette and Properties Editor could be seen in *Figure 4.3*.

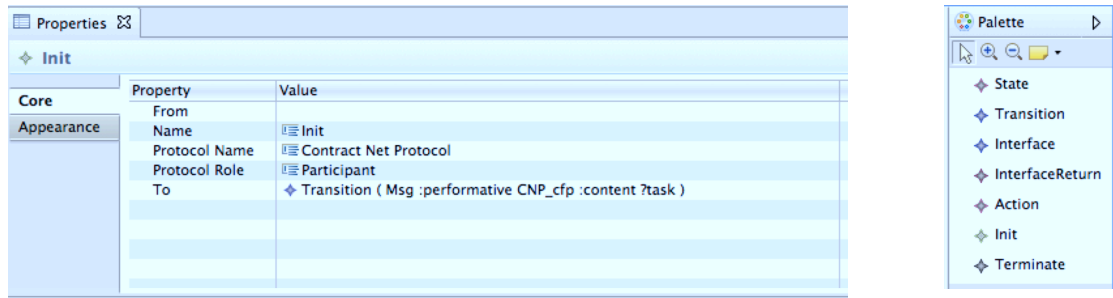

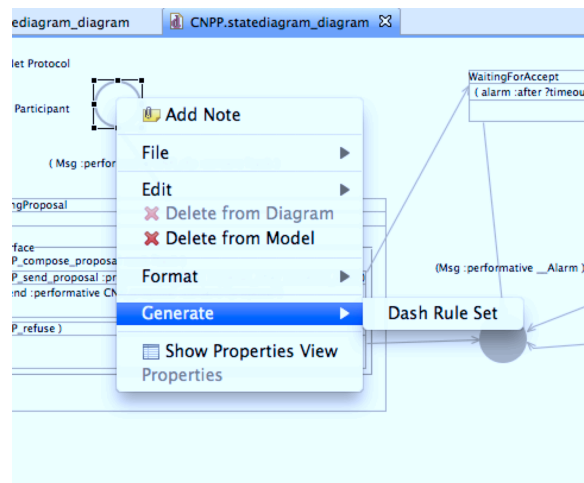


Figure 4.3: Detailed images of Tool Palette and Properties Editor

Once the design of the DASH State Diagram has been finished, the developers could use the “Generate  Dash Rule Set” menu from the popup menu of the init component of the diagram to generate the DASH rule set codes. The menu has been shown in *Figure 4.4*

Figure 4.4: “Generate  Dash Rule Set” menu

The example of output DASH rule set code file could be seen in *Figure 4.5*. And the whole content of the file could be found in Appendix B.

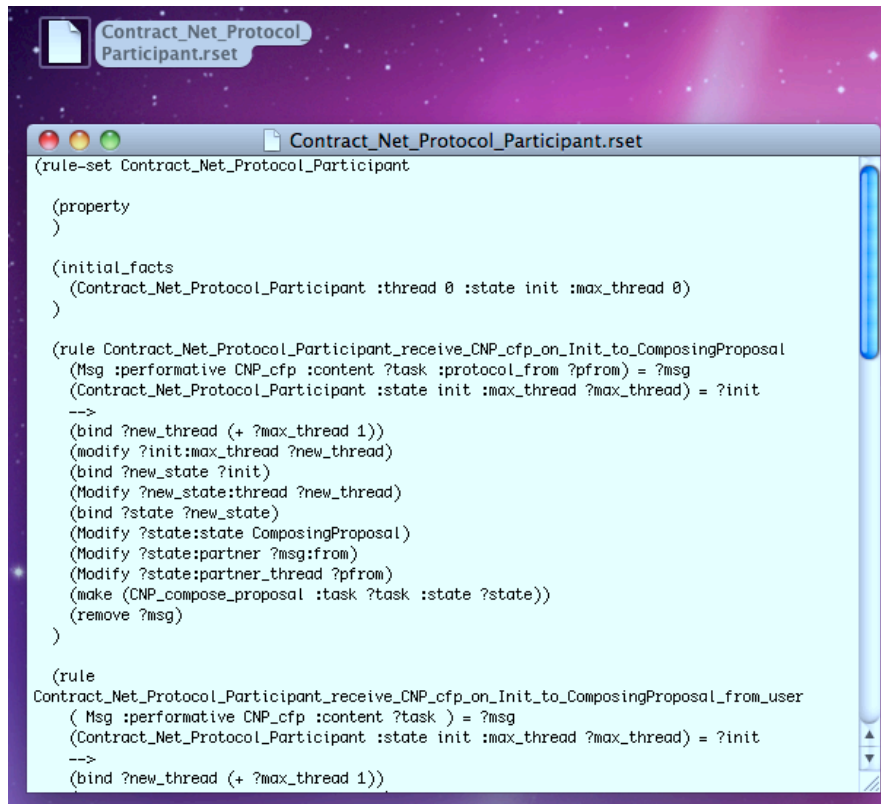


Figure 4.5: DASH rule set code file

With the graphical design tool and the code generation function which have been described in this subsection, the developers who design the protocol templates could easily design the DASH State Diagram and implement the DASH rule set codes without coding works. In next section, an experiment would be performed to verify the validity of the generated codes and to evaluate the effect of the reduction of coding work.

