On the Mutability of Protocols

Jarred P. McGinnis

Doctor of Philosophy

Centre for Intelligent Systems and Applications

School of Informatics

University of Edinburgh

2006

Abstract

The task of developing a framework for which agents can communicate reliably and flexibly in open systems is not trivial. This thesis addresses the dichotomy between reliable communication and facilitation of the autonomy of agents to create more flexible and emergent interactions. By the introduction of adaptations to a distributed protocol language, agents benefit from the ability to communicate interaction protocols to elucidate the social norms (thus creating more reliable communication). Yet, this approach also provides the functionality for the agent to unilaterally introduce new paths for the conversation to explore unforeseen opportunities and options (thus restoring more autonomy than possible with static protocols).

The foundation of this work is Lightweight Coordination Calculus (LCC). LCC is a distributed protocol language and framework in which agents coordinate their own interactions by their message passing activities. In order to ensure that adaptations to the protocols are done in a reasonable way, we examine the use of two models of communication to guide any transformations to the protocols. We describe the use of FIPA's ACL and ultimately its unsuitability for this approach as well as the more fecund task of implementing dialogue games, an model of argumentation, as dynamic protocols.

The existing attempts to develop a model that can encompass the gulf between reliability and autonomy in communication have had varying degrees of success. It is the purpose and the result of the research described in this thesis to develop an alloy of the various models, by the introduction of dynamic and distributed protocols, to develop a framework stronger than its constituents. Though this is successful, the derivations of the protocols can be difficult to reconstruct. To this end, this thesis also describes a method of protocol synthesis inspired by models of human communication that can express the dialogues created by the previous approaches but also have a fully accountable path of construction. Not only does this thesis explore a unique and novel approach to agent communication, it is tested through a practical implementation.

Acknowledgements

Nothing of worth is created inside a vacuum. The worth of this document has yet to be determined but if there is any success and insight found within, the responsibility of its creation cannot be mine alone. There are numerous people who have given me assistance in its creation: from friends and acquaintances who during casual chats unintentionally sparked some new idea, to my supervisors who have patiently guided me through the PhD process.

I fear any enumeration of the people who deserve acknowledgement will inevitably be incomplete. Yet, two individuals have made a huge impact upon the completion of this thesis by their, for lack of a better phrase, academic midwifery. David Robertson and Chris Walton have been the best supervisors both in wisdom and temperament.

To all my fellow research students, especially the men of the four one five, who have at one time dotted my i's and crossed my t's as well as paused their own work so that I could discuss with them my own.

I would like also thank my family for the continual support that I have received throughout my education. It is my family who is most responsible for enabling me to complete my undergraduate degree and providing me with the opportunity to strive for the goal to which this thesis is a testament.

I sincerely thank all of you.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jarred P. McGinnis*)

Publications

Through out the course of this degree, I have taken advantage of the immeasurably helpful input provided by the peer review system. Most of the ideas developed in this thesis have been presented or published for a number of conferences and workshops.

- McGinnis, J., Robertson, D., and Walton, C. (2006). Protocol synthesis with dialogue structure theory. Invited Paper in Maudet, N., Moraitis, P., Rahwan, I., and Parsons, S., editors, *Argumentation in Multi-Agent Systems: Second International Workshop, ArgMAS 2005,* LNAI. Springer-Verlag, Utrecht, The Netherlands.

- McGinnis, J., Robertson, D., and Walton, C. (2005). Protocol synthesis with dialogue structure theory. In Proceedings of *the International Conference for Autonomous Agents and Multi-Agent Systems*, AAMAS, pages 13291330

- Chesnevar, M. C. C., McGinnis, J., Rahwan, I., Reed, C., Modgil, S., Simari, G., South, M., Vreeswijk, G., and Willmott, S. (2005). Aif: Argumentation interchange format strawman model. Technical report, Agentlink.

- McGinnis, J., Robertson, D., and Walton, C. (2005). Protocol synthesis with dialogue structure theory. In Proceedings of *Proceedings of the European Union Multiagent Systems Workshop*, 2005.

- McGinnis, J. and Robertson, D. (2004b). Realizing agent dialogues with distributed protocols. In Developments in Agent Communication, volume 3396 of LNAI. Springer-Verlag.

- McGinnis, J. and Robertson, D. (2004). Dynamic and distributed interaction protocols. In Proceedings of the AISB 2004 Convention, pages 4554.

- Brak R., Fleuriot J. and McGinnis, J., Theorem proving for protocol languages, *Proceedings of the European Union Multiagent Systems Workshop*, 2004.

- McGinnis, J., Robertson, D., and Walton, C. (2003). Using distributed protocols as an implementation of dialogue games. Presented EUMAS 2003.

To S.

Table of Contents

1 Introduction **1**

1.1 Agent Communication 2

1.1.1 Approaches to Agent Communication 3

1.1.2 Agent Communication and Content Languages 4

1.2 Objectives and Hypotheses 6

1.2.1 Bridging the Two Cultures of Agent Communication 9

1.2.2 What I am *Not* Trying to do 9

1.3 Thesis Outline 11

2 The Current Landscape of Agent Communication **13**

2.1 Distributed Protocols 14

2.1.1 The Melbourne Strain 15

2.1.2 The Portuguese Approach 16

2.2 Mentalistic Approaches 17

2.2.1 KQML 17

2.2.2 FIPA ACL 20

2.3 Social and Centralised (Electronic Institutions) 23

2.3.1 Islander and Ameli 23

2.3.2 Others 26

2.4 Social Agent-centric Approaches 26

2.5 Rationalistic Approaches 29

2.5.1 Dialogue Games 31
2.6 Summary and Analysis 34

3 The Lightweight Coordination Calculus(LCC) 37
3.1 Syntax 39
3.2 Expansion Engine and Framework 41
3.3 Agent Engineering Requirements 46
3.4 An Example 47
3.5 Chapter Summary 49

4 Realising Dynamic Protocols with LCC 53
4.1 Communicating FIPA Semantics 54
4.1.1 Problems of FIPA 55
4.1.2 FIPA Communicative Acts as Protocols 56
4.2 Dialogue Games 65
4.2.1 Writing Dialogue Games as LCC Protocols 66
4.2.2 Creating Complex Dialogue Games 82
4.2.3 Chapter Summary 89

5 Interaction Protocols by Dialogue Structure Synthesis 93
5.1 Using Dialogue Structures 94
5.2 Transformations 98
5.3 Synthesising Protocols 111
5.4 An Example Using Dialogue Games for Synthesis 115
5.5 Chapter Summary 125

6 System Design and Implementation 129
6.1 Basic Framework 130
6.2 Creating Dynamic Protocols 132
6.3 Synthesising Protocols 141

6.4 Chapter Summary 143

7 Conclusions **145**

7.1 Summary 145

7.2 Evaluation of Contribution 146

7.3 Future Work 150

7.4 Possible Extensions 151

7.4.1 Trust 151

7.4.2 Permission 152

7.4.3 Possible Collaborative Technologies 152

A Complete Trace of the Dialogue State from 4.2.2.2 **155**

B Prolog Code **157**

B.1 `insert.pl` 157

B.2 `syn.pl` 159

B.3 `rawinterface.pl` 161

B.4 `syninterface.pl` 164

Bibliography **167**

List of Figures

1.1 Layer Cake of Agent Communication 5

2.1 Two-dimensional Matrix of Approaches to Agent Communication . . 14
2.2 KQML Performatives 18
2.3 An Example of a KQML Performative 19
2.4 KQML Semantics for `tell` 20
2.5 FIPA's Communicative Acts 21
2.6 FIPA ACL's SL Semantics for `inform` 22
2.7 *Example Electronic Institution and Scene* 25
2.8 Commitment Revision ACL 28
2.9 Typology of Dialogues 33
2.10 Summary Table of Communication Models 35

3.1 An Abstract Syntax of the Protocol Language 40
3.2 Rules for Expanding an Agent Clause 43
3.3 Sequence of Rewrites 44
3.4 *Figure of the Standard View of Protocol led Multiagent System* 45
3.5 *Figure of the Distributed Protocol Multiagent System* 46
3.6 Conversation Space as LCC Agent Clauses 48
3.7 Expansion of Dialogue State of the Agents 49

4.1 LCC for the Inform Communicative Act 58
4.2 FIPA Inform as LCC Agent Clauses 58

4.3 *AUML diagram for FIPA query protocol* 59
4.4 FIPA Query Protocol as LCC 60
4.5 *AUML diagram for contract net protocol* 61
4.6 FIPA Contract Net Protocol as LCC 62
4.7 FIPA CAs as an Dynamic LCC Clause 63
4.8 *Graphical Representation of the Information seeking Game* 68
4.9 A Protocol for an Information Seeking Dialogue Game 70
4.10 The Recursive Roles for the Information Seeking Dialogue Game Protocol 72
4.11 *Graphical Representation of the Persuasion Game* 74
4.12 A Protocol for a Persuasion Dialogue Game 75
4.13 The Recursive Roles for the Persuasion Dialogue Game Protocol . . . 76
4.14 *Graphical Representation of the Inquiry Game* 77
4.15 A Protocol for an Inquiry Dialogue Game 79
4.16 The Recursive Clauses for the Inquiry Dialogue Game Protocol 80
4.17 The Recursive Clauses for the Inquiry Dialogue Game Protocol[Cont'd] 81
4.18 A Protocol to Define the Control Layer 83
4.19 A Protocol to Return to the Control Layer 84
4.20 Control Layer Part of the Dialogue State 85
4.21 Persuasion Role added to the Dialogue State 86
4.22 Resulting Dialogues State after Persuasion Role were Added 87
4.23 Continuation of the Example's Dialogue State 89
4.24 *Graphical Representation of the Complex Dialogue Game Example* . 90
4.25 Continuation of the Example's Dialogue State 91

5.1 The Twenty-four Possible Transformations 99
5.2 The Twenty-four Possible Transformations [cont'd] 100
5.3 After the First Pruning 102
5.4 After the Second Pruning 104

5.5 After the Final Pruning of Transformations 105
5.6 The Vocabulary of Transformations 107
5.7 Choosing which Synthesis Rules 112
5.8 An illegal Transformation 112
5.9 Revision of the Synthesis of Figure 5.8 113
5.10 The Placing of Constraints 114
5.11 *Graphical Representation of the Information seeking Game* 115
5.12 A Protocol for an Information Seeking Dialogue Game 116
5.13 The Recursive Roles for the Information Seeking Dialogue Game Protocol 117
5.14 Synthesis Rules for an Information Seeking Game 118
5.15 Resulting Dialogue State Using the Information Seeking Protocol . . 121
5.16 Synthesis and Expansion of the Same Information Seeking Dialogue Game 122
5.17 Synthesis and Expansion of the Same Information Seeking Dialogue Game[cont'd] 123
5.18 *Two Versions of the Dialogue State* 126

6.1 Basic LCC Framework 131
6.2 Dynamic LCC Framework 133
6.3 A Protocol to Define the Control Layer 134
6.4 Rules for Expanding an Agent Clause 136
6.5 Dialogue State after agentA's expansion 137
6.6 Dialogue State after *agentB*'s response 138
6.7 Dialogue State after *agentB*'s response 140
6.8 Relationship Diagram Between Components 142

Chapter 1

Introduction

The term ‘computer’ used to be a job description rather than describing ‘a machine for performing calculations automatically’. There were rooms filled with humans doing computations by hand for days. Today, it is taken for granted that those same computations are performed instantaneously and automatically without the need for human intervention. We trust our machines to get one plus one correct every time. The machines now called computers and the humans that have designed them have continued to push the bounds of what is considered a necessarily human task and what can be delegated to machines. As the internet makes massive scale distributed computation more and more ubiquitous, communication and socially-oriented tasks seem to be another human activity that can be left to the machines.

The agent paradigm not only has the baggage inherited from its object-oriented and distributed systems pedigree, but there are also issues concerning concurrency, coordination, and its use of anthropic concepts for internal activities of agents and communication between them. Yet, the solutions to these difficulties hold the promise of making the pervasive uncertainty of communication in an open, automated and distributed system more manageable, the benefits of which are incalculable. The approach described in this thesis attempts to make the minimal engineering requirements on agents in order for them to utilise and adapt the protocols. Despite this restriction, the goal of this

project is to facilitate complex, robust, and reliable communication between agents.

It is likely that the reader of this thesis will be acquainted with the programming paradigm of agency and agent communication. This carries the risk of making an introductory chapter, such as this, superfluous. Rather than strictly introduce the broad research field, the purpose of this chapter is to make clear any implicit assumptions, views, and personal understandings of the many (and at times ill-defined) terms used in the literature. Firstly this chapter explains broadly communication in multiagent systems. In an equally general way, I will demarcate the boundaries of what I set out to accomplish and why. Finally, I will provide a description of the thesis' chapters.

1.1 Agent Communication

An agent in isolation is a sad creature indeed, but they are a rarity. It is an underutilization of the technology. Most agent research implies some form of communication even if it is rudimentary and uninteresting. For example, "blocks world" problems are mainly concerned with the development of more sophisticated agent-based problem solving techniques. Their communication is limited to stating locations of boxes, but issues of coordination and how agents work in an environment made dynamic by the activities of others must still be addressed. At the other end of the spectrum, agents are treated as dialogical only entities. These agents interact with the system and other agents only through 'verbal' utterances. This is the foundation for many e-commerce multiagent systems and argumentation theoretic systems.

Agent communication is the focus of this thesis and in particular the development of a means for communication that is more interesting than the rudimentary messages passed between the blocks world agents. Our focus is on communication in open multiagent systems. Open because agents are free to enter and leave at will and open because agents are not necessarily designed and engineered the same. Their internal make up is hidden from examination, black boxes. Black boxes in the sense that

no assumptions can be made about the agents' problem solving apparatus. The only assumption is that they can accept output from the system and are capable of returning input. At the very least they differ in their observation of the system. There is no assumption of a global view of the system. Each agent can differ in both spatial and temporal as well as semantic view points of the system. The autonomy of the agents creates additional difficulties for their performance of actions. No longer can the execution of a task be guaranteed as is the case for remote function calling or distributed processing. Agents must act in the world through communication. They must reflect upon the system, determine who might be able to assist their task, and initiate a dialogue with that other agent. This activity creates the dynamic environment of a multiagent system. Participants are constantly acting in and reacting to the system, keeping everything in flux. Their communication as a result is asynchronous. All this is a recipe for chaos, and there are various approaches to restore a measure of order. These approaches can be roughly described as either "top-down" or "bottom-up". At times the divide between the two seems unreconcilable. This need not be so, and this thesis makes progress towards reconciling these two cultures of agent communication.

1.1.1 Approaches to Agent Communication

A "top-down" approach to agent communication views agents as members of a society. This paradigm is typified by electronic institutions (EI) [Noriega, 1997, Vázquez-Salceda, 2003, Esteva et al., 2002, Esteva et al., 2000, Estava et al., 2001, Dignum, 2003, Cortés, 2004, Esteva et al., 2004]. The society has mores, norms, and traditions. For agents to participate in multiagent systems and thus participate in the society, the onus rests upon the individual engineer to design its agent to follow the rules of the society. The consequence of this is more reliable agent communication. It also is more scalable due to the ability to know the global state of multiagent system as the dialogical activities are specified by the society. This comes at a cost of autonomy. Agents are not completely free to explore the conversation space; conversation space

being defined as the set of all possible meaningful sequences of messages given an agent communication language. Agents can only converse by following the sequences allowed by the society.

The advantages and disadvantages of the "bottom-up" approaches are reversed. Mentalistic and rationalistic approaches such as FIPA's ACL specifications and dialogue games are examples of the bottom-up approaches. Each agent is assumed to have an understanding of the basic elements for communication. Given its state, it is expected to infer the appropriate dialogical action. The global behaviour of the system emerges from all the individual communicative decisions made by each agent. There is usually no external check upon the message that the agent sends. This allows the greatest amount of autonomy for individual agents but the risk of disorder or break down of the system increases as the population of the system or the complexity of the conversations increase.

Regardless of the high-level approach to communication, there are some aspects that are shared. For example figure 1.1 shows the levels of communication. At the lowest level there is the physical mechanism for ensuring messages are delivered and received unadulterated (e.g. HTTP). This is a well trodden research path upon which we can safely travel. More abstractly there exists the level concerning a language of communicative primitives. Finally, there is the level, which is our ultimate concern, that governs whole patterns of interactions, social norms, and communication within multiagent systems.

1.1.2 Agent Communication and Content Languages

Agent communication is predominantly based on speech act theory developed by two philosophers during the mid twentieth century [Searle, 1969b, Austin, 1962]. The idea is that communication is similar to other actions in that it furthers the intentions of utterer to accomplish some task. These locutions attempt to modify the meaning of the content of the message. Just as humans use inflection in spoken communication and

Figure 1.1: Layer Cake of Agent Communication

punctuation in writing to further color the intent of the person communicating. The simplest sentence, such as 'stop', can have several distinct meanings by the addition of a question mark, exclamation mark or full stop. So it is for agent communication. The content of the message which can be expressed in a knowledge representation language such as KIF [Genesereth and Fikes, 1992] or a statement in propositional logic. This is usually wrapped by a performative or locution. So, the proposition "Caesar is dead" can have a different meaning if it is expressed within a locution that is specified as a query, command or statement. The communication language, content language and the ontology must somehow be agreed by the participating agents, regardless of the implementation of the model or means of communication. Without this assumption little progress can be made, but as we will show, the use of LCC and distributed protocols is able to ease the burden somewhat.

1.2 Objectives and Hypotheses

What are the questions this research proposes to answer? What are the boundaries? What ground shall we tread? What territory should be avoided? The answers to these questions constitute our objectives. Through them we shall formulate our Hypotheses and propose methods to confirm those hypotheses and achieve our objectives.

This work is focused on agent communication. As it occurs in *open multiagent systems* is of particular interest, as well as the use of *interaction protocols*. Interaction protocols are a well known and commonly used paradigm for agent communication. The novelty of this thesis is the use of *distributed* and *dynamic* protocols. Here a distributed protocol means a protocol that has been written not only in a computational and executable manner but also in such a way that the protocol can be explicitly communicated and understood by another agent. Typically, protocols are designed from human written specifications and encoded for the internal use of a single agent. The shared model of interaction is achieved by the agent engineers encoding their agents

with regard to the same protocol specification. Dynamic protocols are interaction protocols that can be modified by the agents during execution.

1.2.0.1 Novel *Means* of Communication

The use of distributed protocols is already novel in the realm of agent communication. By allowing agents, who now have providence over their interaction protocols, to modify them creates interesting possibilities as well as difficulties. Chapter 3 describes the LCC language and framework which provides this means. The question that remains is whether this means is enough to produce *dynamic* protocols. Also, can this extended functionality be done in a modular way (i.e. without requiring a modification to the original protocol language and framework)? In other words, the interdependencies between the language, framework, and transformations should be minimal and changes to one should not generally affect the other. Given the use of dynamic protocols, are there any unique problems that need to be addressed and are they surmountable? Given these objectives, our hypothesis will be an optimistic yes, and our method will be a engineering task to ensure it is realised.

1.2.0.2 Exploit LCC to Communicate Social Models

Several consequences arise from the use of the distributed protocols and framework of LCC. A common method for developing communicating agents is for a human to design and implement their agent in accordance with a specification. In order for this agent to communicate with other agents, it relies on other engineers interpreting and implementing their own agents in a sufficiently similar way to allow some progress in communication. Instead, an engineer develops the social norms for the agent as a distributable and computable protocol that the agent's communicative partners can receive, understand, and utilise. There is no ambiguity to whether these agents can communicate. It is already known that LCC, as discussed in chapter 3, is capable of expressing social models that rely on static and explicit protocols. What this thesis

shall explore is whether by the use of LCC extended to create dynamic protocols can reproduce the more agent-centric and flexible models of communication. Our methodology shall be to examine the research literature for models of communication that fit those criteria, and then our task will be to select a sample of those models that are widespread and well-known as well as already proven capable of representing interesting and useful agent dialogues. It is our hypothesis that a dynamic LCC will be able to express those models of agent communication in the novel way describe in the previous subsection.

1.2.0.3 Allow 'Truly' Heterogeneous Agents to Communicate

Open multiagent systems assume agents are not necessarily homogenous. In practice agents always have a lot of similarities and assumptions. Usually this uniformity is enforced by external agents or software (e.g. Electronic Institutions) or through thoroughly defined specifications for individual agent behaviours. Obviously a certain amount of regularity is necessary, such as the sharing of an agent communication language and a transport protocol, but one aim of this thesis is to alleviate some of the burden of conformity by giving the agents the means to communicate the social norms with which they communicate. A question relating to the previous subsection; if these social norms can be expressed as dynamic protocols, does this reduce some of the assumptions required to get heterogenous agents to communicate? For example, can social commitments or dialogue games be expressed sufficiently as protocols to allow agents to communicate without necessarily understanding commitment revision or the commencement rules? Our hypothesis is that by the use of a distributed protocol that can accommodate the vagaries agent centric models of communication, it is indeed possible to mitigate the assumptions on dialogical partners. The method to prove this point will be shown through the translation and use of dialogue game specifications as dynamic LCC protocols.

1.2.1 Bridging the Two Cultures of Agent Communication

Ultimately, the implementation of "bottom-up" approaches as protocols, distributing that protocol, and allowing modifications when the agent requires it creates a hybrid approach. It provides the stability of a protocol led approach but the flexibility of the agent-centric ones. This thesis shows that the dogmatic divide of the two main camps of agent communication need not be. It is more likely that a amalgamation of the two will be preferred. The following of strict protocols is used when appropriate and emergent behaviours are allowed when necessary, but it is always done in the context of communicable and mutable protocols. Though this thesis focuses on the use of dialogue games in this context, the ideas are readily transferable to the other "bottom-up" models described in the literature review. The question here is whether this research achieves its goal of having the flexibility of the agent-centric bottom-up approach to communication while also preserving some of the reliability and predictability of protocol led 'top-down' models. By developing an implementation to this for a model like dialogue games that achieves those aims, we can show that our hypothesis that it was possible is correct.

All the mentioned approaches suffer from the dialects problem. Even standardisation attempts have only succeeded to achieve superficial conformance. Instead, it would be better to develop a means to reconcile the differences rather than hope for some convergence at the specification level. It is more likely that even more models of communication will be proposed further exacerbating any notion of reconciliation at an abstract level. It is for this goal that this thesis also strives, and answer whether dynamic distributed protocols is the mechanism to achieve it.

1.2.2 What I am *Not* Trying to do

In addition to stating my hypothesis and goals for this project, it is useful to also clarify some of the boundaries for the thesis. This is a list of what I am *not* attempting to do.

1.2.2.1 replace any model of communication

On the occasions when the incremental successes of my research were published in conference proceedings, a common review was to defend whatever model of agent communication I was suggesting to improve by this approach. This is a misunderstanding. The intention is to improve the means of implementing the various models of communication that have been developed, not replace them. Chapters 4 and 5, describe a novel mechanism for agent communication, but it relies on the abstractions and formalisms developed by the authors mentioned in 2.

1.2.2.2 create only another theoretical model

There is a well recognized deficiency in many aspects of agent research and that is the gulf which separates the theoretical work from the practical. This is changing and recent research is addressing that divide. This thesis is also a part of that movement. A goal here is to instantiate and implement existing theoretic work.

1.2.2.3 create yet another language (YAL)

The work of this thesis should be done within the context of existing languages and paradigms. This work will restrict itself to a minimal amount of modifications to the LCC framework. In the spirit of LCC, the work here aims to put the smallest additional burden upon agent engineers.

1.2.2.4 claim this is a panacea

There will be no panacea. There will be inappropriate domains for this approach. Indeed the spirit of this work is in reaction to attempts to dictate an orthodoxy to agent communication. The power (and the difficulty) of multiagent communication comes from its diversity. The philosophy of this work is to spend energy devising a method allowing agents to communicate their semantics for communication and social norms not to proscribe all but one understanding of agency and agent communication. The

final chapter discusses in more detail the domains for which this approach in its current form is ill suited but also what steps can be made to redress those shortcomings.

1.3 Thesis Outline

The following chapter reviews the existing literature. Chapter 3 describes the language and framework that is the foundation for this research work. It describes the LCC language and framework. This chapter is essential for understanding the approach and spirit of the thesis. Chapters 4 and 5 are the main course. The real meat of the work is found here. The former describes the development of the dynamic transformations of interaction protocols with an emphasis on its use in conjunction with dialogue games. The latter explains a further expansion of the idea of adaptable protocols. Protocols are now synthesised completely at run time. This process is informed by notions developed from research into dialogue structures for use in computational linguistics. Finally, the last chapter concludes with a summary, evaluation of the contribution, and a discussion of future avenues for research on this topic.

Chapter 2

The Current Landscape of Agent Communication

The field of agency has evolved rapidly, and many solutions have been developed to address the problem of interoperability in automated communication. Each one must make assumptions in order to solve a portion of the problem. This is absolutely necessary, as developing a truly open, heterogenous multiagent system is a grand task, and different environments require different solutions given their assumptions. Electronic Institutions (EI) [Noriega, 1997, Vázquez-Salceda, 2003, Esteva et al., 2002, Esteva et al., 2000, Estava et al., 2001, Dignum, 2003, Cortés, 2004, Esteva et al., 2004] with its top-down approach gives us reliable and robust multiagent systems. Yet it is at the cost of a third party's intervention or in the worse case a complete loss of autonomy. Distributed protocols, which ultimately provide us our means to enact the method described within these pages have been largely unexplored, especially with respect to on-the-fly protocol construction. Mentalistic (BDI based) approaches to communication are widely used but their inappropriateness for open systems is even more widely known (e.g. the semantic verification problem [Wooldridge, 2000]). Permissive approaches go a great distance, but tend to become a hodgepodge of various models. The game-theoretic approaches

are more generally applicable than the permissive or social approaches, but tend to be heavily theoretic and dependant on ideal world assumptions. This thesis borrows from and expands upon these foundations. Although, it does not completely solve the grand task. Instead it describes two generic approaches that offer flexibility and stability necessary for more complex agent dialogues.

This chapter describes the existing literature in the field, and its relation to the work of this thesis. Any division of categories is going to be unsatisfactory for one reason or another. The two-dimensional matrix show in figure 2.1 is one division to highlight the differences of top-down approaches to bottom-up ones. All of the approaches placed in the figure are models of communication. In addition to these models is the mechanism of distributed protocols. This mechanism will be used to achieve the thesis' goals and implement several of the models in an novel manner.

	Agent-centric	**Centralised**
Mentalistic	KQML FIPA ACL	
Social	Commitment-Based Conversation Policies	EIs
Rationalistic	Argumentation Dialogue Games	Game-Theoretic

Figure 2.1: Two-dimensional Matrix of Approaches to Agent Communication

2.1 Distributed Protocols

Distributed protocols is largely an unexplored approach to agent communication. The orthodoxy has individual engineers developing an agent's communicative model by the interpretation of formal, graphical, or natural language descriptions of a multiagent

system's interactions. Distributed protocols take the view that the agents themselves can communicate, in a computationally digestible format, the interaction protocols for a multiagent system. The advantage is that agents are not tied to a set of predefined protocols that their creator foresaw.

Most of the existing literature for this approach is developed for closed systems, and have a simple model of representations (e.g. finite state machines). All of them restrict themselves to static protocols. The two most prominent approaches are described below.

Chapter 3 is devoted to another distributed protocol language, LCC, that is more developed and expressive than the work described in this section and forms the basis for the dynamic protocols and protocol synthesis that constitute the main contributions of this thesis.

2.1.1 The Melbourne Strain

This work [de Silva, 2002] concentrates on developing algorithms for agents to evaluate a received protocol represented as petri-nets. The concerns addressed are syntax conformity, loop detection, and determining the safety of an agent's private information.

The author echoes the importance of local representation of protocols. By giving the protocol's definition with respect to an individual's actions within a system rather than defining the entire system's catalogue of interaction, agents do not need to sift through the global protocol to find their own role held within. This disambiguation also reduces some of the redundancy of states from the agent's perspective.

This research also affirms the importance of insulating the agent's internal functions from the interaction protocol being used. The declaration of functions and variable instantiation is important for elucidation of a protocol's semantics, but the actual definition of these would impede on an agent's autonomy.

The authors chose the extended petri-net representation because of the limitations

of an FSM representation that has been used in previous implementations of distributed protocols [Nodine and Unruh, 1997, Martin et al., 1996]. Though this representation is more expressive, there is no directly implementable format and it requires translation to a machine reliable format.

This work's findings could extend this thesis' contribution. Issues of trust and safety of protocols have been ignored in this thesis, but it is possible to guarantee that if a given protocol is error free that no corruption will occur by introducing protocol adaptations. The concern of this thesis is not whether a received protocol is deficient, but instead it is an exploration into the possibilities of an agent system with interaction protocols being distributed in a peer to peer manner. In particular the possibilities of run time protocol creation which is a topic that is not addressed by the Melbourne research.

2.1.2 The Portuguese Approach

This work describes an implementation of the use of XML specifications translated from AUML diagrams and then converted to a set of production rules to provide the interaction protocol for the agent to use [Freire and Botelho, 2002]. Beside the representation of AUML protocol specifications as XML, the authors propose two other components for the execution of explicitly represented protocols. Another requirement is an interpreter for the XML protocols. Their approach is a translation from the protocol specifications to a set of `if-then` production rules. Finally, there should be an interface between the agent's decision making apparatus and the protocol's XML. They propose a predicate for the interpreter to specify a set of communicative actions for the agent to choose from, and a predicate to indicate the chosen action.

The authors reinforce our claim of the importance of separating the agent's internal deliberative model from its communicative model to gain the well documented advantages of modularization. The authors argue that by making an agent's internal rationalizations independent of a set of fixed predefined protocols and providing the

agent with a generic interpreter of interaction protocols allows them to participate in unpredicted conversations.

The proposed approach for agents to understand these protocol specifications is to convert them to production rules. There are also number of critiques against using AUML for the representation of agent interaction protocols [Paurobally et al., 2004]. The author's decision to use this format seems to be a legacy from FIPA specifications. The authors identify the usefulness of defining protocols from an agent perspective versus a global one. As this paper is a proposal, the authors do not claim to have a concrete syntax for interaction protocols nor give much detail on the interpreter or interface. It is clear that the protocols are static, defined beforehand, and require a third party, the protocol server agent, which is a potential drawback that LCC avoids.

2.2 Mentalistic Approaches

Mentalistic or Belief-Desire-Intentions (BDI) approaches to agent communication have come from the seminal books of [Searle, 1969b] and [Austin, 1962]. The idea is that human utterances have observable side effects much like physical actions. By utterances alone, humans can affect the state of the world. The commonly used examples of saying 'I do' at your wedding ceremony or a country's leader declaring war illustrate these changes of state obtained by human utterances.

2.2.1 KQML

The *illocutionary* model of communication fits well with BDI theories of agents, and the need for a standardisation of a communication language was quickly recognised. The first language to gain wide recognition is the KQML language [Finin et al., 1994] developed as part of the Knowledge Sharing Effort (KSE). Initially KQML was developed to enhance knowledge sharing and not for agents per se, but the conceptual underpinnings were readily applicable. Of the many issues facing agent communica-

tion, it was the need to not only share information but also the complex attitudes the agent held with regard to that information (e.g. requesting, informing, questioning). The development of the KQML's ACL was the attempt to create a set of performatives to capture the various propositional attitudes an agent might want to express. The KQML set of performative names are listed in the figure 2.2.

Insert	Uninsert	Delete-one	Delete-all
Undelete	Tell	Untell	Broadcast
Forward	Achieve	Unachieve	Broker-one
Recommend-one	Recruit-one	Broker-all	Recommend-all
Recruit-all	Advertise	Unadvertise	Deny
Subscribe	Stream	Eos	Standby
Ready	Next	Rest	Discard
Ask-if	Ask-one	Ask-all	

Figure 2.2: KQML Performatives

Others [Cohen and Levesque, 1995] have criticized the use of the term performative as it implies the *success* of the communication primitive. Ultimately, the term as well as others used such as FIPA ACL's 'communicative act' or more generally 'locution' refers to communication primitives that are sufficiently similar to be used interchangeably.

KQML was developed to be independent of low level transport details (e.g. TCP/IP, IIOP, etc.) as well as the content language and ontology. The basic concepts of KQML resonate through other models of agent communication. This is especially true of FIPA ACL, but also for other non-mentalistic approaches.

A KQML message can be dissected into two parts; locution and content parts. The content portion of the message holds the information being expressed in the agent's own representation language. This part also contains the needed information for the

low-level details for communication such as identifiers for the sender and receiver, the ontology, content language, etc.. The heart of KQML is the speech-act that acts as a wrapper around the content.

```
(tell
      :sender       agent-bob
      :content      ‘‘price(item64,34,gbp)’’
      :receiver     agent-tom
      :in-reply-to  id7.24.97.45391
      :language     Prolog
      :ontology     standard-commerce
)
```

Figure 2.3: An Example of a KQML Performative

Figure 2.3 shows an example of a KQML performative. This performative is an instance of `tell`. It encapsulates the content part containing the details of the message to which the speech act adds its *illocutionary force*. The parameters shown are reserved keywords starting with a colon followed by their values. From the example, the parameter `:sender` has the value of ‘`agent-bob`’. The entire message could be read as “Agent-bob tells agent-tom that the price for item64 is 34 British pounds (gbp). The content of this message is using Prolog as its language and standard-commerce as its content ontology. This `tell` is in response to a previous message labeled as id7.24.97.45391. The set of performatives defined by [Finin et al., 1994] is not considered to be exhaustive, closed, or minimal and there exists a number of variants of KQML [Labrou et al., 1999].

The semantics of the performatives are given in terms of preconditions, postconditions, and completion conditions. Conditions are expressed for both the speaker and hearer of the utterance.

$\texttt{tell}(A,B,X)$

Pre(A): $Bel(A,X) \wedge Know(A,Want(B,Know(B,S)))$

Pre(B): $Int(B,Know(B,S))$

where S may be any of $Bel(B,X)$, *or* $\neg(Bel(B,X))$

Post(A): $Know(A,Know(B,Bel(A,X)))$

Post(B): $Know(B,Bel(A,X))$

Completion: $Know(B,Bel(A,X))$

Figure 2.4: KQML Semantics for `tell`

The figure 2.4 is taken from the example in [Labrou et al., 1999]. In the example agent **A** is the speaker and **B** is the hearer. The precondition, **Pre(A)** for **A** must be satisfied if the agent is to utter the `tell` of X. If the agent is to accept and understand the performative, **Pre(B)** must be satisfied. Once uttered the **Post(A)** conditions should hold, and once **B** has accepted and understood the message **Post(B)**. The **Completion** condition is the final state associated with the intention associated with the performative being expressed. The conditions are all expressed in terms of the BDI language, but the formal definition of these terms is not given.

2.2.2 FIPA ACL

The Foundation for Intelligent Physical Agents is a standardisation body concerned with the numerous issues of interoperability in agent-based systems. One of the technical committees of this organisation was charged with the development of a specification for an ACL [FIPA, 2001]. The resulting set of communicative acts is listed in figure 2.5.

Accept Proposal	Agree	Cancel	Call for Proposal
Confirm	Disconfirm	Failure	Inform
Inform If	Inform Ref	Not Understood	Propagate
Propose	Proxy	Query If	Query Ref
Refuse	Reject Proposal	Request	Request When
Request Whenever	Subscribe		

Figure 2.5: FIPA's Communicative Acts

FIPA ACL is very similar to KQML as it is based on speech acts and a BDI-centric view of agency. The syntax of the individual locutions and the formatting of their content also resembles KQML. The specifications provide an English description of each of their locutions or what FIPA calls communicative acts. In addition there is a formal semantics defined in a form of modal logic. The language used is called 'Semantic Language' (SL) [Sadek, 1991].

SL is a multimodal logic with the typical representations of beliefs, desires, and intentions, but it also adds uncertain beliefs as well. Each communicative act is defined in terms of a SL formula for both the act's feasible preconditions and rational effect. The feasible preconditions are the mental states that must exist before an agent *can* send that communicative act. The existence of the mental states does not behoove the agent to take the action, but only provides the conditions that would make the action appropriate if the agent is to be compliant to the standards. Rational effects are the mental states that are expected, given an agent has performed the communicative act. Though expected states are often defined in the rational effects for recipient agents, conformance does not mandate that these conditions hold.

Figure 2.6 defines the `inform`, a communicative act closely akin to KQML's tell from figure 2.4. The notation $B_i\phi$ is understood to mean that an agent i believes the proposition ϕ to be true. The expressions, $Bif_j\phi \vee Uif_j\phi$, are a uniquely SL construc-

$< i, \texttt{inform}(j, \phi) >$

FP : $B_i\phi \wedge \neg B_i(Bif_j\phi \vee Uif_j\phi)$

RE : $B_j\phi$

Figure 2.6: FIPA ACL's SL Semantics for `inform`

tion to represent a shorthand for a more complicated expression which means simply the agent *j* has no beliefs concerning the proposition ϕ. The feasible preconditions of figure 2.6 can therefore be read as saying that agent *i* believes ϕ and does not have any belief that agent *j* has any belief about the proposition. The rational effect simply states that agent *j* is to believe ϕ.

FIPA has superseded KQML as the most prevalent attempt at a standard agent communication language, but serious foundational difficulties remain for any mentalistic approaches. Numerous criticisms of KQML and FIPA ACL exist [Wooldridge, 2000, Labrou and Finin, 1997, Cohen and Levesque, 1995]. The semantics of mentalistic approaches are either undefined, or poorly defined, or simply not computable. There is the problem with being able to verify any mentalistic semantics [Wooldridge, 2000]. The conclusions from these difficulties must be that mentalistic approaches are an inappropriate basis for a communication language in open multiagent systems. This conclusion can be supported by the growth in popularity of other approaches a sampling of which is described in the following section. Nonetheless, FIPA remains a popular basis for agent communication. This is due to a number of secondary factors to its questionable semantic foundations which include: sizable industrial and governmental funding, and the simple truth that semantics are largely ignored (e.g. JADE [Bellifemine et al., 1999]) [Verdicchio and Colombetti, 2003].

2.3 Social and Centralised (Electronic Institutions)

The impetus for LCC and in particular the use of LCC for dynamic protocols comes from work done with Electronic Institutions, especially the work of [Esteva et al., 2000, Noriega, 1997]. The idea was to retain the benefits of EI approaches, but in a more decentralised and flexible manner. Electronic Institutions (EI) provide structure to large and open multi-agent systems (MAS). By emulating human organizations, Electronic Institutions provide a framework which can increase interoperability. Implicit in the EI approach is this idea of an extraneous participant in the conversation such as a mediating or governing institutional agent to enforce the norms specified by the EI. The other Electronic Institutions described here vary in detail but the basic principles are close to the ISLANDER model.

2.3.1 Islander and Ameli

The ISLANDER [Esteva et al., 2002] framework formally defines several aspects of electronic institutions. The core is the formal definition of roles for agents, a shared dialogical framework, the division of the Institution into a number of scenes and a performative structure which dictates, via a set of normative rules, the relationships between the scenes. Agents interact with an Institution through the exchange of illocutions, i.e. messages with intentional force [Noriega, 1997].

Participating agents are required to adopt a role within the Institution. This is similar to our entering a shop and assuming the role of a customer, and the employee adopting the role of salesperson. A role is defined as a finite set of dialogical actions. By the adoption of a role within an Institution, an agent's activities within the Institution can be anticipated. This abstraction of agents as a role allows the Institution to regulate and identify agent activities without analysing individual agents. Relationships between agents can be dealt with as generalizations. A role can also be defined as subsuming or being mutually exclusive to another role.

The dialogical framework provides a standard for communication. Agents are guaranteed to have a shared vocabulary for communication as well as a common worldview with which to represent the world they are discussing. The dialogical framework is defined as a tuple consisting of an ontology, a representation language, a set of illocutions, and a communication language. The representation is an encoding of the knowledge represented by the ontology and makes up the inner language. This is contained within an individual illocution that is passed between agents. The illocution, as part of the outer language or communication language, expresses the intention of the agent by its communicating the message of the inner language. The dialogical framework, which contains the ontological elements, is necessary for the specification of scenes.

All interactions between agents occur within the context of scenes. Scenes are interaction protocols between agent roles. They are expressed as a well-defined protocol which maps out the conversation space between two agent roles. These scenes are represented as graphs. The nodes are conversation states and arcs representing the utterances of illocutions between the participants. Each scene will have a set of entrance and exit states with conditions that must be satisfied before the agent can begin or exit a scene. A set of roles and scene states are formally defined. An element of the set of states will be the initial state and a non-empty subset will be final states. Between the states there is a set of directed and labelled edges.

Scenes are individual agent conversations. In order for agents to participate in more interesting activities, it is necessary to formalize relationships between these individual conversations. The performative structure formalizes this network of scenes and their association with each other. The roles an agent adopts and the actions of the agents create obligations and restrictions upon the agent. These obligations restrict the further movement of agents. The performative structure is made of a finite non-empty set of scenes. There is a finite and non-empty set of transitions between these scenes. There is a root scene and an output scene. Arcs connect the scenes of the Institution. These

arcs have different constraints placed upon them. For example, the constraints can synchronize the participating agents before the arc can be fully traversed, or there are constraints that provide an agent a choice point upon which scene to enter.

Within the scenes of an Electronic Institution, the actions an agent performs affect the future actions available to the agent. These consequences can extend beyond the current scene. Certain actions could be the requirement for an agent to perform an action in some future scene or even dictate which scenes or sequence of scenes an agent is now required to be a participant of. These normative rules are categorized between two types. *Intra-scene* dictate actions for each agent role within a scene, and *inter-scene* are concerned with the commitments which extend beyond a particular scene and into the performative structure [Esteva et al., 2000, Estava et al., 2001]. Figure 2.7 gives an example of an institution designed for the diagnosis of breast cancer and one of the scenes for this institution for determining candidates for specialist referral.

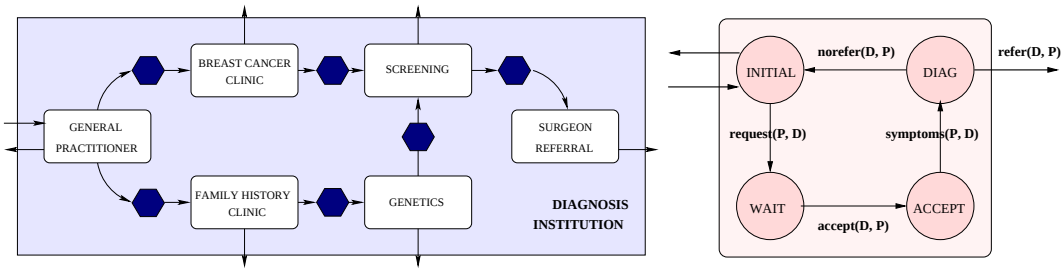

Figure 2.7: *Example Electronic Institution and Scene*

Tools [Esteva et al., 2002] exist to aid in the creation of the various components and development of Electronic Institutions. This includes a tool to verify any specifications developed as well as tools to aid the synthesis of agents that can participate in the Electronic Institution [Vasconcelos, 2002].

Ameli is the infrastructure and governing agent that mediates participating agents' interactions [Esteva et al., 2004]. Islander's focus was the specification of EIs. Ameli's is the execution of them. Ameli's innovative contribution is the ability to implement any electronic institution specification defined in Islander regardless of domain. Given

an Islander EI specification, Ameli ensures that agents participating in the institution adhere to all the norms specified. There is unpublished research in its infancy that is exploring the idea using the LCC language to directly implement and execute EI specifications.

2.3.2 Others

There are several other Electronic Institutional models [Dignum, 2003, Vázquez-Salceda, 2003, López, 2003]. Most of these address ISLANDER's rigidity in representation and enforcement of norms, and are not concerned, in particular, with issues addressed in this thesis. As a result their focus tends to be on the development of computationally hard but expressive logics for the formalisation of norm specifications and addressing the difficulties associated in interpreting and implementing them [Cortés, 2004].

In general, the Electronic Institution approach has a number of difficulties that confront its appropriateness for open heterogeneous multiagent systems. Though EIs are ideal and necessary for systems the require high reliability and predicability, they are not always desirable generally. The primary concern is the requirement of third-party coordination and control. There still remains the questions of how institutions and their rules are disseminated and evaluated by the agents themselves as the normal practice relies on the engineer to endow the agent with sufficient understanding of the institution before the interaction.

2.4 Social Agent-centric Approaches

This section describes a number of related models of agent communication. Different researchers have come up with a number of ideas that could be categorized as social approaches such as conversation policies, obligation, commitment, norms, landmarks, etc. They are related to Electronic Institutions described previously because an agent is

seen as a participant in a system where communication is its primary means to achieve its goal and like natural occurring social systems there are conventions that facilitate the activity in the system to run more smoothly. The distinction is that in electronic institutions these conventions come top down. The social approaches described in this section take a more agent-centric or bottom up view. Agents are armed with social models that allow for a freer exploration of the conversation space to address the dynamism inherit in complex systems, to organise themselves, and to be able to consider characteristics of the system unforseen at design time.

In [Singh, 2000], the author identifies several desirable criteria for the semantics of an Agent Communication Language. According to Singh, an ACL should be formal, declarative, verifiable, and meaningful. To this end, he has developed a *social semantics*. He defines three facets to every communicative act. The *objective claim* which commits an agent to another that some proposition *p* holds. The *subjective claim* is that an agent believes *p*, and the *practical claim* that the agent has some justification or reason for believing *p*. This is a novel approach, because most reactions to the semantic verification problem of the mentalistic approach is to completely throw it away. Singh has, instead, embraced the mentalistic approach but coupled it with the idea of social commitment. The purely mentalistic approach rests on the assumption that the agent is sincere about *p*, but Singh has added that the agent is also socially committed to being sincere about *p*. It is recognized that the use of social semantics does not replace the need for protocols, but the combination of social semantics and protocols would create a much more flexible ACL [Maudet and Chaib-draa, 2002].