4.3 Experiment : Reduction of coding work

Purpose

This experiment would be used to verify the validity of the generated codes and to evaluate the effect of the reduction of coding work of the proposed method and supporting tools.

Method

- The Contract Net Protocol template had be designed and implemented using the proposed design method and supporting tool.
- The multi-agent system of MicroGrid Controlling[14][15] which had been designed and implemented before had been redesign and reimplemented using the generated Contract Net Protocol template.

Running Environment

The experiment was running under the environments described below.

- Operating System: Mac OS
- Java Virtual Machine: Version 1.6
- DASH: Version 1.9.7h
- IDEA: Version 2.0 beta

Result

Contract Net Protocol (CNP) is a well known task sharing protocol that is used for task allocation in multi-agent systems and consists of a collection of nodes or software agents that form the contract net. Each node on the network can, at different times or for different tasks be a manager or a contractor.[16] *Figure 4.6* gives a abstract of the Contract Net Protocol.

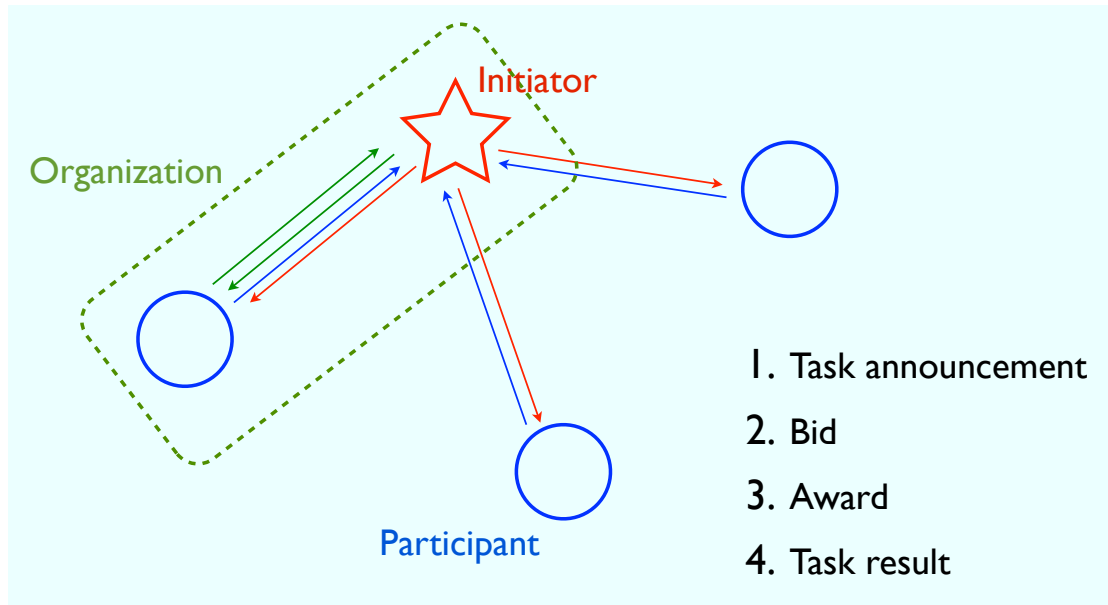


Figure 4.6: Contract Net Protocol

The DASH State Diagrams which has been designed using the proposed design method and DASH Rule Set files which has been generated by the supporting tool has shown in *Figure 4.7*.

The new implemented MicroGrid Control System using the generated protocol templates has been running well as designed, which verified the validity of the

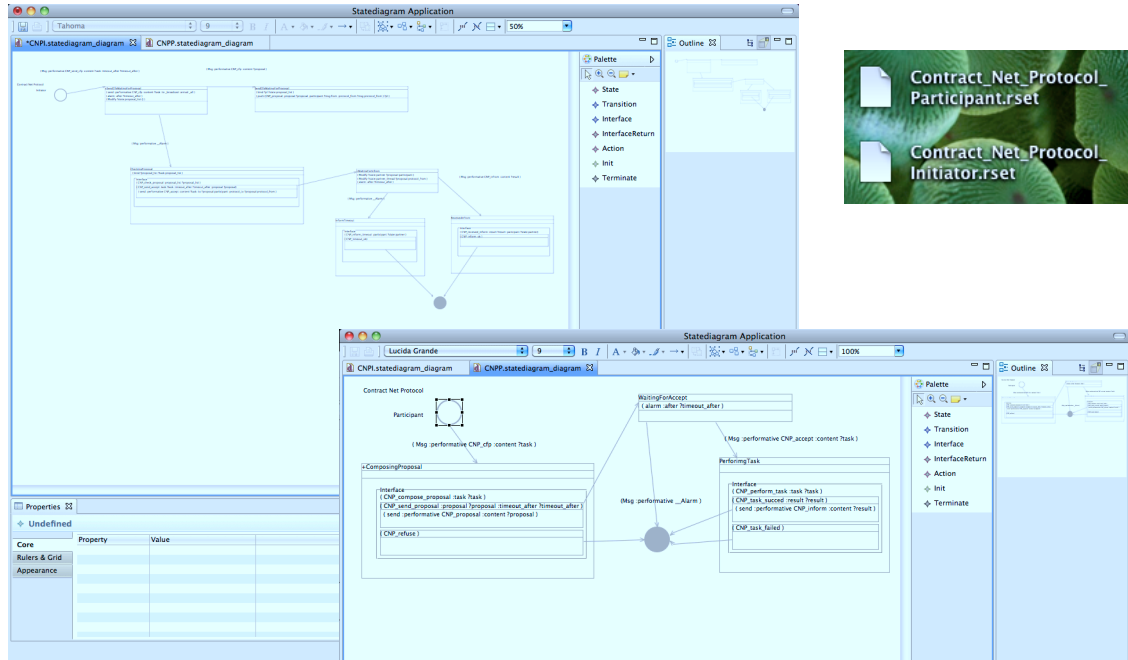


Figure 4.7: DASH State Diagrams and DASH Rule Set files

generated codes. The snap of running system has been shown in *Figure 4.8*.

Comparing to the old implementation which were all written by developers, the experiment implementation using the proposed design method and supporting tool which had only small parts need to be written by developers. The example of code comparisons has been shown in *Figure 4.9*.

Evaluation

To evaluate the coding work reducing effect of the proposed design method, the comparison of lines of code between the old implementation and proposed design method has been performed. The result of the comparison could be seen in *Table 4.1*.