The approach described in [Flores and Kremer, 2002] uses the commitment themselves to develop the conversation between two agents. Flores argues that our verbal utterances carry with them obligations dependent on the role of the agent within a society. The question 'What time is it?' carries with it the obligation (in polite society) to not only reply but make an attempt to actually find out the time. The use of social commitments in multi-agent communication is to provide a number of rules

Utterance	Operation	Creditor	Debtor
Request	Add	Speaker	Addressee
Offer	Add	Addressee	Speaker
Release	Del	Speaker	Addressee
Discharge	Del	Addressee	Speaker

Figure 2.8: Commitment Revision ACL

that dictate appropriate illocutions and actions performed based on the agent voluntarily obligating itself to commitments with other agents and eventually discharging those commitments. A protocol is defined for the negotiation of the adoption of social commitments. The locutions involved for this commitment store revision is shown in figure 2.8. Agents propose to add and remove commitments for action from personal commitment stores. An agent will propose to add a commitment to perform some actions. Once this is accepted and the commitment is satisfied the protocol includes steps to propose the release of any further commitment to that action. It is through this simple protocol and the social commitment-based conversation policies an agent conversation can be developed.

The agent-centric design of these social models for communication are excellent candidates to drive dynamic protocols, as the models would also gain a means for dissemination. For a particular domain it is possible to develop a normative social model to govern whole conversations, but social approaches lack a generic model to do this. Instead they provide the possibility of more local normative constraints. The dialects problem also exists as there has been no standardisation effort like the mentalistic FIPA ACL or any one approach that has gained dominant support. Instead this thesis has chosen to concentrate on rationalist approaches, but their use does not exclude the use of social approaches to further embellish conversation semantics.

2.5 Rationalistic Approaches

The rationalistic approaches to communication can be cleaved, though not cleanly, into two disciplines. Some researchers concentrate on mechanism design [Dash et al., 2003]. It is their goal to produce protocols that can guarantee certain properties such as honesty or profit maximisation. Game theoretic approaches tend to have impractical assumptions for use in agent systems generally such as agents having complete knowledge, participants always acting rationally, or engineers having an *a priori* understanding of the interaction. This field tends to view protocols as rigid, and therefore not very fertile ground for the run-time adaptation of protocols that is our goal.

The other half is the use of argumentation for multi-agent systems. Argumentation in general is a powerful model for agent communication as it allows the discussion of the topic, but also the reasoning and other meta information involved. This allows agents to address knowledge deficiencies of others, correct false beliefs, and influence an agent's utilities and preferences for beliefs. This functionality can potentially result in agents converging on a solution quicker as they can identify the cause of the disagreement rather than relying on a brute force iteration through all permutations of the solution space. This also can be achieve by agent's tailoring their own or other's space of acceptability. These advantages come with increased complexity, but it is hoped through the adoption of dialogue games as distributed protocols, but also through dynamic protocols no loss of flexibility or expressivity occurs.

Argumentation has a long and rich history [Aristotle, 1997, Gautama, 2003]. Through out this long history philosophers have identified a number of modes of argumentation. In one such enumeration [Gilbert, 1994] at least four is given. *Emotional* argumentation appeals to emotions and depends on empathy. *Visceral* arguments require the physical or social aspects of humanity. Bosses and parents can make visceral arguments because of their social position with respect to their employees or children. Similarly, the school yard bully can make a punch in the arm a convincing visceral justification for his proposition for a child to give him his lunch money. *Kisceral* argu-

mentation relies upon the supernatural or religious. The *logical* mode of argumentation is held is the highest esteem. Especially in the occidental secular tradition, the quality of these arguments are generally thought to be better than other modes. They have the feel of inevitability and necessity. One part of the argument is deduced from the other. This is the mode of argumentation that is of most interest especially within the context of multiagent systems.

In the classical view of logic, there is a set of statements or propositions, Δ, that allow one to infer an additional proposition φ, written as $\Delta \vdash \varphi$. To arrive at φ it is not necessary to appeal to all the propositions of Δ. Instead a subset, Γ is used by means of various axioms to arrive at the conclusion. Γ is said to be the grounds for making the argument of φ.

Argumentation in multiagent systems occurs when the possibly inconsistent database of propositions is dispersed between agents. The agents trade arguments in an attempt to defeat other arguments. Defeat can be defined as either *rebut* or *undercut* depending on whether the argument contradicts the conclusion (i.e. φ) or the grounds (i.e. Γ) of the other argument. As there may be many possible ways to attack or defeat an argument, there is usually some type of preference ordering either by a numerical utility function or a more abstract valuation (e.g. argument by appealing to authority is considered weaker than appealing to counter examples). Argumentation research is much more broad than the discussion here presents. There is abstract argumentation which ignores the propositions and is only concerned with the topology of the argument structure and the relationships between arguments [Dung, 1995, Vreeswijk and Prakken, 2000]. There have even been excursions into the analysis of the tetralemma for use in agent systems. Tetralemma is an alterative four valued logical system. The values are (true, false, both true and false, neither true nor false) whereas the western tradition uses the more prosaic two valued system of either true or false.

Instead, dialogue games is concerned with the process of two agents fir-

ing argument salvos at one another. Several authors have developed typologies and protocols for this kind of communication and it is this work that complements our investigation into dynamic interaction protocols for use in multiagent systems [Amgoud et al., 2000, Lebbink et al., 2003, Maudet and Chaib-draa, 2002, Maudet and Evrard, 1998, Dastani, 2001, Parsons et al., 2003b, Parsons et al., 2003a, Parsons et al., 2002, Parsons, 2002, Parsons et al., 2004, McBurney and Parsons, 2004, McBurney and Parsons, 2003, McBurney et al., 2002a, McBurney et al., 2002b, McBurney and Parsons, 2002, McBurney and Parsons, 2001].

2.5.1 Dialogue Games

The philosophers Doug Walton and Erik Krabbe have developed a typology of dialogues to detect fallacious reasoning [Walton and Krabbe, 1995]. This typology was adopted by Chris Reed [Reed, 1998] in a formalism for multi-agent systems and inter-agent communication. Of the six kinds of dialogue identified, five dialogue types are applicable to the domain of agent communication. The sixth, eristic, is a dialogue where reasoning has ceased and the participants use the dialogue for the airing of grievances and one-upmanship. This dialogue type is important for the study of human conversations, but it is ignored by the agent research community.

Dialogues are classified into the different types by three criteria. The first criterion considers the initial situation. What information does each of the participants have? Are the agents cooperative or competitive with each other? The second criterion concerns the individual goals an agent has for the interaction, and the third criterion are the goals shared by the participating agents. In *Information-Seeking* dialogues, one agent seeks the answer to a question which it believes the other agent possesses. *Inquiry* dialogues occur when two agents work together to find the answer to a question whose solution eludes both agents. A *Persuasion* dialogue has one agent attempting to convince another to adopt some proposition which it currently does not believe. *Ne-*

gotiation dialogues occur when the participants haggle over the division of a scarce resource. In *Deliberation* dialogues, the agents attempt to agree on a course of action for a particular situation. It is rare that any actual dialogue will be purely of one instance of one kind of dialogue. It is more likely that a dialogue will consist of an amalgamation of the different types. For example, during a negotiation, propositions may need clarification and an information-seeking dialogue would occur. This dialogue typology is fundamental to recent agent communicative models using dialogue games.

Dialectics have been of interest to philosophers as a tool to formalise argumentation for millennia [Aristotle, 1997]. It is an attempt to identify when an argument or its justification is weakened or undercut by an argument or refutation made be the other participant. By each player making 'moves' and following a set of rules, it was hoped that properties of good and bad arguments could be identified. This formalism for argumentation has been employed to increase the complexity and robustness of software agents conversations. The objective is to produce a meaningful interaction between dialogical partners by following the rules of an individual dialogue game.

There are several components to a dialogue game. Firstly, the participants must share a set of locutions. This is a common requirement for models of agent communication. The commencement and termination rules specify the conditions under which a dialogue can start or end. This is a set of performatives from an agent communication language that is shared between the agents. This language must include the ability to utter assertions as well as justifications and challenges to those assertions. Another component are the combination rules. These rules define when particular illocutions are permitted, required, or illegal. The last part necessary for a dialogue game are the rules for commitment. These rules create obligations on the agent with respect to the dialogical moves of the agent. These commitments can be divided into dialogical and semantic. Dialogical commitments are the obligation of an agent to make a particular move within the context of the dialogue game. Semantic commitments indenture the

agent to an action beyond the dialogue game itself. A record of these commitments is publicly stored. For example, if you say you are willing to pay the highest price in an auction, it will be known that you are committed to actually pay that price.

The dialogue game frameworks [McBurney and Parsons, 2002] and [Maudet and Evrard, 1998] attempt to construct more complex and robust agent conversations. This is achieved by combining different atomic dialogue types which have been identified by philosophers analysing human dialogues [Walton and Krabbe, 1995]. This approach avoids the semantic ambiguities inherent in mentalistic models and the rigidity of static protocol-based approaches [FIPA, 2001]. The dialogue game approach depends on several assumptions about participating agents. Agents participating in the dialogue game framework must agree on all the rules of the framework. The number of requirements made on individual agents in order for them to play dialogue games makes the approach unsuited for open multi-agent systems.

Type	**Initial State**	**Goal**	**Aim**
Persuasion	Opinions conflict	Resolve opinions	Persuade other
Negotiation	Interests conflict	Make a deal	Get the best deal
Inquiry	General ignorance	Gain knowledge	Find a proof
Deliberation	Need for action	Reach a decision	Influence outcome
Information Seeking	Personal ignorance	Spread knowledge	Gain or pass knowledge

Figure 2.9: Typology of Dialogues

2.6 Summary and Analysis

Outlined above are the major approaches to agent communication and the building blocks for this thesis' contribution to the topic. A bullet point summation is shown in figure 2.10. Electronic Institutions have their usefulness for rigid multiagent systems where autonomy of the agents is willingly sacrificed to the systems as a whole. Distributed protocols is the approach taken to achieve the transparent protocol transformation, but the state of the art has not been fully explored. No other research has developed the idea to the extent of LCC and coordination oriented programming. For dynamic protocols to succeed there must be some model of transformation to ensure meaningful adaptations. Of the approaches commitment models and argumentation is the most promising. The emphasis of this document will be the adoption of dialogue games for our purposes.

The next chapter, chapter 3, describes the protocol language and the supporting framework that will be the basis for this thesis' contribution that can go some measure toward addressing the gaps left by the technologies and models just described.

Approach	Advantages	Shortcomings
Institutional	reliable formally defined	dependant on centralized control static FSM representations dissemination problem
Distributed Protocols	modular novel approach	static inchoate development closed systems
Mentalistic	widely used agent-centric	used naively unverifiable requirement heavy too abstract
Social	agent-centric flexible	limited scope requirement heavy
Rational	flexible agent-centric robust	too theoretical requirement heavy

Figure 2.10: Summary Table of Communication Models

Chapter 3

The Lightweight Coordination Calculus(LCC)

The development of LCC came from dissatisfaction with the Electronic Institutions [Walton and Robertson, 2002] model for communication, especially the ISLANDER approach [Esteva et al., 2002]. Although the EI framework provides structure and stability to an agent system, it comes at a cost. Integral to EI is the notion of the administrative agents. Their task is to enforce the conventions of the Institution and shepherd the participating agents. Messages sent by agents are sent through the EI. This synchronises the conversation between the conversing agents, and keeps the administrative agent informed of the state of the interaction. This centralisation of control runs counter to the agent paradigm of distributed processing. This initial idea has been expanded to a more general one of coordination oriented programming. Although, our focus and discussion will be restricted to agents and multiagent systems, the archetype is applicable to processing in open distributed systems in general and in particular semantic web and grid applications.

By coding from the perspective of the interaction between agents rather than from the more traditional agent-centric view, a unique and useful programming model is created. It allows engineers to address some difficult issues associated with coordi-

nating communicating agents independently of the internal construction of the agents themselves.

Context is important in communication. The exact same word uttered has completely different meaning given a change in context. The utterance of ‘fire’ has drastically different consequence compared for the blindfolded man against a wall to the travelling companion entering a pub on a cold night. Context can be understood not only in the abstract sense (i.e. the context of an auction versus an informal discussion), but also in terms of the context of an instance of an interaction. During an auction, the communication of a bid is only appropriate in the context of the bidding still being open. Both senses of context must be understood for an agent to participate successfully in an interaction.

The creation or satisfaction of commitments inherent to the interaction that can not be left to the caprices of the agents themselves must also be considered. For example, it might be important to force an agent to satisfy some meta-dialogical requirement in order, or as a consequence, of communicating a message. This is common in auctions where the communication of “I bid X” commits you to paying ‘X’.

Coordination oriented programming can alleviate the ambiguity of the locutions exchanged between messages. By introducing conditions upon the occurrence of messages during the interaction the meaning of the messages can be clarified. If an agent is new to the interaction, this approach also provides a means for introducing agents to the minimal participation requirements of the multiagent system. This is done with a minimal engineering overhead for the newly arrived agent. These considerations are less of a problem when the multiagent system can be guaranteed to never change, and all the participating agents are developed by the same engineer. These assumptions become impractical for the large, complex and open multiagent systems which are of increasing interest.

Coordination oriented programming is a declarative specification of an interaction that is executable independent of the design of specific agents. This program, when

executed, coordinates the interaction between agents. By a peer to peer oriented implementation, it avoids the need for third-party agents to coordinate the dialogue. The language developed for this purpose is called LCC, lightweight coordination calculus.

In this chapter, the fundamentals of LCC are explained. This work has been developed in a number of papers [Robertson, 2004c, Robertson, 2004a, Robertson, 2004b, Walton and Robertson, 2002, Walton, 2004a, Walton and Barker, 2004, Walton, 2004b]. There are a number of researchers who have further extended LCC and its framework to address a diverse number of problems in agent communication [Hassan et al., 2005, Lambert and Robertson, 2005, McGinnis and Robertson, 2004a, McGinnis et al., 2003, McGinnis and Robertson, 2004b, Besana et al., 2005, Osman et al., 2006, Grando and Walton, 2006]. This language and framework will be the basis for the later chapters of the thesis and my own enhancements.

3.1 Syntax

Figure 3.1 defines the syntax of a protocol language taken from [Robertson, 2004c] which also gives a fuller explanation of the language and framework. The protocol consists of a set of agent clauses, $A^{\{n\}}$. The protocol will contain a set of at least two clauses because LCC protocols are defined locally (i.e. from the perspective of the participating agent roles). An agent clause is the series of communicative actions expected to be performed by an agent adopting the role defined by the agent definition. These clauses make the agent definition consisting of a role (R) and unique identifier (Id). A role is defined in a similar way as Electronic Institutions: it is a way of defining communicative activity for a group of agents rather than individuals, but an important distinction must be made. Electronic Institutions rely largely on a finite state representation of the interaction protocol. The roles act as a bounding box for a set of states and transitions. LCC is based on a process calculus and is therefore well suited to express

$\mathcal{P}$	$\in$	Protocol	::=	$\langle S,A^{\{n\}},K\rangle$	
A	$\in$	Agent Clause	::=	$\theta :: op.$	
θ	$\in$	Agent Definition	::=	**agent(**R,Id**)**	
op	$\in$	Operation	::=	no op	
			|	θ	
			|	**(**op**)**	(Precedence)
			|	$M \Rightarrow \theta$	(Send)
			|	$M \Leftarrow \theta$	(Receive)
			|	*op1* **then** *op2*	(Sequence)
			|	*op1* **par** *op2*	(Parallelization)
			|	*op1* **or** *op2*	(Choice)
			|	$(M \Rightarrow \theta) \leftarrow \psi$	(Prerequisite)
			|	$\psi \leftarrow (M \Leftarrow \theta)$	(Consequence)
M	$\in$	message	::=	$\langle m,\mathcal{P}\rangle$	

Figure 3.1: An Abstract Syntax of the Protocol Language

the concurrency found in multiagent systems.

The agent definition is expanded by a number of operations. Operations can be classified in three ways: actions, control flow, and conditionals. Actions are the sending or receiving of messages, a no op, or the adoption of a role. Control Flow operations temporally order the individual actions. Actions can be put in sequence (one action must occur before the other), performed simultaneously (actions must both be completed without regard to order) or given a choice point (one and only one action should occur before any further action). The definition of the double arrows, '$\Rightarrow$' and '$\Leftarrow$', denote messages, M, being sent and received. On the left-hand side of the double arrow is the message and on the right-hand side is the other agent involved in the interaction.

Constraints can fortify or clarify semantics of the protocols. Those occurring on the left of the '$\leftarrow$' are postconditions and those occurring on the right are preconditions. The symbol ψ represents a first order proposition. For example, an agent receiving a protocol with the constraint to believe a proposition s upon being informed of s can infer that the agent sending the protocol has a particular semantic interpretation of the act of informing other agents of propositions. This operation, $(M \Rightarrow \theta) \leftarrow \psi$, is understood to mean that message M is being sent to the agent defined as θ on the condition that ψ is satisfiable. This operation, $\psi \leftarrow (M \Leftarrow \theta)$, means that once the message (M) is received from agent θ, ψ holds.

3.2 Expansion Engine and Framework

A message is defined as the tuple, $\langle m,\mathcal{P}\rangle$. Where m is the message an agent is currently communicating, and $\mathcal{P}$ is the protocol written using the language described in figure 3.1. The protocol, in turn, is a triple, $\langle S,A^{\{n\}},K\rangle$. S is the dialogue state. This is a record of the path of the dialogue through the conversation space and the current state of the dialogue for the agents. This set of agent clauses is marked to show the progress

of the dialogue and the current state of the interaction. The messages are marked as closed or failed depending on whether they are communicated successfully. Messages which have been communicated are encased by a 'c', *c(M)*.

An operation is decided to be closed, meaning that it has been covered by the preceding interaction, as defined in the following function:

$$
\begin{aligned}
&closed(c(X)) \\
&closed(A \,\mathbf{or}\, B) \leftarrow closed(A) \vee closed(B) \\
&closed(A \,\mathbf{then}\, B) \leftarrow closed(A) \wedge closed(B) \\
&closed(X \mathbf{::=} D) \leftarrow closed(D)
\end{aligned}
$$

The second part is a set of agent clauses, $A^{\{n\}}$, necessary for the dialogue. The protocol also includes a set of axioms, K, consisting of common knowledge to be publicly known between the participants. This explicit communication of the dialogue state provides a means of coordination. It is possible to create an agent which retains no internal record of the state of the dialogue but rather uses the communicated dialogue state as a book mark for which to hold its place and remind it of the next communicative step it can take.

Figure 3.2 describes rules for expanding the received protocols. An agent receives a message of the form specified in figure 3.1. The message is added to the set of messages, M_i, currently being considered by the agent. The agent takes the clause, C_i, from the set of agent clauses received as part of $\mathcal{P}$. This clause provides the agent with its role in the dialogue. The agent then expands C_i by the application of the rules in figure 3.2. The expansion is done with respect to the different operators encountered in the protocol and the response to M_i. The result is a new dialogue state, C_n; a set of output messages, O_n and a subset of M_i, which is the remaining messages to be considered, M_n. The result is arrived at by applying the rewrite rules. For example, expansion rule 1 of figure 3.2 describes the action of the agent adopting a role. The operations defined as part of that role can be expanded (i.e closed). Rules 2 and 3

$$1)\quad A\texttt{::=}B \xrightarrow{M_i,M_o,\mathcal{P},O} A\texttt{::=}E$$
$$if\quad B \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

$$2)\quad A_1\ \texttt{or}\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$
$$if\quad \neg closed(A_2) \wedge A_1 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

$$3)\quad A_1\ \texttt{or}\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$
$$if\quad \neg closed(A_1) \wedge A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

$$4)\quad A_1\ \texttt{then}\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E\ \texttt{then}\ A_2$$
$$if\quad A_1 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

$$5)\quad A_1\ \texttt{then}\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} A_1\ \texttt{then}\ E$$
$$if\quad closed(A_1) \wedge A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

$$6)\quad C \leftarrow M \Leftarrow A \xrightarrow{M_i,M_i-\{M\Leftarrow A\},\mathcal{P},\emptyset} c(M \Leftarrow A)$$
$$if\quad (M \Leftarrow A) \in M_i \wedge satisfy(C)$$

$$7)\quad M \Rightarrow A \leftarrow C \xrightarrow{M_i,M_o,\mathcal{P},\{M\Rightarrow A\}} c(M \Rightarrow A)$$
$$if\quad satisfied(C)$$

$$8)\quad null \leftarrow C \xrightarrow{M_i,M_o,\mathcal{P},\emptyset} c(null)$$
$$if\quad satisfied(C)$$

$$9)\quad agent(r,id) \leftarrow C \xrightarrow{M_i,M_o,\mathcal{P},\emptyset} a(R,I)\texttt{::=}B$$
$$if\quad clause(\mathcal{P}, a(R,I)\texttt{::=}B) \wedge satisfied(C)$$

Figure 3.2: Rules for Expanding an Agent Clause

concern the **or** operation. An **or** operation can be expanded if one half can be expanded and the other is not closed. Rule 4 states the first half of the **then** operation can be expanded, but rule 5 requires that the second half is only expanded after the first half has been successfully closed.

Rules 6, 7, 8, and 9 concern the satisfaction of constraints. Satisfied is meant that a state defined by the constraint also exists in the agent. *satisfied(C)* is true if C can be solved from the agent's current state of knowledge. *satisfy(C)* is true if the agent's state of knowledge can be made such that C is satisfied. The bottom of figure 3.2 should read that $clause(\mathcal{P},X)$ is true if clause X appears in the dialogue framework of protocol $\mathcal{P}$, as defined in figure 3.1. The sequence would be similar to figure 3.3. C_n is then sent as part of $\mathcal{P}$ which will accompany the sending of each message in O_n.

$$\langle C_i \xrightarrow{M_i,M_{i+1},\mathcal{P},O_i} C_{i+1},\ldots,C_{n-1} \xrightarrow{M_{n-1},M_n,\mathcal{P},O_n} C_n\rangle$$

Figure 3.3: Sequence of Rewrites

Figures 3.4 and 3.5 give a visualisation of the LCC approach (figure 3.5) to agent communication compared to the more orthodox approach typified by Electronic Institutions(figure 3.4).

The standard view expects individual agent engineers to interpret formal definitions and specifications to design their agents with. The agents then communicate messages usually via third party agent that maintains the global state of the dialogue and ensures norms are adhered to. In contrast, the LCC approach has the protocol designed separate from the agent and the agents themselves coordinate the dialogue by the explicit communication of the protocols and the dialogue state.