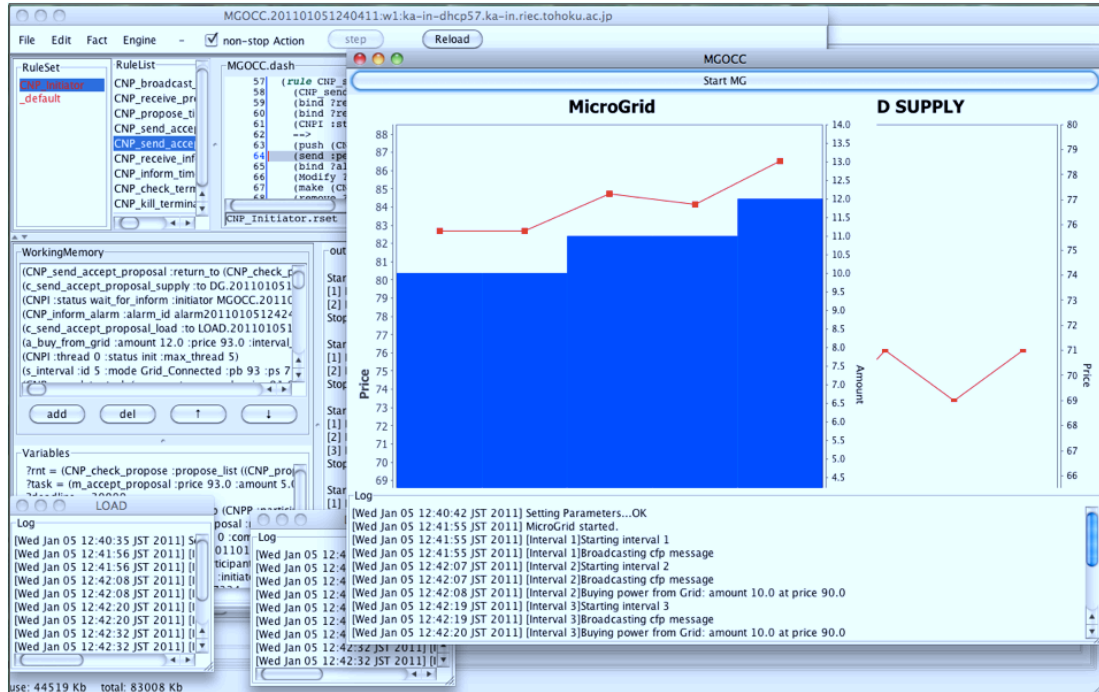


Figure 4.8: MicroGrid Control System using the generated protocol templates

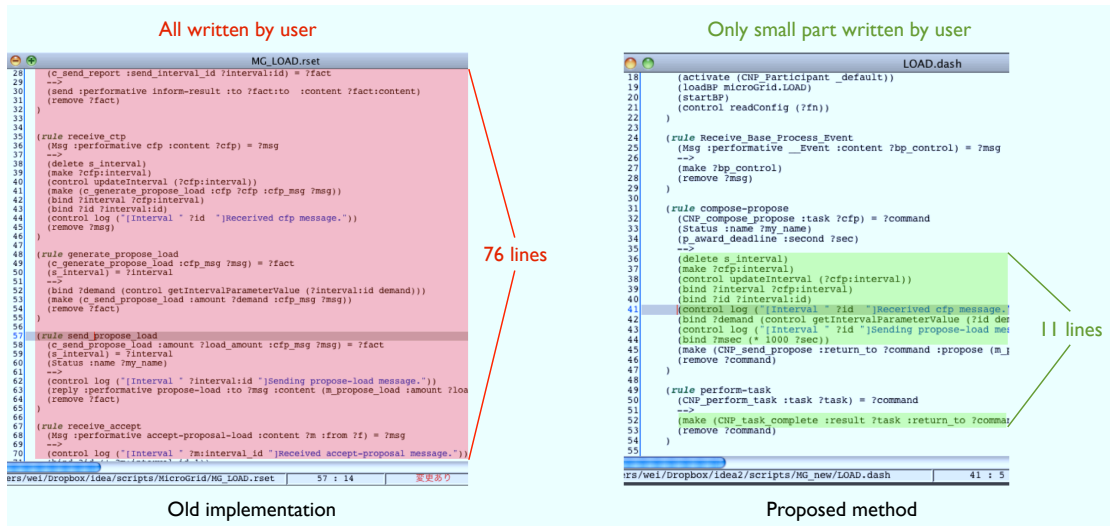


Figure 4.9: Code comparisons between old implementations and proposed method

	MGOCC Agent	LOAD Agent	DG Agent
Lines of codes of old implementation	298 lines	114 lines	136 lines
Lines of codes of proposed method	129 lines	58 lines	62 lines
Coding work reduction	56.7%	49.1%	54.4%

Table 4.1: The result of the comparison of lines of code between the old implementation and proposed design method

According to the result table, about half coding work has been reduced using the proposed design method comparing to the old implementation. The problem of heavy coding work had been solved by the proposed design method.

4.4 Summary of Chapter 4

The implementations of the supporting functions for the proposed design method had been introduced in this chapter. The experiment of reduction of coding work showed the validity of the generated codes and to evaluate the effect of the reduction of coding work of the proposed method and supporting tools. The problems of difficulty for the non-expert developer and heavy coding work had been solved by the proposed design method.

Chapter5 Conclusion

5.1 Conclusion

Research Purpose

This research has been done to support the multi-agent system development in DASH.

Problem

The problem of DASH is the lack of easy-to-use, effective development methodology.

1. Difficult for non-expert developers
2. Heavy coding work due to the low level reusability.

Proposal

To solve that problem of DASH, the proposal of cooperation protocol design method for repository-based agent framework has been proposed.

Result

1. The supporting functions and the proposed design method successfully reduced the difficulty of design and implementation of protocol templates for repository-based multi-agent framework.
2. The proposed design method effectively reduced coding work by practical using design level software reuse.

Acknowledgements

I would like to be very grateful for the significant and helpful advice and support of Professor Tetsuo KINOSHITA of Tohoku University, during my master course. In every stage of this research, precious advice and comments given have improved.

I would like to deeply thank the judges Professor Michitaka KAMEYAMA and Professor Ayumi SHINOHARA for their significant opinions and comments to this thesis.

The dedicated comments and guidance given by Associate Professor Gen KITAGATA of Flexible Information System Research Center in Research Institute of Electrical Communication of Tohoku University have greatly improved and clarified this work. Here, I would like to express my deeply thanks to Associate Professor Gen Kitagata for his significant opinions and comments.

Likewise, I would like to deeply thank the staffs of the KINOSHITA Lab who provided me with assistance during the development of the ideas in this thesis, and for helpful comments on the text.

At last, deeply thanks to the students at the KINOSHITA Lab who contributed to my work in so many ways.

References

- [1] T. Uchiya, T. Katoh, T. Suganuma, T. Kinoshita, N. Shiratori. An architecture of agent repository for adaptive multiagent system. ICOIN 2002, LNCS 2344, pp. 802–813, 2002.
- [2] R. Miura, W. Wei, A. Takahashi, T. Kinoshita. Using Language Grid services based on Active Information Resource. IEICE technical report. Artificial intelligence and knowledge-based processing 108(441), 25-30, 2009-02-20.
- [3] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE: a FIPA2000 compliant agent development environment. In Proceedings of the fifth international conference on Autonomous agents (AGENTS '01). ACM, New York, NY, USA, 216-217. 2001.
- [4] M. Wooldridge, N.R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3), 2000.
- [5] Juan, T., Sterling, L.: The ROADMAP Meta-Model for Intelligent Adaptive Multi-Agent Systems in Open Environments. In: Giorgini, P., Mller, J.P., Odell, J.J. (eds.) *Agent-Oriented Software Engineering IV*. LNCS, vol. 2935, Springer, Heidelberg (2004)