The **par** operator is not implemented by the expansion engine, because of the possible problems that could occur in multiparty dialogues. Since the dialogue state is sent with the message, there is the possibility of deadlock or state inconsistency. If the

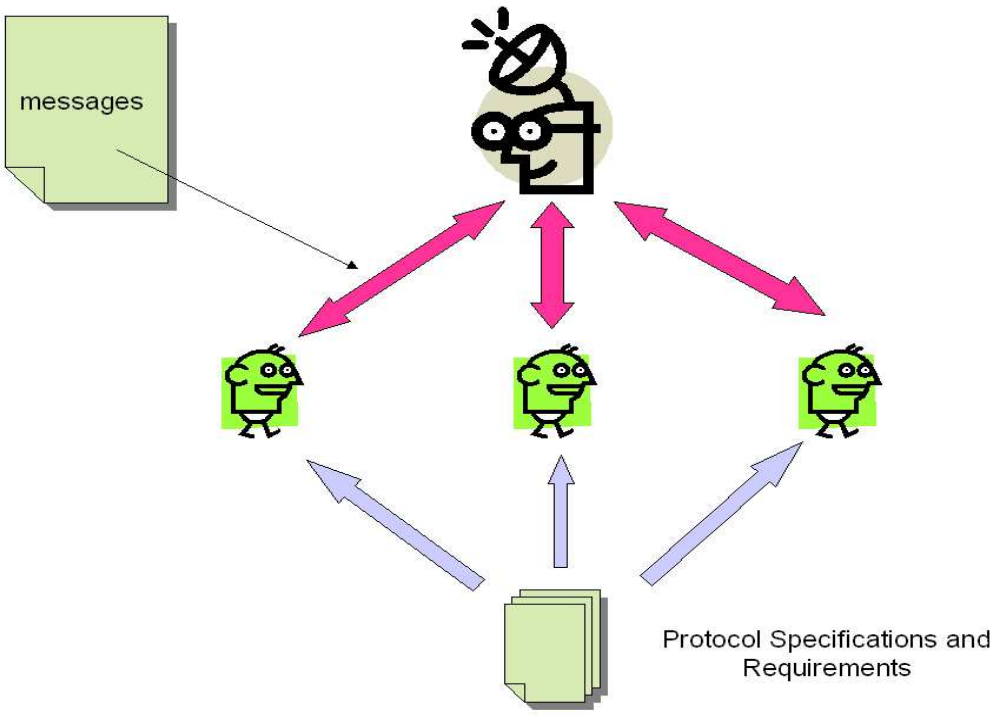

Figure 3.4: *Figure of the Standard View of Protocol led Multiagent System*

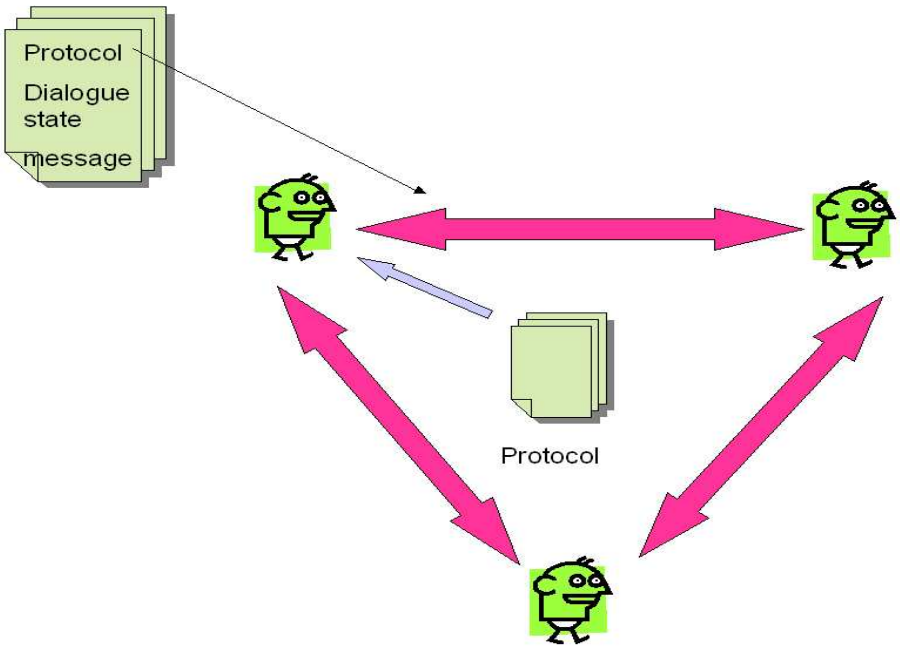

Figure 3.5: *Figure of the Distributed Protocol Multiagent System*

agent receives to differing dialogue states for the same protocol, there is the possibly intractable problem of how to reconcile the two versions of the dialogue state. This is a consequence of the necessity to communicate the dialogue state to be able to adapt them and achieve dynamic protocols. A variation of the LCC language and framework, MAP [Walton, 2004b] does not maintain the dialogue state and avoids this difficulty, but makes it unsuitable for run time protocol adaptations.

3.3 Agent Engineering Requirements

Agents themselves communicate the conventions of the dialogue. This is accomplished by the participating agents satisfying two simple engineering requirements. Agents are required to share a dialogical framework. This an unavoidable necessity in any meaningful agent communication. This includes the requirements on the individ-

ual messages and constraints[1] are expressed in a ontology understood by the agents. The issue of ontology mapping is still open, but the ideas and algorithm developed in this thesis have proven to be useful for research into run-time ontology reconciliation [Besana et al., 2005].

The other requirement obligates the agent to provide a means to interpret the received message and its protocol. The agent must be able to unpack a received protocol, find the appropriate actions it may take, and update the dialogue state to reflect any actions it chooses to perform. The approach taken here is the expansion engine described previously in section 3.2.

3.4 An Example

We will take a simple example to illustrate LCC and the framework being used. The conversation space can follow any one of these sequences. These are all the possible permutations of message sequences for the interaction for the role being described.

$$\textbf{For } a(r1,a1) \ : \left\{ \begin{array}{l} \langle m1 \ \Rightarrow \ a(r2,a2),\ m3 \ \Leftarrow \ a(r2,a2)\rangle, \\ \langle m2 \ \Rightarrow \ a(r2,a2),\ m4 \ \Leftarrow \ a(r2,a2)\rangle, \\ \langle m1 \ \Rightarrow \ a(r2,a2),\ m4 \ \Leftarrow \ a(r2,a2)\rangle, \\ \langle m2 \ \Rightarrow \ a(r2,a2),\ m3 \ \Leftarrow \ a(r2,a2)\rangle \end{array} \right\}$$

The agent identified as $a1$ playing the role of $r1$ can either start the dialogue by the communication of the message $m1$ or $m2$. Afterwards the dialogical partner, $a2$, having adopted the role, $r2$, can reply by either sending the message $m3$ or $m4$. It is easy to imagine how quickly the conversation space between agent can expand to an unmanageable size.

This conversation space can be defined in LCC as the agent clauses shown in figure 3.6. This defines the steps of the protocol that are necessary for the agent playing the role identified in the agent definition (i.e. left-hand side of the '`:::=`'). It is easy to

[1] The specification of the constraint must be understood by the agent, but how it is satisfied is left to the internal reasoning of the agent

see that the agent $a1$ can satisfactorily explore the conversation space by following the agent clause. The same is true for its dialogical partner following the protocol for the role of $r2$.

$$
\begin{array}{l}
a(r1,a1)\texttt{::=} \\
(m1 \Rightarrow a(r2,a2)\ \texttt{or}\ m2 \Rightarrow a(r2,a2))\ \texttt{then} \\
\quad (m3 \Leftarrow a(r2,a2)\ \texttt{or}\ m4 \Leftarrow a(r2,a2)).
\end{array}
$$

$$
\begin{array}{l}
a(r2,a2)\texttt{::=} \\
(m1 \Leftarrow a(r1,a1)\ \texttt{or}\ m2 \Leftarrow a(r1,a1))\ \texttt{then} \\
\quad (m3 \Rightarrow a(r1,a1)\ \texttt{or}\ m4 \Rightarrow a(r1,a1)).
\end{array}
$$

Figure 3.6: Conversation Space as LCC Agent Clauses

Agent $a1$ decides to initiate a dialogue. The protocol is expanded by the rules in figure 3.2. Since this agent is identified by $a1$ playing the role of $r1$, it can satisfy the first expansion rule (e.g. A $\texttt{::=}$ B $\xrightarrow{M_i,M_o,\mathcal{P},O}$ A $\texttt{::=}$ E if B $\xrightarrow{M_i,M_o,\mathcal{P},O}$ E). Expansion can now continue but the agent must first decide which of the two operations of the **or** to perform. Agent $a1$ for one reason or the other chooses to send message $m1$. This particular protocol has not put any explicit constraints to be satisfied on the exchange of messages. Instead it is left to the discretion of the agents to choose between the actions allowed by the protocol. Once the agent makes its choice, it expands its appropriate clause and updates the dialogue state. This is done in accordance with the expansion rules defined by figure 3.2. The expansion can go no further for the agent, because it cannot satisfy the either half of the second **or**, because neither message has been received yet. At the very top of figure 3.7 reflects the closed $m1$ message in $a1$ dialogue clause. There is no dialogue state for $a2$'s agent clause as the agent has yet to take any action.

$a(r1,a1)$ **::=**
$c(m1 \Rightarrow a(r2,a2))$ **then**
$(m3 \Leftarrow a(r2,a2)$ **or** $m4 \Leftarrow a(r2,a2))$.
$\Downarrow$ $\Downarrow$
$a(r1,a1)$ **::=** $a(r2,a2)$ **::=**
$c(m1 \Rightarrow a(r2,a2))$ **then** $c(m1 \Leftarrow a(r1,a1))$ **then**
$(m3 \Leftarrow a(r2,a2)$ **or** $m4 \Leftarrow a(r2,a2))$. $c(m4 \Rightarrow a(r1,a1))$.
$\Downarrow$ $\Downarrow$
$a(r1,a1)$ **::=** $a(r2,a2)$ **::=**
$c(m1 \Rightarrow a(r2,a2))$ **then** $c(m1 \Leftarrow a(r1,a1))$ **then**
$c(m4 \Leftarrow a(r2,a2))$. $c(m4 \Rightarrow a(r1,a1))$.

Figure 3.7: Expansion of Dialogue State of the Agents

The agent $a2$ receives the protocol's agent clauses, the dialogue state showing agent $a1$'s action in the dialogue, and the message $m1$. Upon receiving this, the agent expands the protocol and updates it dialogue state to reflect the reception of the message from $a1$. Following the expansion rules the agent closes that part of its clause in the dialogue state. Agent $a2$ decides to send $m4$, and once again the expansion engine does its work and the outgoing message, the agent clauses, and their dialogue states are sent back to agent $a1$. This is reflected by figure 3.7 after the first $\Downarrow$. The final state is achieved when agent $a1$ receives $m4$ and updates its clause and ends the dialogue.

3.5 Chapter Summary

The use of LCC removes the reliance on centralised agents for synchronisation of individual participants in the system, provides a means for dissemination of the interaction protocol and to separate the interaction protocol from the agent's rationalisations to

allow the dynamic construction of protocols during the interaction. The agents are engineered separately from the agents that use them there by decreasing the dependency between the two. By defining interaction protocols during run-time, agents are able to interact in systems where it is impossible or impractical to define the protocol beforehand. For example, negotiation dialogues where the domain of negotiation is not fixed or unknown. Another example would be diagnosis dialogues where the course of the dialogue is determined by the information sent and not a fixed sequence of messages.

A similar approach to LCC is MAP [Walton, 2004b]. MAP is more orientated toward web services, but this difference is trivial for the issues being addressed. They differ in their approaches to synchronisation. Both frameworks distribute protocols but for MAP this is done only at required times (e.g. initiation of the dialogue). It does not maintain or communicate the dialogue state during the interaction. A consequence of this difference allows MAP to easily parallelize and multi-cast interactions. LCC is capable of producing protocols with more than two participants but it is limited to interactions that can be serialized. However, MAP can have any number of distinct parallel interactions.

Other issues of agent communication must also be assumed such as issues of trust, agent discovery, and timing. Trust in agency is a large concern for agency and it is recognized that the use of distributed protocols emphasises this issue as agents themselves control the social norms and do not depend on a third party arbiter. How the agents find their communicative partner and determine the suitability of the protocol is also assumed and the implementation described relies on a bootstrapping mechanism which directs the agent to others. LCC assumes asynchronous communication and does not directly address potential real time demands on messages. As they are currently described the protocols are static and brittle and unforgiving to failure regardless of the triviality of the fault. This is currently being addressed within constraint satisfaction relaxation [Hassan et al., 2005] and with dynamic protocols as described in the next chapters of this thesis.

Despite these relatively innocuous limitations, this protocol language is well suited for our purposes. By distributing the protocol during the interaction, the agents have providence over the interaction protocol allowing agents to make transformations. The explicit transmission of the dialogue state records and communicates the choices made as the protocol is realised. It also catalogues the transformations made and the resulting properties which now hold because of those changes. This allows the mechanism for the flexible protocolled approach we seek. With the machinery to make transformations possible, it is important to ensure they are controlled and meaningful. The following chapters detail two different possibilities to realize this. In addition we have the advantage of an explicit, computational, and executable representation of the social norms (as a protocol) the agents communicating will employ. If the agent can parse the protocols, satisfy the constraints, and update the dialogue state, they will be able to communicate. This includes the dynamic protocols explored in the later chapters of this thesis.

Chapter 4

Realising Dynamic Protocols with LCC

The ability to modify the interaction protocol being used by conversing agents requires modifications to the LCC framework. There must be a point of contact between the expansion engine and the decision making procedures of the agent to allow the agent to evaluate the protocol, including the dialogue state, and determine if an adaption to the protocol is warranted. If so the transformation is made by sending the protocol and the adaption to the predicate performing the transformation. The framework has been extended to include a transformation predicate to ensure all transformations preserve the symmetry [1] of the protocols. This extension follows the modular spirit of the original framework and does not modify it. The expansion engine performs exactly the same and the protocol language remains unchanged. Protocols are passed from the framework to the transformation engine and a protocol is returned. If an adaptation is unnecessary the agent returns the original protocol unadulterated. If a change has been made, no special consideration must be communicated to the framework and the expansion engine performs as normal on the modified protocols. The point of contact exists after the agent receives the protocol from a dialogical partner, and right before

[1] The agent clauses must have symmetry, because protocols in LCC are described locally (i.e. defined in terms of the individual participants rather than from a global view of the interaction). For each message sent, there must be a step in another agent clause that receives the message. No operator in an agent clause can be defined unless there is another clause that compliments it. Otherwise, deadlock or failure occurs. Hence the importance of preserving symmetry when making adaptations to the agent clauses of the protocol

the transmission as these points reflect a change in the dialogue state and thus the possibility that the remainder of the protocol no longer addresses the needs arisen from that new state. Chapter 6 details the specific changes in the code of the framework.

This chapter will concentrate on two models of communication for use with the LCC framework to realize dynamic protocols: BDI and dialogue games. The use of BDI logics, typified by the FIPA ACL, is a popular model for creating an agent centric model of communication. Although there are numerable and some insurmountable difficulties associated with its use, nevertheless an implementation as a distributed protocols addresses some. Section 4.1 lists some of the problems identified by the research literature. Further extensions to LCC with BDI are discussed but ultimately abandoned. Instead the later part of the chapter focuses on the use of dialogue games from argumentation theory as a preferable model within which to base our adaptable protocols.

4.1 Communicating FIPA Semantics

There have been a number of systems developed using the FIPA ACL such as [Bellifemine et al., 1999], but the existence of those FIPA-compliant systems fails to prove the ACL is appropriate as a standard for all agent communication. There are a number of fundamental problems with the model for agent communication which impede its adoption as the standard for which it was designed. The proposal here is a much more direct one: to provide a means for agents to communicate their semantics for conversation by the act of conversing. This is applicable for any model, but this discussion will use FIPA to illustrate its points as it is a well known, although troubled, agent communication language. An implementation of FIPA communicative actions as LCC dynamic protocols solves some of the difficulties and illustrates the advantages of this thesis' approach. These difficulties and the solution described here are applicable to agent communication in general.

4.1.1 Problems of FIPA

FIPA ACL's fate seems to be that of the "straw man" of agent communication. There is a great litany of research papers that use the failures of the ACL's design as a counter example to the ideas each author is purporting.

The criticism expressed here are largely taken from some of the seminal critiques of the FIPA ACL specifications [Wooldridge, 2000, Pitt and Mamdani, 1999, Mayfield et al., 1995, Labrou and Finin, 1997]. The hope is to avoid the more niggling complaints that also exist in abundance and focus on the more fundamental flaws. The purpose is to identify those problems that can be solved by this thesis' approach. The debate of whether a canonical set of locutions is possible or even desirable will be avoided. As the literature of the field as well as precedence in other fields of AI seem to soundly point toward the negative.

Much of the criticism stems from FIPA's basing the semantics of its locutions[2]. BDIs of the communicating agents. The developers of KQML [Finin et al., 1994], a preceding ACL, knew that computational complexity and ungrounded semantics of multimodal BDI logics presented a problem. They also recognized the contentiousness of attempting to define the semantics of that logic. A caution ignored by the FIPA initiative. Another line of attack is that this approach takes one model of reasoning for agency and makes it inextricable from one particular model of agent communication. If the goal is interoperability in open multiagent systems, this is not a good starting point.

The formal semantics for FIPA ACL are defined in terms of Semantic Language (SL). Each communicative act has a feasibility condition and a rational effect defined in terms of a formula in SL. The problem of developing agents to conform to these requirements is compounded by the relative inaccessibility of the foundational docu-

[2]The FIPA specifications use the term Communicative Act. KQML calls them performatives. The literature in general uses the term locution. This document use the terms locution and communicative act interchangeably. Though arguments have been made against the use of the term performative, for consistency its usage has been maintained but only to refer to the communicative primitives of KQML

ments of SL [Sadek, 1991], and the relative importance of the informal versus formal descriptions of the locutions for purposes of conformity to the standard.

This unsteady foundation is further shaken by the questioning of whether intentions are at all necessary for basing the preconditions of locutions [Habermas, 1991]. Some consider context of the locution sufficient to determine the intention of an individual utterance. One way context is given is through the use of protocols. This has been recognized with regards to FIPA ACL specifically [Pitt and Mamdani, 1999]. There are also criticism for the lack of an operational semantics and questions over the soundness and complexity of SL.

The advantage LCC and dynamic protocols can provide FIPA and other models of communication is an explicit expression of the conditions for individual locutions and their context with respect to other locutions. This can be done using the traditional static or dynamic protocols. Unfortunately, the most fundamental problems with the FIPA ACL cannot be addressed. It is for this reason that the use of a more agreeable approach is necessitated, namely the adoption of dialogue games which is explored later in this chapter.

4.1.2 FIPA Communicative Acts as Protocols

By encoding each communicative act as a series of protocol steps or agent clauses, agents can communicate using FIPA performatives and have a publicly accountable expression of the feasible preconditions and the rational effect associated with that act. These conditions are expressed as LCC constraints or other communicative acts when it is implied to be a response to another. Rational effects within agents are not ensured. They are not necessary for conformance to the standard. If rational effects are written as LCC constraints, the interpretation is much less lenient as LCC requires the satisfaction of all constraints. The solution is to exclude the rational effects from the protocols altogether.

The most widely used FIPA communicative act is *inform*. Many of the

other communicative acts are defined in terms of *inform*. The formal description from [FIPA, 2001] for *inform* is as follows:

$$\texttt{Feasible Preconditions(FP): } \mathtt{B}_i\phi \wedge \neg \mathtt{B}_i(\mathtt{Bif}_j\phi \vee \mathtt{Uif}_j\phi)$$
$$\texttt{Rational Effects(RE): } \mathtt{B}_j\phi$$

'B' is stating that an agent identified by the following subscript letter has some belief. Therefore, $\mathtt{B}_i\phi$ states that agent '$\mathtt{i}$' believes ϕ. The predicate $\mathtt{Bif}_i\phi$ represents the uniquely SL construction which is shorthand for $\mathtt{B}_i\phi \vee \mathtt{B}_i\neg\phi$ (i.e. $\mathtt{i}$ either believes ϕ or its negation). Similarly, $\mathrm{Uif}_j\phi$ is short hand for either the agent is uncertain or is not uncertain about a proposition, $\mathtt{U}_i\phi \vee \mathtt{U}_i\neg\phi$.

Unfortunately we are already into some difficulties, because the informal english description gives three conditions:

- holds that some proposition is true,
- intends that the receiving agent also comes to believe that the proposition is true, and,
- does not already believe that the receiver has any knowledge of the truth of the proposition.

The first and third can be mapped more or less to the two conjuncts of the formal feasible preconditions, but the intention $\mathtt{I}_i(\mathtt{B}_j\phi)$ does not appear. Such is the fate of the poor engineer who attempts to develop an FIPA compliant agent. Luckily by using LCC, whatever interpretation we decide upon will be clear to our dialogical partners.

The translation from the formal description is straightforward and is shown in figure 4.1. For this to work, two additional requirements are made on the agents. Firstly, the agents must understand SL its definition of *belief*, *intentions*, *uncertainty*, etcetera, but this is unavoidable if one is to implement the FIPA ACL. The requirement introduced is the representation of the SL formula.

$$inform(\phi) \Rightarrow a(\_, Pid) \leftarrow believe(Id, \phi) \textbf{ and}$$
$$not(believe(Id, (bif(Pid, \phi) \textbf{ or } uif(Pid, \phi))))$$

$$assert(believe(Pid, \phi)) \leftarrow inform(\phi) \Leftarrow a(\_, Id)$$

Figure 4.1: LCC for the Inform Communicative Act

Two lines of attack are available that can be taken to implement the FIPA communicative acts as protocols. Each communicative act could be written as a snippet of protocol as shown in figure 4.1 which would then be spliced in by the transformation function when the agent wanted to use it. Another possibility, although it has the same functional characteristics, has more appeal for aesthetic reasons. Each communicative act could be written within an agent clause. Figure 4.2 has the same protocol actions described but they are encapsulated within the context of a role and agent clause.

$a(inform\_ca\_sender, Id)$ **::=**
$inform(\phi) \Rightarrow a(inform\_ca\_receiver, Pid) \leftarrow believe(Id, \phi)$ **and**
$not(believe(Id, (bif(Pid, \phi) \textbf{ or } uif(Pid, \phi))))$.

$a(inform\_ca\_receiver, Pid)$ **::=**
$assert(believe(Pid, \phi)) \leftarrow inform(\phi) \Leftarrow a(inform\_ca\_sender, Id)$.

Figure 4.2: FIPA Inform as LCC Agent Clauses

The use of an agent clause may seem to be unnecessary for a single communicative act, but this role encapsulation will be helpful when constructing more complex protocols from the single communicative act ones.

FIPA also recognizes the importance of context for defining communicative semantics and has defined a number of interaction protocols for typical agent interactions. From our simple protocols which govern the BDI constraints of an individual communicative act, the more complex protocols are constructed. For example, figures 4.3 and 4.5 show two AUML descriptions of FIPA protocols.