- [6] Giovanni Caire, Francisco Leal, Paulo Chainho, Richard Evans, Francisco Garijo, Jorge Gomez, Juan Pavon, Paul Kearney, Jamie Stark, and Philippe Massonet. Agent oriented analysis using MESSAGE/UML. In Michael Wooldridge, Paolo Ciancarini, and Gerhard Weiss, editors, Second International Workshop on Agent-Oriented Software Engineering (AOSE- 2001), pages 101–108, 2001.
- [7] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.
- [8] Michael Winikoff and Lin Padgham. The prometheus methodology. In Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, chapter 11. Kluwer Academic Publishing (New York), 2004.
- [9] Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. Troops: An agent-oriented software development methodology. Technical Report DIT-02-0015, University of Trento, Department of Information and Communication Technology, 2002.
- [10] Bernon, C., Gleizes, M.-P., Peyruqueou, S., Picard, G.: ADELFE: A methodology for adaptive multi-agent systems engineering. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) *ESAW 2002. LNCS*, vol. 2577, pp. 156–169. Springer, Heidelberg (2003)
- [11] A. Ghose, G. Governatori, and R. Sadananda (Eds.): *Towards Method Engineering for Multi-Agent Systems: A Validation of a Generic MAS*

Metamodel, PRIMA 2007, LNAI 5044, pp. 255–267, 2009.

- [12] B. Apolloni et al. (Eds.): A Comparison of Three Agent-Oriented Software Development Methodologies: ROADMAP, Prometheus, and MaSE, 2007/WIRN 2007, Part III, LNAI 4694, pp. 909–916, 2007.
- [13] P. Giorgini et al. (Eds.): Comparing Agent-Oriented Methodologies, AOIS 2003, LNAI 3030, pp. 78–93, 2004.
- [14] Hak-Man Kim and Tetsuo Kinoshita. A New Challenge of Microgrid Operation. Communications in Computer and Information Science, 2010, Volume 78, 250-260.
- [15] Hak-Man Kim, Wenpeng Wei and Tetsuo Kinoshita. A New Modified CNP for Autonomous Microgrid Operation Based on Multiagent System. Journal of Electrical Engineering and Technology Vol. 6, No. 1, pp. 139–146, 2011
- [16] http://en.wikipedia.org/wiki/Contract_Net_Protocol

Publication

1. Hak-Man Kim, Wenpeng Wei and Tetsuo Kinoshita, "A New Modified CNP for Autonomous Microgrid Operation Based on Multiagent System," Journal of Electrical Engineering and Technology Vol. 6, No. 1, pp. 139 146, 2011
2. Wenpeng Wei, Tetsuo Kinoshita, "A Survey on Agent Design Support Functions using Repository based Agent Framework," First International Joint Agent Workshop & Symposium 2010 (iJAWS2010), X-4, 2010,10
3. Wenpeng Wei, Ryosuke Miura, Tetsuo Kinoshita, "Discovering And Composing Web Services Using Repository-based Agent Framework," 2010 Tohoku-Section Joint Convention Record of Institutes of Electrical and Information Engineers, Japan, pp.38, 2010,8
4. Wenpeng Wei, Ryosuke Miura, Toru Abe, Tetsuo Kinoshita, "A Multi-Agent Architecture for Discovering and Composing Semantic Web Services," 合同エージェントワークショップ&シンポジウム JAWS2009 講演論文集 , pp.572-574, 2009.10
5. Wenpeng Wei, Ryusuke Miura, Toru Abe, Tetsuo Kinoshita, "Discovering And Composing Semantic Web Services Using Agents," 第8回科学技術フォーラム FIT2009 講演論文集 F-023 , 電子情報通信学会/情報処理学会 , 2009.9

6. 水内翔太, 魏 文鵬, 三浦良介, 阿部 亨, 木下哲男, "Web サービスの能動化と連携の支援," 情報処理学会 創立 50 周年記念大会 (第 72 回) 全国大会講演論文集 , 2V-2, 2010.3
7. 水内翔太, 魏 文鵬, 三浦良介, 阿部 亨, 木下哲男, "Web サービスを対象とした能動的情報資源の設計," 第 8 回科学技術フォーラム FIT2009 講演論文集 F-031 , 電子情報通信学会/情報処理学会 , 2009.9
8. Ryoseke Miura, Wenpeng Wei, Akiko Takahashi, Tetsuo Kinoshita, "Using Language Grid services based on Active Information Resource," IEICE technical report. Artificial intelligence and knowledge-based processing 108(441), 25-30, 2009-02-20
9. RADESCU George Serban , OGAWA Satoshi , WEI Wenpeng , KITAGATA Gen , TAKEDA Atsushi , SHIRATORI Norio, "A Flexible Model of Access Control based on Social Relations," IPSJ SIG Notes 2008(54), 105-111, 2008-06-12
10. RADESCU George Serban , OGAWA Satoshi , WEI Wenpeng , KITAGATA Gen , TAKEDA Atsushi , SHIRATORI Norio, "Access Control in Cooperative Environments utilizing Social Systems (ACCESS)," IEICE technical report. Information networks 107(222), 143-148, 2007-09-13
11. OGAWA Satoshi , GEORGE RADESCU Serban , WEI Wenpeng , KITAGATA Gen , TAKEDA Atsushi , SHIRATORI Norio, "A Flexible and Secure Access Control Scheme using Social Behavior in the Real World," IEICE technical report. Information networks 107(222), 137-142, 2007-09-13