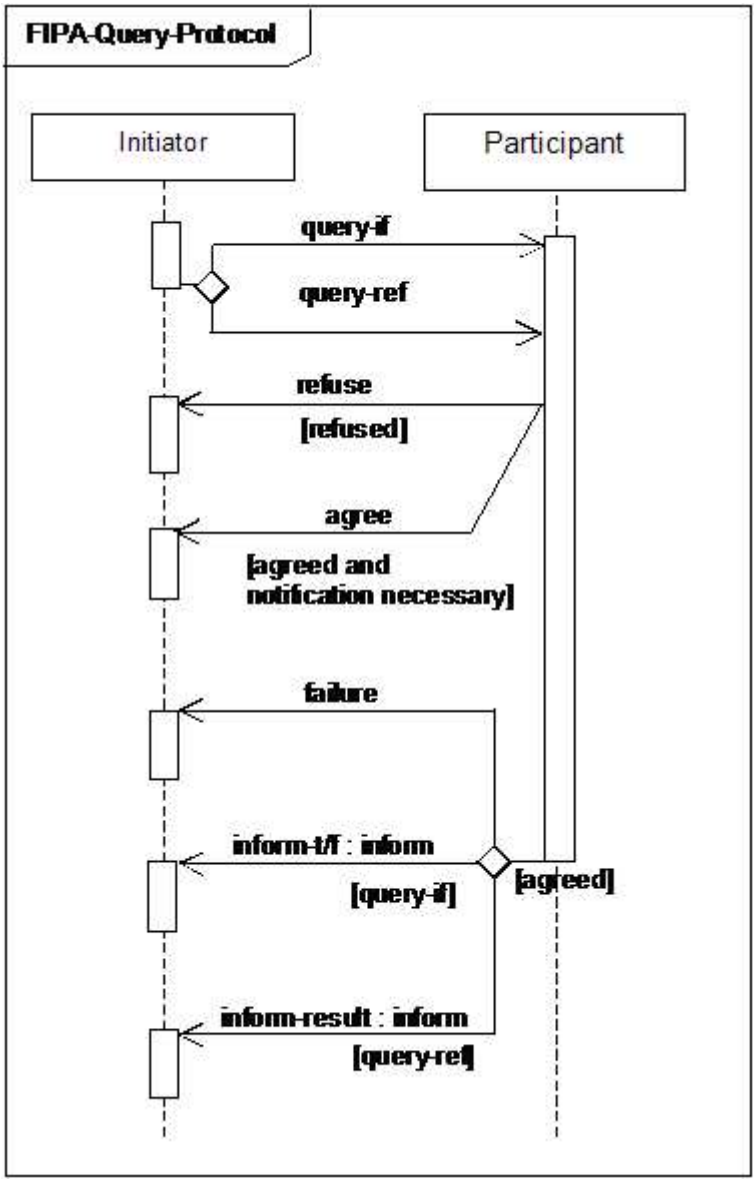

Figure 4.3: *AUML diagram for FIPA query protocol*

Figure 4.3 shows the protocol FIPA proposes for requests. A *query* is sent and an agent is meant to reply with either an *inform*, *failure*, *not_understood*, or *refuse*. This

is easily represented as an LCC protocol as shown in figure 4.4.

$$
\begin{array}{l}
a(query\_IP\_initiator, Id)\texttt{::=} \\
a(query\_ca\_sender, Id)\ \texttt{then} \\
\left(\begin{array}{l}
a(inform\_ca\_receiver, Id)\ \texttt{or} \\
a(failure\_ca\_receiver, Id)\ \texttt{or} \\
a(not\_understood\_ca\_receiver, Id)\ \texttt{or} \\
a(refuse\_ca\_receiver, Id)
\end{array}\right).
\end{array}
$$

$$
\begin{array}{l}
a(query\_IP\_responder, Pid)\texttt{::=} \\
a(query\_ca\_receiver, Pid)\ \texttt{then} \\
\left(\begin{array}{l}
a(inform\_ca\_sender, Pid)\ \texttt{or} \\
a(failure\_ca\_sender, Pid)\ \texttt{or} \\
a(not\_understood\_ca\_sender, Pid)\ \texttt{or} \\
a(refuse\_ca\_sender, Pid)
\end{array}\right).
\end{array}
$$

Figure 4.4: FIPA Query Protocol as LCC

Another well known protocol is the contract net protocol represented as AUML in figure 4.5. The initiator sends the *cfp* communicative act having satisfied its feasible preconditions. The recipient of the *cfp* can either reply with *not_understood*, *refuse*, or *propose*. Following the *propose*, the agent should next send a *reject_proposal* or an *accept_proposal*. If the agent sent an *accept_proposal* it can send a *cancel* or wait for the other agent to send an *inform* if it is successful or a *failure*.

An example of encoding this protocol in LCC is shown in figure 4.6.

Though these protocols are considered standards by FIPA, some authors have pointed out ambiguities or a lack of desirable functionality [Paurobally, 2002, Pitt and Mamdani, 1999]. This is much easier to fix through distributed dynamic protocols as modifications can be done at run time without changing the basic and standard

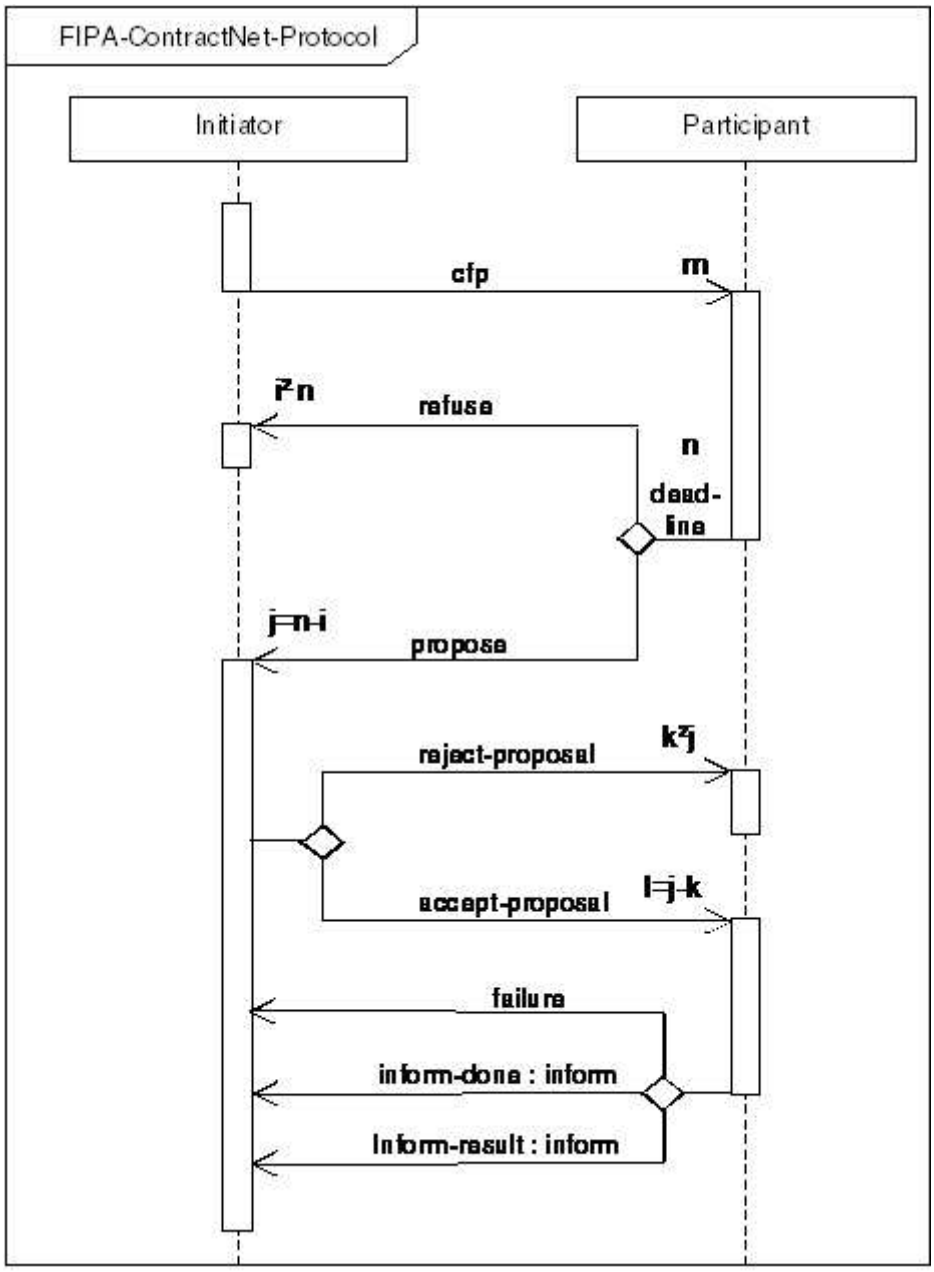

Figure 4.5: *AUML diagram for contract net protocol*

$$
\begin{aligned}
&a(contract\_net\_IP\_initiator, Id)\ \mathbf{::=}\\
&a(cfp\_ca\_sender, Id)\ \mathbf{then}\\
&\left(\begin{array}{l} a(propose\_ca\_receiver, Id)\ \mathbf{or} \\ a(refuse\_ca\_receiver, Id)\ \mathbf{or} \\ a(not\_understood\_ca\_receiver, Id) \end{array}\right)\ \mathbf{then}\\
&\left(\begin{array}{l} a(accept\_proposal\_ca\_sender, Id)\ \mathbf{or} \\ a(reject\_proposal\_ca\_sender, Id) \end{array}\right)\ \mathbf{then}\\
&\left(\begin{array}{l} a(inform\_ca\_receiver, Id)\ \mathbf{or} \\ a(failure\_ca\_receiver, Id)\ \mathbf{or} \\ a(cancel\_ca\_sender, Id) \end{array}\right).
\end{aligned}
$$

$$
\begin{aligned}
&a(contract\_net\_IP\_responder, Pid)\ \mathbf{::=}\\
&a(cfp\_ca\_receiver, Pid)\ \mathbf{then}\\
&\left(\begin{array}{l} a(propose\_ca\_sender, Pid)\ \mathbf{or} \\ a(refuse\_ca\_sender, Pid)\ \mathbf{or} \\ a(not\_understood\_ca\_sender, Pid) \end{array}\right)\ \mathbf{then}\\
&\left(\begin{array}{l} a(accept\_proposal\_ca\_receiver, Pid)\ \mathbf{or} \\ a(reject\_proposal\_ca\_receiver, Pid) \end{array}\right)\ \mathbf{then}\\
&\left(\begin{array}{l} a(inform\_ca\_sender, Pid)\ \mathbf{or} \\ a(failure\_ca\_sender, Pid)\ \mathbf{or} \\ a(cancel\_ca\_receiver, Pid) \end{array}\right).
\end{aligned}
$$

Figure 4.6: FIPA Contract Net Protocol as LCC

interaction protocols defined by FIPA's standards. Even if a remedy is impossible, at least with the protocol expressed in a manner that can be communicated and evaluated, errors or inconsistencies can be identified.

By the composition of individual communicative act protocols and the predefined FIPA interaction protocols as LCC, the FIPA complaint agent can create protocols at run time that also communicate the expectations for others on the use of those individual communicative acts and interaction protocols. For example, figure 4.7 shows the dialogue state for the initiating agent of the contract net protocol. For simplicity's sake only one of the participating agent's clauses is shown and the syntax of the FIPA ACL content languages is not followed.

$$
\begin{array}{l}
a(contract\_net\_IP\_initiator, agentA)\texttt{::=} \\
\quad a(cfp\_ca\_sender, agentA)\texttt{::=} \\
\quad\quad c(cfp(newParliment) \Rightarrow a(cfp\_ca\_receiver, agentB))\ \texttt{then} \\
\quad\quad a(propose\_ca\_receiver, agentA)\texttt{::=} \\
\quad\quad\quad c(propose(300B, 20years) \Leftarrow a(propose\_ca\_sender, agentB))\ \texttt{then} \\
\left(\begin{array}{l}
\underline{a(query\_IP\_initiator, agentA)\texttt{::=}} \\
\quad \underline{a(query\_ca\_sender, agentA)\texttt{::=}} \\
\quad\quad \underline{c(query(seriously) \Rightarrow a(query\_ca\_reciever, agentB))\ \texttt{then}} \\
\quad\quad \underline{a(inform\_ca\_receiver, agentA)\texttt{::=}} \\
\quad\quad\quad \underline{c(inform(seriously) \Leftarrow a(inform\_ca\_sender, agentB))}
\end{array}\right)\ \texttt{then} \\
\quad \left(\begin{array}{l}
a(accept\_proposal\_ca\_sender, agentA)\ \texttt{or} \\
a(reject\_proposal\_ca\_sender, agentA)
\end{array}\right)\ \texttt{then} \\
\quad \left(\begin{array}{l}
a(inform\_ca\_receiver, agentA)\ \texttt{or} \\
a(failure\_ca\_receiver, agentA)\ \texttt{or} \\
a(cancel\_ca\_sender, agentA)
\end{array}\right)
\end{array}
$$

Figure 4.7: FIPA CAs as an Dynamic LCC Clause

In the figure, the agent is sending a "call for proposals" (*cfp*) as part of the contract net protocol to receive proposals on the building of the new parliament. The protocols is followed by the agents until the agent must decide whether to accept or reject the proposal. The bid received seems more of an error than a serious bid. Rather than reject the proposal and cause the interaction to terminate unsuccessfully, the initiating agent, *agentA*, decides to query *agentB* to ensure the submitted proposal was not erroneous. At this point the protocol is adapted by the introduction of the initiating role for the query protocol as shown in figure 4.4. This in turn is a series of protocols for individual communicative acts. An example of the protocol for the *inform* act was shown in figure 4.2. The query is answered with the *inform* and concludes the query interaction protocol. The actions of FIPA's query interaction protocol are underlined in the figure. The agent, confident that the received bid is not an error now, returns to the step in the contract net protocol where it must decide whether to accept the received proposal.

It has been shown that it is possible to use LCC and FIPA's communicative acts and interaction protocols to create dynamic and meaningful protocols. However, the use of BDI-logic based models of communication as a model for distributed protocols can not be endorsed. Ultimately, there is a problem at the abstract level. The inclusion of explicit constraints on the internal states of the agent seems a step too far in the balance between autonomy and reliability. It provides the system with a public declaration of intended belief states to which agents could be held accountable, but it does not solve the problem of an insincere agent. A sufficiently clever agent can still pretend to satisfy the constraint for a belief state that it does not believe. This is the main reason, that we turn to a more accommodating communicative model to try to facilitate dynamic protocols. Dialogue games' focus is on the interaction itself to build a model of communication rather than the internal mental caprices of agents, and should fit well with LCC's coordination oriented approach to protocols.

4.2 Dialogue Games

LCC is a mechanism for communication designed independent of any one rational or communicative model of agency. The language allows the creation of flexible protocols as well as the ability to have dynamic protocols. The framework for the language provides the means for the dissemination of the interaction protocols, which will be useful for dynamic conversation spaces. This is done in a decentralised manner in keeping with the spirit of the agency paradigm. Yet this is not enough to drive the complex and robust agent conversations which we seek. It would not be desirable to allow agents to make arbitrary changes to their protocols. A model of communication is required which can take advantage of the unique properties of the LCC approach. After thoroughly investigating the popular models of chapter 2, the model that proved to be most promising is dialogue games.

Dialogue games have several important characteristics which make them suitable for our purposes. Dialogue games are similar to conversation policies [Greaves et al., 2000] with the advantage of having scope over the entire conversation rather than specifying policies over only segments of a dialogue. Dialogue games are one of the many progeny of argumentation theory to find application in multiagent communication. The orderly and structured nature of dialogue games are easily translated to LCC protocols. The dialogue games from which we develop our protocols are not defined with any preference for a single semantics of agency. There is a separation between the semantics of agency and the semantics and syntax for communication. This avoids sematic verification woes that bedevil BDI approaches to agent communication as mentioned earlier in this chapter. Previous research has provided us with a number of formal specifications for dialogue games [McBurney and Parsons, 2002].

The shortcomings of using dialogue games can be addressed by an implementation in the LCC framework. Implementation is the keyword. The majority of dialogue game research is focused on the theory and formalisms though a sea change is occurring with the argumentation community, typified by the Argumentation Interchange

Format (AIF) effort [Chesnevar et al., 2005]. The development of solid formal foundations have been made to the detriment of practicalities. The question of whether these dialogue games will survive in the wilds of open multiagent systems are of less interest to the community. The implementation of dialogue game specifications for use in the LCC framework is facilitated by both approaches having their roots in propositional logic. The various dialogue games developed suffer from a common problem in agent communication. This is the problem of dialects. Each agent will be developed from specifications of a game which detail the 'house rules' for that particular type of game. Two agents who can play a game of negotiation or deliberation may not be able to communicate if their 'house rules' differ. Work has been done to establish when these differences are negligible [Johnson et al., 2002], but the LCC answer is to ensure only one protocol is used. Agents by the act of playing the dialogue game also communicate its 'house rules'.

Dialogue games and LCC compliment each other to create dynamic interaction protocols for use in open multiagent systems. The models of agent communication found in dialogue games fit well with the novel mechanism of communication found in the LCC framework. Both approaches see the separation of syntax and semantics as of paramount importance. Their shared foundation in propositional logic makes their amalgamation a simpler task. Together a far better solution to the problem at hand is found than with either in isolation. The spirit of dialogue games has found a host in the body of LCC.

Section 4.2.1 defines several dialogue games as LCC clauses. Using these, section 4.2.2 illustrates how more complex dialogue games are implemented as adaptable protocols.

4.2.1 Writing Dialogue Games as LCC Protocols

The following subsections define several types of Dialogue Games taken from [Parsons et al., 2004]. There are several reasons for the focus on the definitions

described in that paper. The paper is the result of work that has had several iterations of development and peer review. The clearly written 'English' definitions, which are reproduced in this chapter, facilitates translation to an LCC protocol. The same authors have also published work concerning the creation of complex dialogue games, and commitment in dialogue games. This provides consistency for the demonstration using LCC in section 4.2.2 . The literature that this chapter utilises is also bolstered by formal definitions of the locutions used, dialogue games, and their combinations. All of which makes our task of translation much easier.

The set of locutions used are also from [Parsons et al., 2003b, Parsons et al., 2003a, Parsons et al., 2002]. It is assumed the agents know the semantics of these locutions. This assumption also covers the definition of the logical operators that are used in the content of the messages. As seen previously, the semantics of individual locutions can be clarified through the constraints or common knowledge. This is of secondary concern and is left as an exercise for the reader. Instead the focus will be on the protocols and their dynamic composition to create complex dialogue games from atomic ones. The games described in the following subsections have not been altered and are true to the authors specifications. It is correct to identify certain anomalies or even errors such as non-termination in some of the games definitions. It is not the task of this thesis to address these, instead we advocate that the formal and computational representation into which we translate the games would facilitate the identification of these problems. Errors such as these are a constant problem with paper and pen representations.

4.2.1.1 Information Seeking Game

Information Seeking games are used by an agent that wishes to know the answer to some proposition and it believes another agent knows this information. This game is less overtly antagonistic than some of the others. The goal of the game is to spread information. The initial condition of the initiating agent lacks that information, and the other agent is meant to alleviate that.

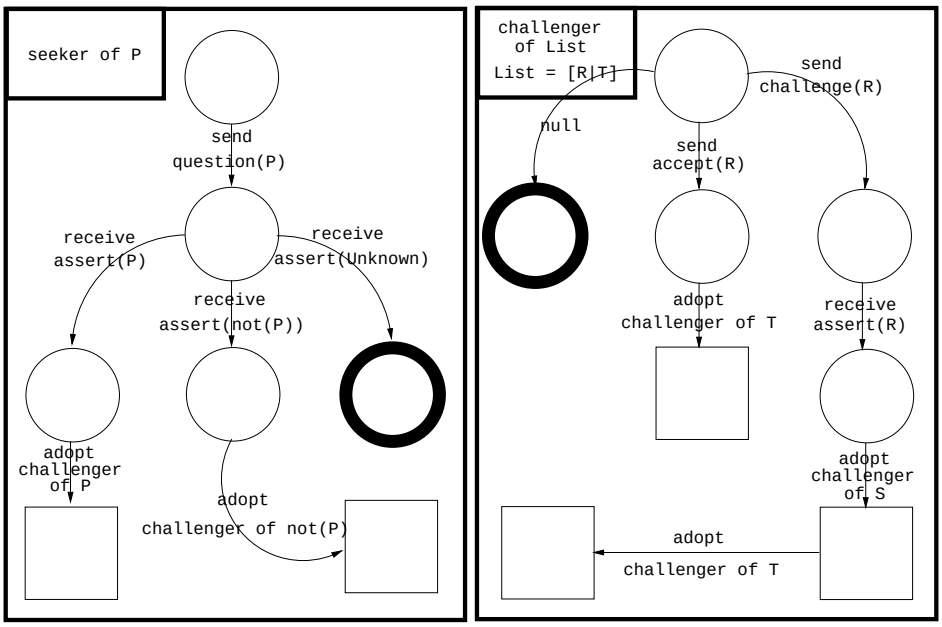

Figure 4.8: *Graphical Representation of the Information seeking Game*

The English description of the information seeking game defined in [Parsons et al., 2004] is as follows:

1. Agent *A* asks *question(p)*.

2. Depending upon the contents of its knowledge-base and its assertion attitude, agent *B* replies with either *assert(p)*, *assert(¬p)*, or *assert($\mathcal{U}$)*, where $\mathcal{U}$ indicates that, for whatever reason, *B* cannot give an answer.

3. *A* either *accepts B*'s response, if its acceptance attitude allows, or *challenges*. $\mathcal{U}$ cannot be challenged, and as soon as it is asserted, the dialogue terminates without the question being resolved.

4. *B* replies to a challenge with an *assert($\mathcal{S}$)*, where $\mathcal{S}$ is the support of an argument for the last proposition challenged by *A*.

5. Go to (3) for each proposition in $\mathcal{S}$ in turn.

Figure 4.8 provides a diagram to represent the dialogue game. The left hand side role coresponds to the agent clause found in figure 4.9 and the role on right hand side

is defined in an agent clause from figure 4.10. Arcs are actions taken with the dialogue which are either messages being communicated or roles being adopted. Circles identify states and roles are represented as squares. The figures represent the interaction from the view point of the initiating agent. Figures 4.9 and 4.10 are one translation from the english description above to the computational LCC representation of the protocol. Translation is the keyword. It is well known that interpretations and translations commonly result in information loss. Traditionally in protocol led multiagent systems, two agent engineers read the same specification, create two interpretations, and two implementations and assume that the other agent engineer has developed his agent in such a way that the agents can still communicate. The risk of such assumptions may not be that great in domains of lesser complexity and regularity, but as the complexity of agent interactions increase so does the risk of misunderstanding. The use of LCC, and the introduction of the dynamic adaptations explored in this thesis provide an advantage over this traditional model. In LCC there is only one protocol and it is communicated during interactions. If the other agent has different expectations on the interaction, it is at least possible to evaluate the LCC protocol as it is expressed in a computational and declarative format. Having the communicative norms as LCC in a computational format also allows the protocols to be verified before interactions (e.g. simulation, model checking, etc.).

The protocol of figure 4.9 defines the initial roles for the agents involved. The two roles are symmetrical with one another. The focus will be from the perspective of the initiating agent as the explanation for the design decisions is given. The separation of the information seeking game into two roles is to enable the recursion over the set of supporting propositions described by rule 5 of the english description. Figure 4.10 is the definition of the two recursive roles that the agents can adopt during the dialogue.