Appendix A

Multi-Agent System Methodologies Comparison Result Tables

Phases	ROADMAP	MaSE	Prometheus
System specification	Detailed	Medium	Detailed
Analysis	Detailed	Detailed	Detailed
Architectural design	Abstract, high-level	Detailed	Detailed
Detailed design	Not exists (Architecture)	Not exists (Architecture)	Detailed (BDI agents)

Figure A.1: Illustrates the scale of the details within each development phase[12]

Concept	ROADMAP	MaSE	Prometheus
Autonomy	Medium	Medium	High
Mental attitudes	Uses knowledge schema.	Agents do not have to be intelligent	Agents are intelligent agents. BDI agents.

Figure A.2: Presents the measure of agent concept that each methodology support[12]

Criteria	ROADMAP	MaSE	Prometheus
Clear notation	Strongly agree	Strongly agree	Strongly agree
Ease of learning	Strongly agree	Strongly agree	Agree
Ease of use	Agree	Strongly agree	Agree
Adaptability	Strongly agree	Strongly agree	Agree
Traceability	Strongly agree	Agree	Strongly agree

Figure A.3: Shows the scale of the modeling criteria within each methodology[12]

Property	ROADMAP	MaSE	Prometheus
Openness	High	Low	Medium
Environment	High	Medium	Medium
Abstraction	High	High	High
Traceability	High	High	High
Modelling	Medium	High	High
Complexity	Low	Low	Medium
Ease of use	Easy, requires some	Easy	Complicated,
Limitations	Lack of richer notations	Lack of	Highly
Language	Low	Medium	High
Reusability	High	Medium	Medium

Figure A.4: Compares the properties of the methodologies[12]

Phases	ROADMAP	MaSE	Prometheus
System Specification			Stakeholders Scenarios diagram
Analysis	Environment, Knowledge, Goal, Role, Revised role model, Social model	Use cases, Goal hierarchy, Sequence, Concurrent task diagram	Goal overview, Role, Data coupling diagrams
Architectural Design	Agent I, Service, Acquaintance model	Agent classes, Conversations	Agent acquaintance, System Overview Agent Descriptors Protocols
Detailed Design		Agent's internal architectures, Deployment diagram	Process , Agent Overview Diagrams, and Capacity, Capability overview, Event, Data, and Plan

Figure A.5: Illustrates the available activities in each development phase[12]

Methodology	Application
ROADMAP	Coarse-grained computational, complex, open systems
MaSE	Heterogeneous multi agent systems
Prometheus	Intelligent (BDI) agents' systems

Figure A.6: Illustrates the type of the system domain that each methodology is suitable for[12]

Methodology	Development Tool
ROADMAP	REBEL is a tool for building Goal Models and Role Models during the analysis stage.
MaSE	AgentTool, able to do printing, verification on developing system, and generating a skeleton code in java.
Prometheus	Prometheus Design Tool (PDT), which is able to do cross checking, saving diagrams as pictures, JDE (JACK Development Environment) generates JACK code.

Figure A.7: Summarizes the toolkits that are available for each methodology[12]

	GAIA	TROPOS	MAS-COMMONAKADS	PROMETHEUS	PASSI	ADELFE	MASE	RAP	MESSAGE	INGENIAS
Development lifecycle	Iterative within each phase but sequential between phases	Iterative and incremental	Cyclic risk-driven process	Iterative across all phases	Iterative across all phases (except for coding and testing)	Rational Unified Process (RUP)	Iterative across all phases	RUP	RUP	Unified software development process
Coverage of the development lifecycle	Analysis and Design	Analysis and Design	Analysis and Design	Analysis and Design	Analysis and Design	Analysis, Design and Implementation	Analysis and Design	Analysis and Design	Analysis, Design and Implementation	Analysis, Design and Implementation
Development perspective	Top-down	Top-down	Hybrid	Bottom-up	Bottom-up	Top-down	Top-down	Hybrid	Hybrid	Hybrid
Application domain	Independent (business process management, GIS, traffic simulation)	Independent (e-business systems, knowledge management, health IS)	Independent (Flight reservation, automatic control)	Independent (holonic manufacturing, online bookstore)	Independent (distributed robotics applications, online bookstore)	Dependent - adaptive systems (adaptive intranet system, system, finetabling system)	Independent (distributed planning, database integration system, computer virus immune system, automatic control)	Dependent - distributed organizational IS (supply chain management, enterprise resource planning)	Independent (network management, operational support systems, computer management systems)	Independent (collaborative information filtering, personal computer management, robot battles)
Size of MAS	≤ 100 agent classes	Not specified	Not specified, but possibly any size	Any size	Not specified	Not specified, but possibly any size	≤ 10 agent classes	Any size	Not specified, but possibly any size	Not specified, but possibly any size
Agent nature for verification and validation	Heterogeneous	BDI-like agents	Heterogeneous	BDI-like agents	Heterogeneous	Adaptive	Not specified but possibly heterogeneous	Reactive agents	Heterogeneous	Agents with goals and states
Edge of understanding of the process	No	Yes	Mentioned but no explicit steps/guidelines provided	Yes	Yes	Yes	Yes	No	Mentioned as future enhancement	Yes
Usability of the methodology	High	High	High	High	High	High	High	High	High	High
Usability of the methodology	Medium	Medium	Medium	High	High	High	High	Medium	Medium	High
Refinability	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Approach towards MAS development	a. OO b. RO (OrO)	a. i* modelling framework b. NRO	a. KE b. NRO	a. OO b. NRO	a. OO b. RO	a. OO b. NRO	a. OO b. RO (GO)	a. OO b. RO	a. OO b. RO (GO and BO)	a. OO b. RO

Figure A.8: Comparison of Concept[11]

Concepts & Properties	MaSE	Prometheus	Tropos
Autonomy	H/M/DK	H/NA/H	H/M/M
Mental attitudes	L/M/H	H/M/H	H
Proactive	H/M/H	H/M/DK	H
Reactive	M	H/M/DK	H/L/DK
Concurrency	H/M/H	M/L/DK	H/M/H
Teamwork	H/M/H	N/L/NA	H/H/M
Protocols	H	M/H/M	NA/M/M
Situated	M/L/H	H	H
Clear concepts	SA/A/A	A/A/DA	SA/A/N
Concepts overloaded	A/N/SA	N	SDA/N/DA
Agent-oriented	SA/A/A	SA	SA/A/SA

Figure A.9: Comparing a methodology 's properties, attributes, process and pragmatics. Notation: L for Low, M for medium, H for High, DK for Don't Know, SDA for Strongly Disagree, DA for Disagree, NA for Not Applicable, N for Neutral, A for Agree, SA for Strongly Agree, for no response. The first two entries in each column are the developers of the methodology, the third is the student. A single entry in the column indicates that all three answers agreed.[13]

Concepts & Properties	MaSE	Prometheus	Tropos
Modelling & Notation			
Static+Dynamic	SA/A/A	SA/A/A	N/A/A
Syntax defined	A/A/SA	SA/A/A	SA/N/A
Semantics defined	A/SA/SA	A	SA/A/A
Clear notation	A	SA/A/A	SA/A/N
Easy to use	SA/A/A	A/N/A	SA/A/N
Easy to learn	N/N/A	SA/NA/SA	SA/N/A
Different views	N/N/A	A/A/SA	SA/A/N
Language adequate & expressive	SA/N/N	A	SA/A/N
Traceability	A/SA/SA	A	A/N/A
Consistency check	SA/A/SA	SA/A/A	/A/DA
Refinement	SA/A/A	SA	SA/A/DA
Modularity	SA/A/A	SA/SA/A	SA/A/N
Reuse	N/SA/A	N/A/N	/A/DA
Hierarchical modelling	N/A/A	SA/A/A	SA/A/DA
Process			
Requirements	SPEH	SPEH	SPE
Architectural design	SPEH	SPEH	SPE
Detailed design	SPEH	SPEH	SPE
Implementation	SEH/SPE/S	SPEH/S/n	SE/SPE/SPEH
Testing & Debugging	SPE/n/n	SPEH/S/n	n
Deployment	SE/SPE/SPEH	n	n
Maintenance	n/SPE/n	n	n
Pragmatics			
Quality	N/DA/A	A/N/N	DA/A/_
Cost estimation	/DA/SA	DA/DA/N	DA/N/_
Management decision	/DA/SA	SDA/N/_	SA/A/_
apps	21+	6-20	1-5
Real apps	no	no	no
Used by non-creators	yes	yes	yes/no/no
Domain specific	no	no	yes/no/no
Scalable	/N/N	N/A/N	N/N/_
Distributed	/SA/SA	SA/A/N	N/A/_

Figure A.10: Comparing methodology 's properties, attributes, process and pragmatics. Notation: L for Low, M for medium, H for High, DK for Don 't Know, SDA for Strongly Disagree, DA for Disagree, NA for Not Applicable, N for Neutral, A for Agree, SA for Strongly Agree, for no response. S for Stage mentioned, P for Process given, E for Examples given, H for Heuristics given, n for none. The first two entries in each column are the developers of the methodology, the third is the student. A single entry in the column indicates that all three answers agreed.[13]