The initial role that an agent takes to begin an information seeking game is the *seeker* role of figure 4.9. Part of this role's definition, a(seeker(P,B),A), is *P* which is the proposition for which the information is being sought and *B* the identifier of the

$$
\begin{array}{l}
a(seeker(P,B),A)\texttt{::=} \\
question(P) \Rightarrow a(provider(P,A),B)\ \texttt{then} \\
\left(\begin{array}{l} assert(P) \Leftarrow a(provider(P,A),B)\ \texttt{then} \\ a(challenger([P],B),A) \end{array}\right)\ \texttt{or} \\
\left(\begin{array}{l} assert(not(P)) \Leftarrow a(provider(P,A),B)\ \texttt{then} \\ a(challenger([not(P)],B),A) \end{array}\right)\ \texttt{or} \\
assert(\mathcal{U}) \Leftarrow a(provider(P,A),B).
\end{array}
$$

$$
\begin{array}{l}
a(provider(P,A),B)\texttt{::=} \\
question(P) \Leftarrow a(seeker(P,B),A)\ \texttt{then} \\
\left(\begin{array}{l} assert(P) \Rightarrow a(seeker(P,B),A)\ \texttt{then} \\ a(defender([P],A),B) \end{array}\right)\ \texttt{or} \\
\left(\begin{array}{l} assert(not(P)) \Rightarrow a(seeker(P,B),A)\ \texttt{then} \\ a(defender([not(P)],A),B) \end{array}\right)\ \texttt{or} \\
assert(\mathcal{U}) \Rightarrow a(seeker(P,B),A).
\end{array}
$$

Figure 4.9: A Protocol for an Information Seeking Dialogue Game

agent with whom the agent *A* would like to begin the game. The LCC protocol begins the game proper with the *question* locution sent is the commencement rule specified in step 1 of the game's description. Looking at the partner's clause, we see the *provider* role has the complementary step of receiving that *question* locution. The *provider* can respond by *assert*ing the proposition *P*, its negation, or unknown. Unknown is the the statement expressed as $\mathcal{U}$ in rule 2 of the specification. Step 3 states that upon the assertion of unknown, $\mathcal{U}$, the dialogue should terminate. The *seeker*'s clause has the steps for the reception of the three possible replies from the *provider* to the initial *question*. If *P* or its negation are asserted the next step is the adoption of the recursing role of '*challenger*' defined in 4.10. The immediate completion of the protocol, and thus the interaction, upon receiving the *assert* of $\mathcal{U}$ is also defined by the protocol.

Already important design decisions have been made. Another engineer encoding this dialogue game might choose to explicitly constrain the sending of the *question* and the subsequent *assert* responses. I have throughout the definition of these protocols tried to include the least number of explicit constraints upon the messages, such as when the description makes an explicit condition upon the sending of an message. Allowing the focus to be on the messages being exchanged and ultimately the modification of the protocol to achieve the dynamism that is our goal. This is also consistent with the basic philosophy of the LCC framework in that it is important to make the minimal engineering requirements upon communicating agents. The consequence of this style puts the burden upon the agent's reasoning ability, but also increases its autonomy.

After the assertion of either the proposition or its negation, the role of *challenger* is taken by the initiating agent as defined in figure 4.10. For its next action the agent has a choice; it can send the *accept* of the previously received assertion or *challenge* it. The third choice, $null \leftarrow (List = [])$, is the base case for the recursion. The constraint is satisfied when the list of propositions is empty. The *null* action is done and the role, and thus the protocol, is finished.

$$
\begin{array}{l}
a(challenger(List,B),A)\,\mathbf{::=} \\
null \leftarrow (List = []) \;\mathbf{or} \\
\left(\begin{array}{l}
accept(R) \Rightarrow a(defender(List,A),B) \leftarrow (List = [R|T])\;\mathbf{then} \\
a(challenger(T,B),A))
\end{array}\right)\;\mathbf{or} \\
\left(\begin{array}{l}
challenge(R) \Rightarrow a(defender(List,A),B) \leftarrow (List = [R|T])\;\mathbf{then} \\
assert(S) \Leftarrow a(defender(List,A),B)\;\mathbf{then} \\
a(challenger(S,B),A)\;\mathbf{then}\; a(challenger(T,B),A)
\end{array}\right).
\end{array}
$$

$$
\begin{array}{l}
a(defender(List,A),B)\,\mathbf{::=} \\
null \leftarrow (List = []) \;\mathbf{or} \\
\left(\begin{array}{l}
(List = [R|T]) \leftarrow accept(R) \Leftarrow a(challenger(List,B),A)\;\mathbf{then} \\
a(defender(T,A),B)
\end{array}\right)\;\mathbf{or} \\
\left(\begin{array}{l}
(List = [R|T]) \leftarrow challenge(R) \Leftarrow a(challenger(List,B),A)\;\mathbf{then} \\
assert(S) \Rightarrow a(challenger(List,B),A) \leftarrow support(R,S)\;\mathbf{then} \\
a(defender(S,A),B)\;\mathbf{then}\; a(defender(T,A),B)
\end{array}\right).
\end{array}
$$

Figure 4.10: The Recursive Roles for the Information Seeking Dialogue Game Protocol

The *accept* and *challenge* locutions have the constraint, $\leftarrow (List = [R|T])$. This takes the list of propositions under consideration which for the first iteration is always the single proposition P and separates the first element of that list. Once again I have left explicit constraints concerning the agent's acceptance attitude out of the protocol, but certainly it is appropriate to include them especially considering their explicit mention in rule 3 of the specification. For example, an additional constraint could be added to the sending of *accept* in the challenger role like this:

$$accept(R) \Rightarrow a(defender(List,A),B) \leftarrow (List = [R|T]) \textbf{ and } acceptable(R)$$

Now the protocol explicitly requires an acceptability predicate be satisfied before *accept* can be sent rather than trusting the agent to do this itself. However this has no effect on creating complex dialogue games with dynamic protocols.

If the agent chooses to *accept* the agent's assertion of either P or *not(P)*, the protocol recurses upon the tail of the list of propositions. If this acceptance was for the initial proposition the tail will be empty and the base case constraint will succeed and complete the dialogue game. This termination rule is not explicitly expressed in the steps of the proposition, but it is safe to assume that it is not controversial design decision. The specification is very clear as to what should happen after a *challenge* is sent. Rule 4 tells us the recipient of the challenge should send an *assert($\mathcal{S}$)*. The *defender* role has defined this step in the protocol. The constraint is included in order to introduce the new propositions into the protocol. Rule 5 is the recursive step, the agents must continue through the supporting propositions considering whether to *accept* or *challenge* them and if they in turn are challenged it must also consider those newly introduced propositions. The *challenger* agent first adopts the *a(challenger(S,B),A)* role for the newly introduced $\mathcal{S}$ propositions and then continues to take the *a(challenger(T,B),A)* for the remainder of the existing list propositions, T.

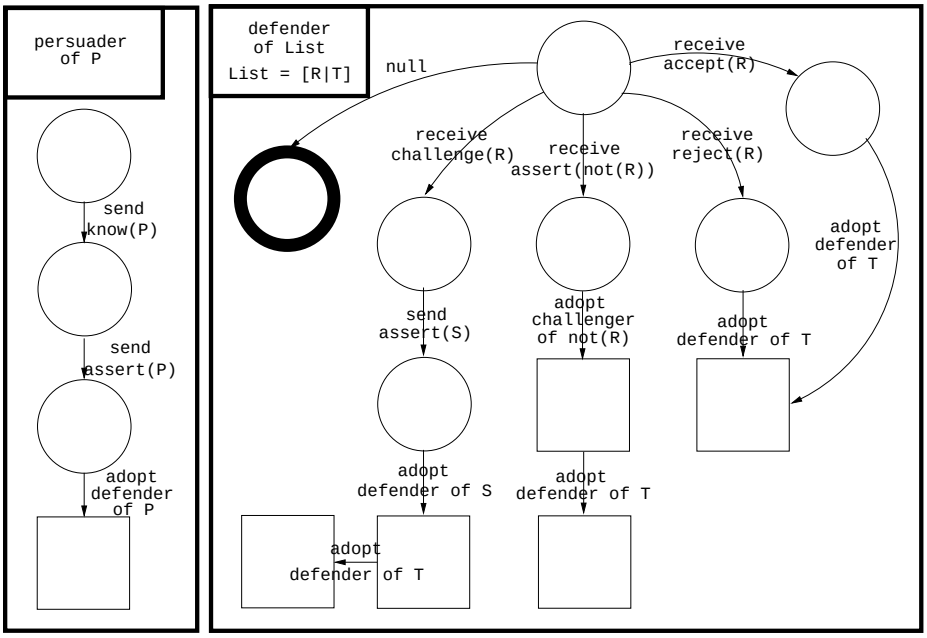

Figure 4.11: *Graphical Representation of the Persuasion Game*

4.2.1.2 Persuasion

Persuasion dialogues games are played when one agent wishes to convince another of some proposition. The initiator of the game wishes to have another adopt some belief or intention. The agent proposes some statement and attempts to convince the other to agree upon that statement. The goal of the dialogue is to reconcile the disparity of belief between the agents. It may be the case that the other agent already believes in the proposition which makes the persuader's task much easier. The English description of the game's rules are as follows [Parsons et al., 2004]:

1. *A* issues a *know(p)*, indicating it believes that *p* is the case.

2. *A asserts p*.

3. *B accepts p* if its acceptance attitude allows, else *B* either *asserts* $\neg p$ if it is allowed to, or else *challenges p*.

4. If *B* asserts $\neg p$, then go to (2) with the roles reversed and $\neg p$ in the place of *p*.

5. If B has *challenge*d, then:

   (a) A *asserts* $\mathcal{S}$, the support for p;

   (b) Go to (2) for each $s \in \mathcal{S}$ in turn.

6. If B does not *challenge*, then it issues either *accept(p)* or *reject(p)*, depending upon the status of p for it.

$$
\begin{array}{l}
a(persuader(P,B),A)\texttt{::=} \\
know(P) \Rightarrow a(listener(P,A),B)\ \texttt{then} \\
assert(P) \Rightarrow a(listener(P,A),B)\ \texttt{then} \\
a(defender([P],B),A).
\end{array}
$$

$$
\begin{array}{l}
a(listener(P,A),B)\texttt{::=} \\
know(P) \Leftarrow a(persuader(P,B),A)\ \texttt{then} \\
assert(P) \Leftarrow a(persuader(P,B),A)\ \texttt{then} \\
a(challenger([P],A),B).
\end{array}
$$

Figure 4.12: A Protocol for a Persuasion Dialogue Game

Figure 4.11 shows the graph for the game, and figures 4.12 and 4.13 are the persuasion game as an LCC protocol. The initiating role is the *persuader* role. The bootstrapping variables for this game are the same, p is the proposition the agent wants to discuss and B is the dialogical partner's identifier. The game begins by the sending of two locutions, *know(p)* and *assert(p)*. These rules 1 and 2 are translated to first steps in the *persuader*'s clause.

The three possible responses described in rule 3 are *accept*, *challenge*, or *assert(not(P))*. After the *persuader* agent communicates the opening locutions, it adopts the *defender* role as shown in figure 4.13. This is the recursive role and once again we have the *null* step and the list dividing constraints. Also, there is the ability to commu-

$$
\begin{array}{l}
a(defender(List,A),B)\texttt{::=} \\
null \leftarrow (List=[])\ \texttt{or} \\
\left(\begin{array}{l}
\left(\begin{array}{l}
(List=[R|T]) \leftarrow accept(R) \Leftarrow a(challenger(List,B),A)\ \texttt{or} \\
(List=[R|T]) \leftarrow reject(R) \Leftarrow a(challenger(List,B),A)
\end{array}\right)\ \texttt{then} \\
a(defender(T,A),B)
\end{array}\right)\ \texttt{or} \\
\left(\begin{array}{l}
(List=[R|T]) \leftarrow challenge(R) \Leftarrow a(challenger(List,B),A)\ \texttt{then} \\
assert(S) \Rightarrow a(challenger(List,B),A) \leftarrow support(R,S)\ \texttt{then} \\
a(defender(S,A),B)\ \texttt{then}\ a(defender(T,A),B)
\end{array}\right)\ \texttt{or} \\
\left(\begin{array}{l}
(List=[R|T]) \leftarrow assert(not(R)) \Leftarrow a(challenger(List,B),A)\ \texttt{then} \\
a(challenger([not(R)],A),B)\ \texttt{then}\ a(defender(T,A),B)
\end{array}\right).
\end{array}
$$

$$
\begin{array}{l}
a(challenger(List,B),A)\texttt{::=} \\
null \leftarrow (List=[])\ \texttt{or} \\
\left(\begin{array}{l}
\left(\begin{array}{l}
accept(R) \Rightarrow a(defender(List,A),B) \leftarrow (List=[R|T])\ \texttt{or} \\
reject(R) \Rightarrow a(defender(List,A),B) \leftarrow (List=[R|T])
\end{array}\right)\ \texttt{then} \\
a(challenger(T,B),A)
\end{array}\right)\ \texttt{or} \\
\left(\begin{array}{l}
challenge(R) \Rightarrow a(defender(List,A),B) \leftarrow (List=[R|T])\ \texttt{then} \\
assert(S) \Leftarrow a(defender(List,A),B)\ \texttt{then} \\
a(challenger(S,B),A)\ \texttt{then}\ a(challenger(T,B),A)
\end{array}\right)\ \texttt{or} \\
\left(\begin{array}{l}
assert(not(R)) \Rightarrow a(defender(List,A),B) \leftarrow (List=[R|T])\ \texttt{then} \\
a(defender([not(R)],B),A)\ \texttt{then}\ a(challenger(T,B),A)
\end{array}\right).
\end{array}
$$

Figure 4.13: The Recursive Roles for the Persuasion Dialogue Game Protocol

nicate a *reject* of *P* which is described in rule 6. If the agent *asserts*s the negation then they switch roles. The *defender* of *P* becomes the *challenger* of *not(P)* and vice versa for the other agent. Rule 5 is similar to the information seeking game. If a *challenge* is sent the agents adopt recursive roles over the set $\mathcal{S}$ of supporting propositions. This behaviour, similar to the information seeking game, can also be seen in 4.13.

4.2.1.3 Inquiry

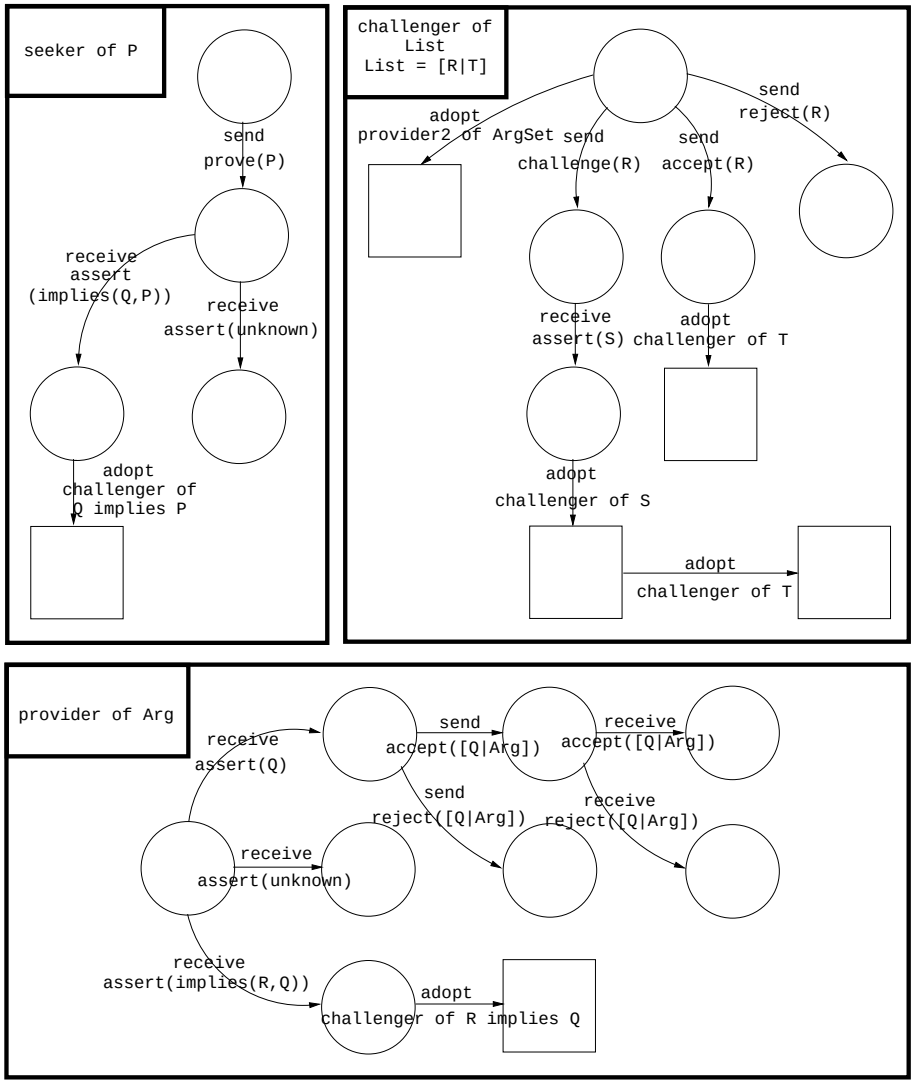

Figure 4.14: *Graphical Representation of the Inquiry Game*

Another game defined in [Parsons et al., 2004] is inquiries. Inquiry games are

played when two agent seek to develop the argument in support for some proposition. This game is a less explicitly antagonistic dialogue as they agents collaborate to prove some statement that neither can develop alone. There is also more equality of role in this dialogue as both agents have a lack in knowledge and both share the goal to alleviate that deficiency.

1. *B* proffers *prove(p)*, inviting *A* to join it in the search for a proof of *p*.
2. *A asserts* q→p for some *q* or *asserts* $\mathcal{U}$.
3. *B accepts* q→p if its acceptance attitude allows, or *challenges* it.
4. *A* replies to a *challenge* with *assert(*$\mathcal{S}$*)*, where $\mathcal{S}$ is the support of an argument for the last proposition challenged by *B*.
5. Go to (2) for each proposition $s \in \mathcal{S}$ in turn, replacing $q \rightarrow p$ by *s*
6. *B* asserts *q*, or $r \rightarrow q$ for some *r*, or $\mathcal{U}$.
7. If $\mathcal{A}(CS(A) \cup CS(B))$ includes an argument for *p* that is acceptable to both agents, then first *A* and then *B accept* it and the dialogue terminates successfully.
8. If at any point one of the propositions is not acceptable to an agent, in issues a reject, and the dialogue ends successfully.
9. Go to 6, reversing the roles of *A* and *B* and substituting *r* for *q* and some *t* for *r*.

Figure 4.14 shows the agent clauses as a graph. The LCC protocol for the *inquiry* dialogue game is defined in figure 4.15 and continued in figures 4.16 and 4.17. The initiating role for this game is the *seeker* role. The first message sent is *prove(P)* as specified by rule 1. The response can be either the assertion of a proposition that implies *P* (e.g. *Q implies P*) or unknown, $\mathcal{U}$. The assertion of unknown ends the dialogue. The initiating agent takes the role of *challenger*. The role has the extra variable, *ArgSet*, which keeps record of the argument as it develops. This feature could

$$
\begin{array}{l}
a(seeker(P,A),B)\texttt{::=} \\
prove(P) \Rightarrow a(provider(P,B),A)\ \texttt{then} \\
\left(\begin{array}{l} \left(\begin{array}{l} assert(implies(Q,P)) \Leftarrow a(provider(P,B),A)\ \texttt{then} \\ a(challenger([implies(Q,P)],[P,implies(Q,P)],A),B) \end{array} \right) \texttt{or} \\ assert(unknown) \Leftarrow a(provider(P,B),A) \end{array} \right).
\end{array}
$$

$$
\begin{array}{l}
a(provider(P,B),A)\texttt{::=} \\
prove(P) \Leftarrow a(seeker(P,A),B)\ \texttt{then} \\
\left(\begin{array}{l} \left(\begin{array}{l} assert(implies(Q,P)) \Rightarrow a(seeker(P,A),B) \leftarrow implies(Q,P)\ \texttt{then} \\ a(defender([implies(Q,P)],[P,implies(Q,P)],B),A) \end{array} \right) \texttt{or} \\ assert(unknown) \Rightarrow a(seeker(P,A),B) \end{array} \right).
\end{array}
$$

Figure 4.15: A Protocol for an Inquiry Dialogue Game

have been excluded as the argument is recorded in the dialogue state. Once again design decisions have had to be made when reading the english description.

The reply to the assertion of *Q implies P* can be *accept*, *challenge* or *reject*. The *reject* step is allowed by rule 8. The *accept* or *challenge* is specified by rule 3. The usual business occurs when a *challenge* occurs and comes from rule 4 and 5. The next stage is captured by the *provider2* role. The initiating agent can now assert the argument *Q* or another implication such as *R implies Q* or $\mathcal{U}$. If the argument is satisfactory the second agent may *accept* or *reject* the argument. In response to an *accept*, the initiating agent can also either *accept* or *reject* in accordance with rule 7.

The agent can reply to the assert of *R implies Q* the agents switch roles, it is now the initiating agents turn to have its assertions accepted, rejected, or challenged by taking the defender role. This mutual recursion is performed between the *challenger / defender* and the *provider2 / seeker2* roles until an argument is agreed upon. Thus

$$
\begin{array}{l}
a(challenger(List,ArgSet,A),B)\,\mathbf{::=} \\
(a(provider2(ArgSet,A),B) \leftarrow (List=[])) \ \mathbf{or} \\
\left(\begin{array}{l} accept(R) \Rightarrow a(defender(List,ArgSet,B),A) \leftarrow (List=[R|T]) \ \mathbf{then} \\ a(challenger(T,ArgSet,A),B) \end{array}\right) \mathbf{or} \\
\left(\begin{array}{l} challenge(R) \Rightarrow a(defender(List,ArgSet,B),A) \leftarrow (List=[R|T]) \ \mathbf{then} \\ assert(S) \Leftarrow a(defender(List,ArgSet,B),A) \ \mathbf{then} \\ a(challenger(S,[S|ArgSet],A),B) \ \mathbf{then}\ a(challenger(T,[S|ArgSet],A),B) \end{array}\right) \mathbf{or} \\
reject(R) \Rightarrow a(defender(List,ArgSet,B),A) \leftarrow (List=[R|T]).
\end{array}
$$

$$
\begin{array}{l}
a(defender(List,ArgSet,B),A)\,\mathbf{::=} \\
(a(seeker2(ArgSet,B),A) \leftarrow (List=[])) \ \mathbf{or} \\
\left(\begin{array}{l} (List=[R|T]) \leftarrow accept(R) \Leftarrow a(challenger(List,ArgSet,A),B) \ \mathbf{then} \\ a(defender(T,ArgSet,B),A) \end{array}\right) \mathbf{or} \\
\left(\begin{array}{l} (List=[R|T]) \leftarrow challenge(R) \Leftarrow a(challenger(List,ArgSet,A),B) \ \mathbf{then} \\ assert(S) \Rightarrow a(challenger(List,ArgSet,A),B) \leftarrow support(R,S) \ \mathbf{then} \\ a(defender(S,[S|ArgSet],B),A) \ \mathbf{then}\ a(defender(T,[S|ArgSet],B),A) \end{array}\right) \mathbf{or} \\
(List=[R|T]) \leftarrow reject(R) \Leftarrow a(challenger(List,ArgSet,A),B).
\end{array}
$$

Figure 4.16: The Recursive Clauses for the Inquiry Dialogue Game Protocol

$$
\begin{array}{l}
a(provider2(Arg,A),B)\texttt{::=} \\
\left(\begin{array}{l}
assert(Q) \Rightarrow a(seeker2(Arg,B),A) \leftarrow know(Q)\ \texttt{then} \\
\left(\begin{array}{l}
accept([Q|Arg]) \Leftarrow a(seeker2(Arg,B),A)\ \texttt{then} \\
accept([Q|Arg]) \Rightarrow a(seeker2(Arg,B),A)\ \texttt{or} \\
reject([Q|Arg]) \Rightarrow a(seeker2(Arg,B),A)
\end{array}\right)\ \texttt{or} \\
reject([Q|Arg]) \Leftarrow a(seeker2(Arg,B),A)
\end{array}\right)\ \texttt{or} \\
\left(\begin{array}{l}
assert(implies(R,Q)) \Rightarrow a(seeker2(Arg,B),A) \leftarrow implies(R,Q)\ \texttt{then} \\
a(defender([implies(R,Q)],[implies(R,Q)|Arg],A),B)
\end{array}\right)\ \texttt{or} \\
assert(unknown) \Rightarrow a(seeker2(Arg,B),A).
\end{array}
$$

$$
\begin{array}{l}
a(seeker2(Arg,B),A)\texttt{::=} \\
\left(\begin{array}{l}
assert(Q) \Leftarrow a(provider2(Arg,A),B)\ \texttt{then} \\
\left(\begin{array}{l}
accept([Q|Arg]) \Rightarrow a(provider2(Arg,A),B)\ \texttt{then} \\
accept([Q|Arg]) \Leftarrow a(provider2(Arg,A),B)\ \texttt{or} \\
reject([Q|Arg]) \Leftarrow a(provider2(Arg,A),B)
\end{array}\right)\ \texttt{or} \\
reject([Q|Arg]) \Rightarrow a(provider2(Arg,A),B)
\end{array}\right)\ \texttt{or} \\
\left(\begin{array}{l}
assert(implies(R,Q)) \Leftarrow a(provider2(Arg,A),B)\ \texttt{then} \\
a(challenger([implies(R,Q)],[implies(R,Q)|Arg],B),A)
\end{array}\right)\ \texttt{or} \\
assert(unknown) \Leftarrow a(provider2(Arg,A),B).
\end{array}
$$

Figure 4.17: The Recursive Clauses for the Inquiry Dialogue Game Protocol[Cont'd]

armed with these agent clauses and the Dynamic LCC framework, it is now possible to demonstrate its practice.

4.2.2 Creating Complex Dialogue Games

The protocol language and the agent clauses it defines, as described so far, already allows for a spectrum of adaptability. At one extreme, the protocol can be fully constrained. Protocols at this end of the spectrum would be close to the traditional protocols and some electronic institutions. By rigidly defining each step of the protocol, agents could be confined to little more than remote processing. This sacrifice allows the construction of reliable and verifiable agent systems. At the other extreme, the protocols could be nothing more than the ordering of messages or even just the statement of legal messages (without any ordering) to be sent and received. Though the protocol language is expressive enough for both extremes of the spectrum, the bulk of interactions are going to be somewhere in the middle. A certain amount of the dialogue will need to be constrained to ensure a useful dialogue can occur. Unfortunately, any flexibility must decided *a priori* to the execution of the protocols. Instead, this section proposes an approach that maintains the use of protocols and their reliable and predictable norms that map out a course through the conversation space, but also allow a dynamic composition of more complex protocols by the combination of atomic dialogue games in order to explore runtime opportunities.

4.2.2.1 The Formal Framework for Complex Dialogue Games

The formal framework for which these dynamic protocols is based is described in the paper of [McBurney and Parsons, 2002]. These are the same authors who have defined the rules for the atomic games of [Parsons et al., 2004]. In [McBurney and Parsons, 2002] the authors formally describe a framework for creating complex dialogue games.

The framework proposes three layers; topic, dialogue, and control. The topic layer

is concerned with the content of locutions. They are the propositions represented in a logical language, $\mathcal{L}$. The assumptions made about the topic layer are consistent with our implementation.

The dialogue layer is the game itself. This layer consists of various commencement, combination, termination rules and locutions of individual games, and is described using LCC protocols. The previous sections gave examples of this for three of those games. These will be the basis for the example of the complex dialogue game demonstrated in section 4.2.2.2

Finally, the control layer requires a few more agent clauses to be defined. This layer is where the agent proposes the start of individual games to be played. During the interaction, agents can propose to initiate another game. The protocol for the control layer is defined in 4.18. The initiating game proposes to *begin* an instance of an atomic dialogue game. The other agent can *reject* or *accept* the proposal. If it *accept*s, the protocol for that game is started as a consequence of the constraint *startgame(Game,P)*. This is a possible way to indicate a transformation *must* be done rather than solely relying upon the agent's rational capabilities.

$$a(control\_layer\_a, A) \texttt{::=}$$
$$begin(Game, P) \Rightarrow a(control\_layer\_b, B) \leftarrow choosegame(Game, P, B) \texttt{ then}$$
$$\left(\begin{array}{l} reject(Game, P) \Leftarrow a(control\_layer\_b, B) \texttt{ or} \\ startgame(Game, P) \leftarrow accept(Game, P) \Leftarrow a(control\_layer\_b, B) \end{array} \right).$$

$$a(control\_layer\_b, B) \texttt{::=}$$
$$begin(Game, P) \Leftarrow a(control\_layer\_a, A) \texttt{ then}$$
$$\left(\begin{array}{l} reject(Game, P) \Rightarrow a(control\_layer\_a, A) \texttt{ or} \\ accept(Game, P) \Rightarrow a(control\_layer\_a, A) \end{array} \right).$$

Figure 4.18: A Protocol to Define the Control Layer

Following the framework's specification, agents are able to interrupt dialogue games with the locution *propose_return_control* in order have the meta-dialogue about dialogues. Figure 4.19 defines the protocol necessary to interrupt the dialogue game and suggest the starting of another. The other agent can *reject* or *accept*. If the agent accepts, they take on their control layer roles.

$$
\begin{array}{l}
a(interrupt\_a, A) \texttt{::=} \\
propose\_return\_control \Rightarrow a(interrupt\_b, B) \texttt{ then} \\
\left(\begin{array}{l} reject(return\_control) \Leftarrow a(interrupt\_b, B) \texttt{ or} \\ \left(\begin{array}{l} accept(return\_control) \Leftarrow a(interrupt\_b, B) \\ \texttt{then } a(control\_layer\_a, A) \end{array} \right) \end{array} \right).
\end{array}
$$

$$
\begin{array}{l}
a(interrupt\_b, B) \texttt{::=} \\
propose\_return\_control \Leftarrow a(interrupt\_a, A) \texttt{ then} \\
\left(\begin{array}{l} reject(return\_control) \Rightarrow a(interrupt\_a, A) \texttt{ or} \\ \left(\begin{array}{l} accept(return\_control) \Rightarrow a(interrupt\_a, A) \\ \texttt{then } a(control\_layer\_b, B) \end{array} \right) \end{array} \right).
\end{array}
$$

Figure 4.19: A Protocol to Return to the Control Layer

With these agent clauses for the control layer and the atomic games (i.e. the dialogue layer) defined, the process of creating complex dialogue games dynamically can be achieved. The following example illustrates how it is done.

4.2.2.2 An Example

The example has the first agent, named prosaically agentA, attempting to persuade another agent to believe the proposition that 'Thursday is the best day for a meeting'. Thus it proposes to play a *persuasion* dialogue game with the other. To simplify the

explanation, the following figures will show the dialogue state of the initiating agent only.

$$
\begin{aligned}
&a(control\_layer\_a, agentA)\texttt{::=}\\
&\quad c(begin(persuasion, thursday\_best\_meeting\_day) \Rightarrow\\
&\qquad a(control\_layer\_b, agentB))\ \texttt{then}\\
&\quad c(accept(persuasion, thursday\_best\_meeting\_day) \Leftarrow\\
&\qquad a(control\_layer\_b, agentB)).
\end{aligned}
$$

Figure 4.20: Control Layer Part of the Dialogue State

Figure 4.20 shows the dialogue state for agentA. At the top, the dialogue shows that two locutions have been communicated. Both the *begin* and *accept* have been communicated. By receiving the accept, the postcondition *startgame(Game,P)*, as specified by figure 4.18, must be satisfied. This provides the agent with a function, whose specific definition remains internal to the agent, with an explicit constraint to perform a transformation. Figure 4.21 shows in bold the result of the transformation. The agent has added the appropriate agent definition for it to initiate an instance of a *persuasion* game. The complementary role is automatically added to the other agent's dialogue state as well.

The dialogue state is returned to the expansion engine and the dialogue continues normally. Figure 4.22 shows the result of this expansion as the agents play the persuasion game. Firstly, the *persuader* agent sends his *know* and *assert* of his proposition, *thursday_best_meeting_day*. The agent then adopts the *defender* role and his dialogical partner chooses to *assert* the negation. Following the protocol, the agent becomes the *challenger* to this new proposition, and communicates the *challenge* locution. In response, the other agent *assert*s a set of supporting propositions. It just happens to be a set of one.

In this example, let us say that *agentA*'s acceptance attitude cannot accept the sup-

$a(control\_layer\_a, agentA)$ **::=**
  $c(begin(persuasion, thursday\_best\_meeting\_day) \Rightarrow$
    $a(control\_layer\_b, agentB))$ **then**
  $c(accept(persuasion, thursday\_best\_meeting\_day) \Leftarrow$
    $a(control\_layer\_b, agentB))$ **then**
a(persuader(thursday_best_meeting_day,agentB),agentA).

Figure 4.21: Persuasion Role added to the Dialogue State

porting argument. If this protocol was static it would communicate the *reject* locution and the game would conclude unsuccessfully. However, let's assume this agent is clever enough to recognize that its rejection is based on a lack of personal knowledge. If the agent could rectify that gap of information, it may be able to accept the argument proposed by *agentB*. This increased complexity in dialogue protocols is a motivation for the authors of [McBurney and Parsons, 2002]. They identify a number of possible combinations of dialogues.

- **Iteration**: This is a repetition of one kind of dialogue that begins upon the immediate conclusion of another.
- **Sequencing**: Similar to iteration but the successive dialogue game need not be of the same type.
- **Parallelization**: Two dialogues are conducted in parallel until both have concluded.
- **Embedding**: During the participation of one game, another is begun until its conclusion at which time the initial game is picked up at the point of interruption and played until it conclusion.
- **Testing**: An odd inclusion in this list and has been ignored as it seems more of a

$a(control\_layer\_a, agentA)$ **::=**

$c(begin(persuasion, thursday\_best\_meeting\_day) \Rightarrow$

$a(control\_layer\_b, agentB))$ **then**

$c(accept(persuasion, thursday\_best\_meeting\_day) \Leftarrow$

$a(control\_layer\_b, agentB))$ **then**

a(persuader(thursday_best_meeting_day,agentB),agentA)::=

$c(know(thursday\_best\_meeting\_day) \Rightarrow$

$a(listener(thursday\_best\_meeting\_day, agentA), agentB))$ **then**

$c(assert(thursday\_best\_meeting\_day) \Rightarrow$

$a(listener(thursday\_best\_meeting\_day, agentA), agentB))$ **then**

$a(defender([thursday\_best\_meeting\_day], agentB), agentA)$ **::=**

$c(assert(not(thursday\_best\_meeting\_day)) \Leftarrow$

$a(challenger([thursday\_best\_meeting\_day], agentB), agentA))$ **then**

$a(challenger([not(thursday\_best\_meeting\_day)], agentA), agentB)$ **::=**

$*$ $c(challenge(thursday\_best\_meeting\_day) \Rightarrow$

$a(defender([thursday\_best\_meeting\_day], agentA), agentB))$ **then**

$c(assert([room4\_unavailable]) \Leftarrow$

$a(defender([thursday\_best\_meeting\_day], agentA), agentB))$ **then**

$a(challenger([room4\_unavailable], agentB), agentA)$ **then**

$a(challenger([], agentB), agentA)$

Figure 4.22: Resulting Dialogues State after Persuasion Role were Added

kind of dialogue rather than a kind of dialogue combination.

In our example thus far we have seen an example of **sequencing**. Figure 4.21 showed the insertion of the agent definition for the agents to play the *persuasion* dialogue at the end of the *control layer* game. The mechanism for introducing an **iteration** transformation is the same. **Parallelization** is the most difficult to address. LCC is capable of parallel operations, but in order to maintain the dialogue state there must always be only one copy being used to maintain consistency. Since this is necessary to make the protocols dynamic, it is a necessary trade off. However, it could be done through parallel but separate execution of the dialogue games. The draw back to this approach is that there would not be one single and unified dialogue state to record this combination such as shown in the figures of the example (e.g. 4.20, 4.21, 4.22, 4.23 and 4.25). The other approach is to adapt the dialogue one locution at a time rather than the whole game at once. This approach is explored in chapter 5. **Embedding** is the situation facing our example. If an instance of an *information seeking* game could be embedded and agentA's ignorance alleviated, the *persuasion* game can be concluded successfully. Figure 4.23 starts at the last message closed and before the agent takes on the role of *a(challenger([room4_unavailable],agentB),agentA)*. Figure 4.22 has an asterisk beside the line that is represented by the series of dots in figures 4.23 and 4.25. The dots are meant to represent the line marked by an asterisk and all the preceding lines in the dialogue state. This is done in order to focus on the current transformations being discussed. For completeness, the entire and complete dialogue state is shown in appendix A.

Figure 4.23 shows the transformation occurring by the assertion of the *interrupt* role which defined in figure 4.19 after the assertion of the supporting proposition. The agent proposes the return to the control level, and the other obliges and they follow the protocol to take on the *control layer* roles. An *information seek*ing game is proposed and accepted. The agent clause for *information seek*ing game are inserted and played to completion. AgentA now has the information necessary to accept the support

$\ldots$

$c(assert([room4\_unavailable]) \Leftarrow$

$a(defender([thursday\_best\_meeting\_day], agentA), agentB))$ **then**

a(interrupt_a,agentA) then

$a(challenger([room4\_unavailable], agentB), agentA)$ **then**

$a(challenger([], agentB), agentA).$

Figure 4.23: Continuation of the Example's Dialogue State

proposition asserted in the *persuasion* dialogue. The embedded control and information seeking dialogue are encapsulated by the large parenthesis in the figure 4.25. The *persuasion* game then also concludes with the *accept* of the assert. Figure 4.24 shows the combination of dialogue games and control layer for this example.

4.2.3 Chapter Summary

As this section has shown, the formal framework for complex dialogue games works well with a distributed dynamic protocol framework. The formalism provides a modular and controlled method for modifying agents' interaction protocols during their dialogues. Using LCC,eases the requirements on the participating agents by having an explicit and computational representation of their expectations for the dialogue (i.e. the encoding of the dialogue game rules as agent clauses). This was also shown to be useful in addressing the ambiguity as found in the use of FIPA ACL's semantics.

This reduction in ambiguity is applicable to any model of communication which can be encoded as LCC clauses. The use of dynamic protocols extends the number of models that can be accommodated. The orthodox view that agent-centric models and protocol led models of communication are exclusive has been blurred. Protocols can be viewed as expectations for interaction rather than static specifications.

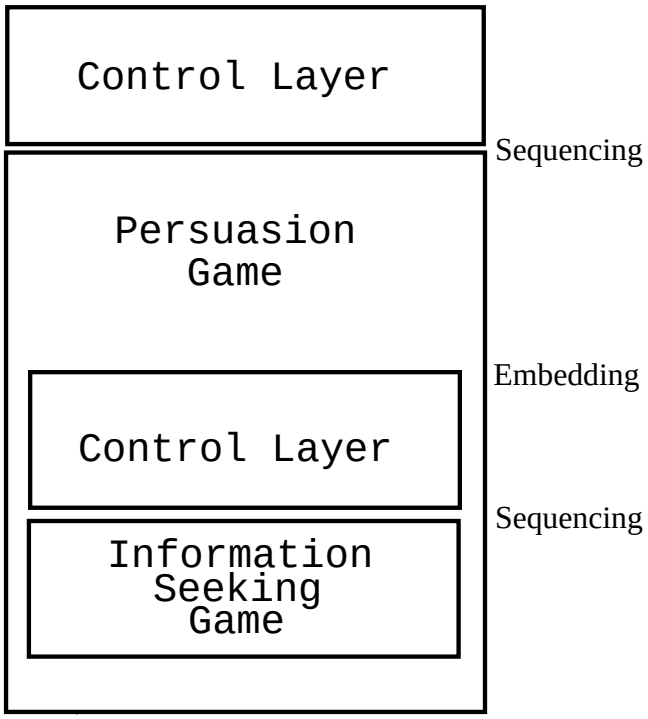

Figure 4.24: *Graphical Representation of the Complex Dialogue Game Example*

By allowing transformations, the agents are free from the static tradition of the protocol led approach, but retain the reliability and accountability associated with that approach. Received protocols can be evaluated but also modified if needed. The following chapter explores the idea of protocol transformation further. Rather than the introduction of agent definitions which are then expanded to a sequence of messages, the agent synthesises 'just enough' protocol to drive the interaction forward.

$\ldots$

$c(assert([room4\_unavailable]) \Leftarrow$

$a(defender([thursday\_best\_meeting\_day], agentA), agentB))$ **then**

$\Big($

a(interrupt_a,agentA)::=

$c(propose\_return\_control \Rightarrow a(interrupt\_b, agentB))$ **then**

$c(accept(return\_control) \Leftarrow a(interrupt\_b, agentB))$ **then**

$a(control\_layer\_a, agentA)$**::=**

$c(begin(info\_seek, room4\_has\_white\_board) \Rightarrow$

$a(control\_layer\_b, agentB))$ **then**

$c(accept(info\_seek, room4\_has\_white\_board) \Leftarrow$

$a(control\_layer\_b, agentB))$ **then**

$a(seeker(room4\_has\_white\_board, agentB), agentA)$**::=**

$c(question(room4\_has\_white\_board) \Rightarrow$

$a(provider(room4\_has\_white\_board, agentA), agentB))$ **then**

$c(assert(room4\_has\_white\_board) \Leftarrow$

$a(provider(room4\_has\_white\_board, agentA), agentB))$ **then**

$a(challenger([room4\_has\_white\_board], agentB), agentA)$**::=**

$c(accept(room4\_has\_white\_board) \Rightarrow$

$a(defender([room4\_has\_white\_board], agentA), agentB))$ **then**

$a(challenger([], agentB), agentA))$**::=** *null* **then**

$\Big)$

$a(challenger([room4\_unavailable], agentB), agentA)$**::=**

$c(accept(room4\_unavailable) \Rightarrow$

$a(defender([room4\_unavailable], agentA), agentB))$ **then**

$a(challenger([], agentB), agentA)$ **then**

$a(challenger([], agentB), agentA).$

Figure 4.25: Continuation of the Example's Dialogue State

Chapter 5

Interaction Protocols by Dialogue Structure Synthesis

The previous chapter explored the consequences and possibilities of having social norms distributed by the interacting agents. Besides having protocol specifications as implementations, we investigated the possibility of introducing more flexibility and dynamism into otherwise static mechanisms of communication. This made it possible for agents to make adaptations by inserting protocol steps for the purpose of addressing run-time dialogical needs. The use of the formal definition provided by the framework for complex dialogue games made this much easier. Embedding and other combinations could be made safely without corrupting or derailing the individual game instances. The previous chapter showed how the use of LCC protocols could facilitate the creation of complex dialogue games.

We concentrated on the use of one particular dialogue game framework [McBurney and Parsons, 2002], but there are others [Amgoud et al., 2000, Lebbink et al., 2003, Maudet and Chaib-draa, 2002, Maudet and Evrard, 1998, Dastani, 2001]. Dialogue games, too, are just one example of using argumentation for multiagent communication, and argumentation itself is one of a number of communicative models. Not all models have a clear and formal definition of how to combine

or modify protocols nor is it straightforward to guarantee that those modifications do not break the protocol as we can when using dialogue games to adapt the protocols like in chapter 4.

Even with the use of dialogue games, a difficulty persists with adaptable protocols. It is not possible to clearly identify or trace the process of adaptation. At the end of the dialogue all that exists is a record of the locutions communicated and their ordering. It is not obvious that an adaptation was made or which adaptation was performed at a given time. This motivated an examination into protocol synthesis. Rather than performing transformations on existing protocols, the entire protocol is developed during the interaction informed by a set of synthesis rules and motivated by the last actions taken in the dialogue. In other words each step of the protocol is adapted. At the end of the interaction the same record of communicated locutions would exist but in addition the agent could identify how the locution step was introduced as well as see any steps which were not taken. This could be done by the examination of the synthesis rules which created the protocol.

Relationships exist between messages regardless of the particular domain with which the messages are concerned. A question usually implies the anticipation of the eventual occurrence of an answer even if the reply is a shrug of the shoulders. This is regardless of whether that answer be to the question of "What time is it?" or "Can you compare and contrast the post-modern interpretations of abstract expressionism to a random sequence of adjectives?" It is these generalised patterns which exist in human communication that we have adopted for our purposes of synthesising protocols in automated communication.

5.1 Using Dialogue Structures

In human dialogue the utterances that the participants make do not occur in isolation. Humans rely on tacit patterns to ground communication. Some have proposed this

is an example of following of certain rules, and others have argued these rules are only descriptions of the process of having a conversation [Searle, 1969a]. Regardless, these patterns can be generalised without concern to the content of the messages. The idea for this approach was largely inspired by the works of [Asher and Gillies, 2003, Searle, 1969a], and the standardisation efforts of Dialogue Structure Theory (DST) for the annotation of human dialogue transcriptions [Core and Allen, 1997].

There are a number of approaches that could be used for the run time synthesis of interaction protocols. Although each has proved its worth for a variety of multiagent applications, each fails in some aspect to provide the unique advantages found by the use of dialogue structures.

Performatives are a common approach for agent communication, and it may be possible to pack pan-dialogical concerns into individual performatives. Yet, this would be an ungainly implementation and an abuse of the spirit of performatives. They are meant to reflect the conditions and effects of a single communicative act rather than the relationships between them or their place within a sequence of message exchanges. In the previous chapter, it was shown how the LCC framework could be used to reproduce the FIPA interaction protocols in such a way that they are not just specifications but 'specifications as implementations' which are communicated by the agents themselves. In addition the individual preconditions and rational effects can be encoded with these protocols to clearly communicate the semantics of the individual performatives used. Also, our concern is more generic than particular performatives in a given ACL. It is our goal to capture the generic structure of conversation that occurs in dialogues regardless of the locution or ACL used, and because this is done using the LCC framework we still retain the abilities described in the previous chapter.