Appendix B

Example of Generated DASH Rule Set Code

Content of the generated file `Contract_Net_Protocol_Participant.rset`:

```
(rule-set Contract_Net_Protocol_Participant

  (property
  )

  (initial_facts
    (Contract_Net_Protocol_Participant :thread 0 :state init :max_thread 0)
  )

  (rule Contract_Net_Protocol_Participant_receive_CNP_cfp_on_Init_to_ComposingProposal
    (Msg :performative CNP_cfp :content ?task :protocol_from ?pfrom) = ?msg
    (Contract_Net_Protocol_Participant :state init :max_thread ?max_thread) = ?init
    -->
    (bind ?new_thread (+ ?max_thread 1))
    (modify ?init:max_thread ?new_thread)
    (bind ?new_state ?init)
    (Modify ?new_state:thread ?new_thread)
    (bind ?state ?new_state)
    (make ?state)
    (Modify ?state:state ComposingProposal)
    (Modify ?state:partner ?msg:from)
    (Modify ?state:partner_thread ?pfrom)
    (make (CNP_compose_proposal :task ?task :state ?state))
    (remove ?msg)
  )

  (rule Contract_Net_Protocol_Participant_receive_CNP_cfp_on_Init_to_ComposingProposal_from_user
```



```

( Msg :performative CNP_cfp :content ?task ) = ?msg
(Contract_Net_Protocol_Participant :state init :max_thread ?max_thread) = ?init
-->
(bind ?new_thread (+ ?max_thread 1))
(modify ?init:max_thread ?new_thread)
(bind ?new_state ?init)
(Modify ?new_state:thread ?new_thread)
(bind ?state ?new_state)
(make ?state)
(Modify ?state:state ComposingProposal)
(Modify ?state:partner ?msg:from)
(make (CNP_compose_proposal :task ?task :state ?state))
(remove ?msg)
)

(rule Contract_Net_Protocol_Participant_receive_CNP_send_proposal_to_WaitingForAccept
  (CNP_send_proposal :proposal ?proposal :timeout_after ?timeout_after :return_to ?interface) = ?command
  (bind ?rs ?interface:state)
  (Contract_Net_Protocol_Participant :state == ?rs:state :thread == ?rs:thread ) = ?state
  -->
  (send :performative CNP_proposal :content ?proposal :to ?state:partner :protocol_to ?state:partner_thread
    :protocol_from (Contract_Net_Protocol_Participant :thread ?state:thread))
  (Modify ?state:state WaitingForAccept)
  (alarm :after ?timeout_after :content (Contract_Net_Protocol_Participant :state WaitingForAccept
    :thread ?state:thread))
  (remove ?command)
)

(rule Contract_Net_Protocol_Participant_receive_CNP_refuse_to_Terminate
  (CNP_refuse :return_to ?interface) = ?command
  (bind ?rs ?interface:state)
  (Contract_Net_Protocol_Participant :state == ?rs:state :thread == ?rs:thread ) = ?state
  -->
  (Modify ?state:state Terminate)
  (remove ?command)
)

(rule Contract_Net_Protocol_Participant_receive__Alarm_on_WaitingForAccept_to_Terminate
  (Msg :performative __Alarm :content (Contract_Net_Protocol_Participant :thread ?athread
    :state WaitingForAccept)) = ?msg
  (Contract_Net_Protocol_Participant :state WaitingForAccept :thread == ?athread) = ?state
  -->
  (Modify ?state:state Terminate)
  (remove ?msg)
)

```

```

)

(rule Contract_Net_Protocol_Participant_receive_CNP_accept_on_WaitingForAccept_to_PerformingTask
  (Msg :performative CNP_accept :content ?task :protocol_to (Contract_Net_Protocol_Participant
    :thread ?tthread) :protocol_from ?pfrom) = ?msg
  (Contract_Net_Protocol_Participant :state WaitingForAccept :thread == ?tthread) = ?state
  -->
  (Modify ?state:state PerformingTask)
  (Modify ?state:partner ?msg:from)
  (Modify ?state:partner_thread ?pfrom)
  (make (CNP_perform_task :task ?task :state ?state))
  (remove ?msg)
)

(rule Contract_Net_Protocol_Participant_receive_CNP_task_succed_to_Terminate
  (CNP_task_succed :result ?result :return_to ?interface) = ?command
  (bind ?rs ?interface:state)
  (Contract_Net_Protocol_Participant :state == ?rs:state :thread == ?rs:thread ) = ?state
  -->
  (send :performative CNP_inform :content ?result :to ?state:partner :protocol_to ?state:partner_thread
    :protocol_from (Contract_Net_Protocol_Participant :thread ?state:thread))
  (Modify ?state:state Terminate)
  (remove ?command)
)

(rule Contract_Net_Protocol_Participant_receive_CNP_task_failed_to_Terminate
  (CNP_task_failed :return_to ?interface) = ?command
  (bind ?rs ?interface:state)
  (Contract_Net_Protocol_Participant :state == ?rs:state :thread == ?rs:thread ) = ?state
  -->
  (Modify ?state:state Terminate)
  (remove ?command)
)

(rule Contract_Net_Protocol_Participant_kill_terminated_thread
  (Contract_Net_Protocol_Participant :state Terminate) = ?state
  -->
  (remove ?state)
)
)

```