Planning research has been brought to bear on the problem [Rao, 1996, Kakas et al., 2004, Bodenstaff et al., 2006]. Agents use planning techniques to produce an interaction protocol to reach a previously defined goal by means of communicating with other agents. Firstly, communication is not always driven by clearly defined

a priori goals or end states. It is better to view communication as a process. Planning is also presented with the unique challenges of the agency model. Besides planning's reputation for a paucity in terms of data structures, there is also another difficulty in using planners for this purpose. It will be difficult for a planner to produce anything more robust than a look-ahead planner, because of the unpredictability of other agents. The planning agent would constantly be replanning in reaction to others' actions. It would result in a lot of computation without much satisfaction. Even with the help of making assumptions about other agents' rational behaviour, existing approaches still have speed issues for real-time systems. It is for this reason that it would be much more appropriate to have a small set of transformations which the agent can apply mechanically to achieve the same goal. This set defined in terms of a process calculus is exactly what we described in this paper.

Machine learning is also being applied to the many aspects of the agency paradigm. These techniques have been shown to be useful for the purposes of agent communication [Rovatsos, 2004], but the techniques of machine learning introduce a number of unnecessary difficulties. For example, it would be helpful to have transparency and readability of the protocols used by the agents to facilitate human/computer interaction or even humans to understand the protocols used which will assist in the design of new agents. Also, the common problem of producing corpora that hounds machine learning for agency is also a problem in synthesising interaction protocols. Similar to planning approaches, the same goal can be achieved with a set of transformations which can free the agent to spend its computation on learning a strategy for the domain rather than the discussion of that domain.

It is correct to point out the work using social commitments, norms, dialogue games, and other such models of communication provides agents with the ability to reason about communication. Yet, we still retain advantages. To reiterate one of the mantras of this research: it is not the goal to replace any particular model of agency. The goal is to exploit the unique advantages described in chapter 3 provided by the

LCC framework, but to enhance its flexibility. The transformations are purely dialogical in the sense they are generic operations which unfold a single message protocol to a two message protocol which in turn can be used to synthesise a three message protocol, and so on. The agent receiving the synthesised protocol can follow it blindly without needing to understand that its dialogical actions satisfy some commitment, norm, or rule of a dialogue game. This gives us a simple dialogically informed means to drive protocol led communication while maintaining an agents ability to unilaterally explore dialogical options not currently present in a given protocol. The other unique advantage is that not only can an agent generate its expected moves given its model (e.g. norms, commitments, etc.) but it can also communicate its expectations to others. Whereas, most models typically only provide guidance for a single agent and depend on other agents also having the same model of communication to coordinate their conversation. An example described in this chapter will show how synthesis is done with a dialogue game agent for a complex dialogue game.

The details of dialogue structure theory are largely concerned with issues unique to human communication. Our focus on agent communication neatly avoids the most difficult issues associated with this research. DST has been useful for developing metaphors for the development of protocols and protocol synthesis, but its use is superficial. DST, whether used for annotating human dialogue or generating natural language, must concern itself with the minutia and subtleties that software communication does not. All aspects of agent communication are engineered. As a result, there is a regularity, simplicity, and explicitness to it. This artifactual form of communication is not complicated by thousands of years of culture and tradition that complicates human discourse [Rickard, 1989]. Having been saved from the most onerous tasks of DST, we are freed to concentrate on the much more modest task at hand, which is using some basic ideas from the field to drive protocol synthesis.

5.2 Transformations

There are various structures which occur in human dialogue which have a different semantic interpretation but share the same syntactical shape. For example, a question followed by an answer has the same structure as a statement and a confirmation. An agent sends a message which is followed by another message being received. It is therefore useful to generalise the vocabulary of transformations to those whose semantics can be uniquely identified by its syntactic structure. Otherwise a kind of semantic leakage occurs and ambiguity seeps into the dialogue and protocol. The sort of distinctions of a question and answer versus a propose and accept should be dealt with at the ACL level. Our concern is makes no assumptions about the particular locutions used for the protocol.

Throughout this section the protocols are restricted to two party dialogues with no explicit role adoption (i.e. the ability to define recursion). Some multiparty conversations can be treated as a number of dialogues, and for these conversations synthesis is applicable. One of the reasons for this restriction is that the permutations of multiparty protocols increases very quickly as the number of participants increases, making their categorization a daunting task. Using only synthesis of adjacency pairs from the fringe of an existing protocol, we can create every possible protocol given this restriction. By an inductive argument we show that all possible permutations of transformations for a simple protocol can be extended to cover all dialogue LCC protocols. Even after removing all illegal protocol constructions, there are twenty-four possible protocol sequences, shown in figure 5.1 and continued in figure 5.2. This is already too ungainly a number, but the process of pruning away errant transformations is shown in figures 5.3, 5.4, and 5.5.

After we have whittled away the set of transformations, we have a reasonable number of transformations to synthesise all the possible LCC protocols. We have gained a considerable amount dynamism without the loss of any expressivity. It seems all blue skies for us, the two restrictions we made are important and necessary. Firstly, our

Before a Message is Sent

$$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

$$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta \textbf{ then } M2 \Rightarrow \theta$$

$$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta \textbf{ or } M2 \Leftarrow \theta$$

$$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta \textbf{ or } M2 \Rightarrow \theta$$

Before a Message is Received

$$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

$$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta \textbf{ then } M2 \Rightarrow \theta$$

$$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta \textbf{ or } M2 \Leftarrow \theta$$

$$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta \textbf{ or } M2 \Rightarrow \theta$$

Upon Failure of a Sent Message

$$f(M1 \Rightarrow \theta) \longrightarrow f(M1 \Rightarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$

$$f(M1 \Rightarrow \theta) \longrightarrow f(M1 \Rightarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$f(M1 \Rightarrow \theta) \longrightarrow f(M1 \Rightarrow \theta) \textbf{ or } M2 \Leftarrow \theta$$

$$f(M1 \Rightarrow \theta) \longrightarrow f(M1 \Rightarrow \theta) \textbf{ or } M2 \Rightarrow \theta$$

Upon Having Sent a Message

$$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$

$$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta) \textbf{ or } M2 \Leftarrow \theta$$

$$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta) \textbf{ or } M2 \Rightarrow \theta$$

Figure 5.1: The Twenty-four Possible Transformations

Upon Failure of a Received Message

$$f(M1 \Leftarrow \theta) \quad \longrightarrow \quad f(M1 \Leftarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$

$$f(M1 \Leftarrow \theta) \quad \longrightarrow \quad f(M1 \Leftarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$f(M1 \Leftarrow \theta) \quad \longrightarrow \quad f(M1 \Leftarrow \theta) \textbf{ or } M2 \Leftarrow \theta$$

$$f(M1 \Leftarrow \theta) \quad \longrightarrow \quad f(M1 \Leftarrow \theta) \textbf{ or } M2 \Rightarrow \theta$$

Upon Having Received a Message

$$c(M1 \Leftarrow \theta) \quad \longrightarrow \quad c(M1 \Leftarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$

$$c(M1 \Leftarrow \theta) \quad \longrightarrow \quad c(M1 \Leftarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$c(M1 \Leftarrow \theta) \quad \longrightarrow \quad c(M1 \Leftarrow \theta) \textbf{ or } M2 \Leftarrow \theta$$

$$c(M1 \Leftarrow \theta) \quad \longrightarrow \quad c(M1 \Leftarrow \theta) \textbf{ or } M2 \Rightarrow \theta$$

Figure 5.2: The Twenty-four Possible Transformations [cont'd]

restriction to dialogues does restrict the class of interactions we can describe, but explained in the previous chapter the was necessary. However this does not diminish the thesis' contribution greatly, because we can still consider multi-party interactions that are capable of being serialised. The other restriction is the removal of role adoption from the LCC language. This does affect the language's ability to explicit represent roles and role changes. Role adoption was a powerful means for the protocol designer to recurse over a set. The dialogue game protocols defined in the previous chapters all used this to enable the participants to evaluate a set of propositions. The actual ability to express any LCC sequence of messages has not been impaired. Instead, the onus now lies with the agent to be able to synthesise the correct steps and retain role information internally rather than rely on the protocol to define them.

The set of transformations in figures 5.1 and 5.2 was created by taking all the possible permutations of the two message protocol given an atomic protocol. An atomic protocol is defined as a single message being communicated, as a more simpler (non-empty) protocol can not be conceived. To reiterate the syntax we are using, M is the message being sent or received. A number following the M serves to distinguish the

two messages as they are not necessarily the same. A right facing arrow, '⇒', shows a message that is outgoing while an incoming message is shown by the '⇐' style arrow. The 'θ' represents the definition of the agent's partner (i.e. either the source or the recipient of the message). The single message is marked as open, closed or failed, open being defined as unsent or unevaluated. Each of these single message protocols can be expanded to a two message protocol by the addition of a **then** or an **or** operator followed by another message either incoming or outgoing. The total number of these two message protocols is seventy-two[1]. By excluding protocols not possible within the LCC framework, the set is thinned to the twenty-four possible transformations of figures 5.1 and 5.2. For example a protocol cannot exist that has a closed message following an open one (e.g. $M1 \Leftarrow \theta$ **then** $c(M2 \Rightarrow \theta)$). This is because of the way the protocol is expanded by the LCC framework. The message *M2* could not be expanded unless *M1* had been expanded. If the conditions of *M1* were satisfied, it too would be marked closed. If the conditions were not satisfiable, the message would not be marked closed but this would also make it impossible by the definition of the expansion rules for the **then** operator to expand and close *M2*. Therefore this particular protocol sequence will never exist and can be ignored when considering synthesis rules.

Figure 5.3 shows the set after the first pruning. The set of twenty-four sheds six:

$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta$ **or** $M2 \Leftarrow \theta$
$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta$ **or** $M2 \Rightarrow \theta$
$f(M1 \Rightarrow \theta) \longrightarrow f(M1 \Rightarrow \theta)$ **or** $M2 \Leftarrow \theta$
$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta)$ **or** $M2 \Leftarrow \theta$
$f(M1 \Leftarrow \theta) \longrightarrow f(M1 \Leftarrow \theta)$ **or** $M2 \Rightarrow \theta$
$c(M1 \Leftarrow \theta) \longrightarrow c(M1 \Leftarrow \theta)$ **or** $M2 \Rightarrow \theta$

The transformations which have both an outgoing message and an incoming message disjointed with an **or** operator cannot exist due to the nature of the 'hot potato' coordination of LCC. Although not strictly illegal with respect to syntax, there will never be the ambiguity of whose turn it is to speak. This is due to our limitation on protocol synthesis for dialogues only. Protocols for multiparty conversations could in-

[1]The full set of seventy-two is not shown as their enumeration would be tedious and uninformative

Before a Message is Sent

$$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

$$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta \textbf{ then } M2 \Rightarrow \theta$$

$$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta \textbf{ or } M2 \Rightarrow \theta$$

Before a Message is Received

$$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

$$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta \textbf{ then } M2 \Rightarrow \theta$$

$$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta \textbf{ or } M2 \Leftarrow \theta$$

Upon Failure of a Sent Message

$$f(M1 \Rightarrow \theta) \longrightarrow f(M1 \Rightarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$

$$f(M1 \Rightarrow \theta) \longrightarrow f(M1 \Rightarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$f(M1 \Rightarrow \theta) \longrightarrow f(M1 \Rightarrow \theta) \textbf{ or } M2 \Rightarrow \theta$$

Upon Having Sent a Message

$$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$

$$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta) \textbf{ or } M2 \Rightarrow \theta$$

Upon Failure of a Received Message

$$f(M1 \Leftarrow \theta) \longrightarrow f(M1 \Leftarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$

$$f(M1 \Leftarrow \theta) \longrightarrow f(M1 \Leftarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$f(M1 \Leftarrow \theta) \longrightarrow f(M1 \Leftarrow \theta) \textbf{ or } M2 \Leftarrow \theta$$

Upon Having Received a Message

$$c(M1 \Leftarrow \theta) \longrightarrow c(M1 \Leftarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$

$$c(M1 \Leftarrow \theta) \longrightarrow c(M1 \Leftarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$c(M1 \Leftarrow \theta) \longrightarrow c(M1 \Leftarrow \theta) \textbf{ or } M2 \Leftarrow \theta$$

Figure 5.3: After the First Pruning

deed have this ambiguity. Also, if an agent requires its partner to speak it can apply the appropriate transformation to the protocol creating a prompt for response and pass the protocol to the other.

Failure is defined as both the inability to communicate at the semantic level (i.e. the message was sent and received but not sensible with respect to an agent's knowledge base) as well as the physical failure to send or receive a message. Either a message being received or a message being sent is considered failed. When the failed message is outgoing. The sending agent has marked the failure in the dialogue state (i.e. that agent knows about the failure). What should be done? He could synthesise a protocol to have an incoming message come in to address that failure,

$$f(M1 \Rightarrow \theta) \quad \longrightarrow \quad f(M1 \Rightarrow \theta) \ \textbf{then}\ M2 \Leftarrow \theta$$

This would be incorrect for two reasons. The first reason is the other agent does not have the protocol. By the definition of the framework for use in dialogues, the agent who marked the protocol is the one expected to communicate. The other reason is that the other agent may or may not be aware of the failure and it would be an odd protocol which expected agents to correct failures unknown to them. Another possibility for a failed outgoing message is:

$$f(M1 \Rightarrow \theta) \quad \longrightarrow \quad f(M1 \Rightarrow \theta) \ \textbf{or}\ M2 \Rightarrow \theta$$

It is possible to synthesise such a protocol, but there are practical problems. Firstly, as the LCC framework is defined this would fail to expand. Even if the framework was modified to handle this protocol, and the expansion of the protocol would recognised the failure of '$M1 \Rightarrow \theta$', backtrack, and attempt to expand the righthand side of the **or** operator. The problem is that the recording of the failed message would be lost as only the one message of the **or**, which was chosen to be communicated, is recorded in the dialogue state. The reasoning is similar for the dismissal of the protocols synthesised from a failed received message.

$$f(M1 \Leftarrow \theta) \quad \longrightarrow \quad f(M1 \Leftarrow \theta) \ \textbf{then}\ M2 \Leftarrow \theta$$
$$f(M1 \Leftarrow \theta) \quad \longrightarrow \quad f(M1 \Leftarrow \theta) \ \textbf{or}\ M2 \Rightarrow \theta$$

Before a Message is Sent

$$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

$$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta \textbf{ then } M2 \Rightarrow \theta$$

$$M1 \Rightarrow \theta \longrightarrow M1 \Rightarrow \theta \textbf{ or } M2 \Rightarrow \theta$$

Before a Message is Received

$$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

$$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta \textbf{ then } M2 \Rightarrow \theta$$

$$M1 \Leftarrow \theta \longrightarrow M1 \Leftarrow \theta \textbf{ or } M2 \Leftarrow \theta$$

Upon Failure of a Sent Message

$$f(M1 \Rightarrow \theta) \longrightarrow f(M1 \Rightarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

Upon Having Sent a Message

$$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$

$$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$c(M1 \Rightarrow \theta) \longrightarrow c(M1 \Rightarrow \theta) \textbf{ or } M2 \Rightarrow \theta$$

Upon Failure of a Received Message

$$f(M1 \Leftarrow \theta) \longrightarrow f(M1 \Leftarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

Upon Having Received a Message

$$c(M1 \Leftarrow \theta) \longrightarrow c(M1 \Leftarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$

$$c(M1 \Leftarrow \theta) \longrightarrow c(M1 \Leftarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$c(M1 \Leftarrow \theta) \longrightarrow c(M1 \Leftarrow \theta) \textbf{ or } M2 \Leftarrow \theta$$

Figure 5.4: After the Second Pruning

Before a Message is Sent

$$M1 \Rightarrow \theta \quad \longrightarrow \quad M1 \Rightarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

$$M1 \Rightarrow \theta \quad \longrightarrow \quad M1 \Rightarrow \theta \textbf{ then } M2 \Rightarrow \theta$$

$$M1 \Rightarrow \theta \quad \longrightarrow \quad M1 \Rightarrow \theta \textbf{ or } M2 \Rightarrow \theta$$

Before a Message is Received

$$M1 \Leftarrow \theta \quad \longrightarrow \quad M1 \Leftarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

$$M1 \Leftarrow \theta \quad \longrightarrow \quad M1 \Leftarrow \theta \textbf{ then } M2 \Rightarrow \theta$$

$$M1 \Leftarrow \theta \quad \longrightarrow \quad M1 \Leftarrow \theta \textbf{ or } M2 \Leftarrow \theta$$

Upon Failure of a Sent Message

$$f(M1 \Rightarrow \theta) \quad \longrightarrow \quad f(M1 \Rightarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

Upon Failure of a Received Message

$$f(M1 \Leftarrow \theta) \quad \longrightarrow \quad f(M1 \Leftarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

Upon Having Received a Message

$$c(M1 \Leftarrow \theta) \quad \longrightarrow \quad c(M1 \Leftarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

Figure 5.5: After the Final Pruning of Transformations

Therefore, the only possible transformation which should be applied is the sending of a second message, a *correction*. Figure 5.4 shows the result of another four transformations being cast away by this second pruning.

It is not possible to make a transformation on the closed atomic protocol of a single sent message. By the definition of LCC, the message has already been communicated and with it the protocol one wishes to transform. For this reason, we can dismiss any protocol synthesised upon a closed outgoing message such as these:

$$c(M1 \Rightarrow \theta) \quad \longrightarrow \quad c(M1 \Rightarrow \theta) \textbf{ then } M2 \Leftarrow \theta$$
$$c(M1 \Rightarrow \theta) \quad \longrightarrow \quad c(M1 \Rightarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$
$$c(M1 \Rightarrow \theta) \quad \longrightarrow \quad c(M1 \Rightarrow \theta) \textbf{ or } M2 \Rightarrow \theta$$

This point, $c(M1 \Rightarrow \theta)$, in the dialogue state occurs after the agent has evaluated, made its decision with respect to the conversation, and has expanded the protocol and only just before the message with the protocol and the dialogue state are sent to the

other agent. It would be too late to synthesise more protocol steps, because having sent the message and the protocol, the agent no longer possesses the protocol to change it. The situation is different for an incoming message which has been closed. The agent has just received the message. It has marked the message as closed and is at the point to make a decision, and can respond or synthesise additional protocol steps.

For a closed received message the only transformation which can be applied is the addition of an outgoing message. As the agent has just received the message and protocol and it should not be able to synthesise more steps for the dialogical partner that it won't be able to know about.

$$c(M1 \Leftarrow \theta) \longrightarrow c(M1 \Leftarrow \theta)\ \textbf{then}\ M2 \Leftarrow \theta$$
$$c(M1 \Leftarrow \theta) \longrightarrow c(M1 \Leftarrow \theta)\ \textbf{or}\ M2 \Leftarrow \theta$$

Five more transformations can be scratched from the list leaving a more manageable nine shown in figure 5.5. Figure 5.5 now shows the exhaustive set of the only possible syntactic transformations from an atomic protocol to one with two steps. Given all the possible two step protocols, one can apply the transformations to each of those and have all the permutations of a three step protocol, and in turn apply the transformations again to have all four step protocols. This can be done indefinitely in order to represent all possible protocols that can written an LCC agent clause. There is no universally acceptable model of conversation, but we can map phenomena from linguistics research to the identified transformations. This does not imply that the transformation is a canonical match to the phenomena in human dialogue, but the mappings have an easily identifiable similarity. This is how figure 5.6 is derived from figure 5.5.

In dialogues, humans cue for response by a number of verbal and non-verbal cues. This is captured by the two transformations with their transitions labelled *response* in figure 5.6. A message is sent to an agent and, at some point later, a message is received from the same agent. The messages and their content can be said to be a *response*.

During discussions, humans will provide choice to their dialogical partners when appropriate. In the example below, the speaker defines the allowable response set.

Before a Message is Sent

$$M1 \Rightarrow \theta \xrightarrow{response(M1,M2)} M1 \Rightarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

$$M1 \Rightarrow \theta \xrightarrow{continuation(M1,R2)} M1 \Rightarrow \theta \textbf{ then } M2 \Rightarrow \theta$$

$$M1 \Rightarrow \theta \xrightarrow{counter(M1,M2)} M1 \Rightarrow \theta \textbf{ or } M2 \Rightarrow \theta$$

Before a Message is Received

$$M1 \Leftarrow \theta \xrightarrow{continuation(M1,M2)} M1 \Leftarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

$$M1 \Leftarrow \theta \xrightarrow{response(M1,M2)} M1 \Leftarrow \theta \textbf{ then } M2 \Rightarrow \theta$$

$$M1 \Leftarrow \theta \xrightarrow{counter(M1,M2)} M1 \Leftarrow \theta \textbf{ or } M2 \Leftarrow \theta$$

Upon the Reception of a Message

$$c(M1 \Leftarrow \theta) \xrightarrow{clarification(M1,M2)} c(M1 \Leftarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

Upon Failure of a Message

$$f(M1 \Rightarrow \theta) \xrightarrow{correction(M1,M2)} f(M1 \Rightarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

$$f(M1 \Leftarrow \theta) \xrightarrow{correction(M1,M2)} f(M1 \Leftarrow \theta) \textbf{ then } M2 \Rightarrow \theta$$

Figure 5.6: The Vocabulary of Transformations

"500 dollars for the set. Take it or Leave it."

The hearer can respond by *taking* the offer, but the speaker has also provided a counter to that response (i.e. allowing the hearer to also *leave* it and reject the offer). It other situations where power dynamics differ or the initial speech act is a command a *counter* might not be appropriate and the hearer's only allowable response is to acquiesce to the will of the speaker.

This same need exists in agent communication. The *counter* transformation allows agents to introduce this type of step in dialogues. Here we have a departure from the phenomenon occurring in human dialogue, versus agent interaction protocols. Rarely in human dialogue are the options for response so explicitly stated as in our example. In agent communication it is not only common, but usually necessary.

Another feature of human dialogues is the use of cues to signify the speaker wishes to continue their turn in the dialogue. In the example below, the speaker was not finished with the enumerations of the trespasses committed by the person referred to in the sentence.

A:"She ate the whole cake. And you know what else? She didn't even say thanks."
B:"The nerve."

The speaker signalled this by the "And you know what else" phrase and then stated a further transgression performed by the subject of conversation. The *continuation* transformation enables software agents to do the same. The protocol coordinates whose turn it is to speak and an agent wishing to communicate more than one locution would not need a signalling phrase usually required for polite human dialogue but instead have a protocol allowing the multiple messages to be communicated.

Clarifications and Corrections are of great interest to those studying dialogue structures [Ginzburg, 1996, Asher and Gillies, 2003]. Corrections are usually reactions to failures in the dialogue. We have addressed outright failures such as message loss or complete misunderstanding as criteria for a *correction* transformation. Whereas,

clarifications occur when a message received is understood but found to be wanting in detail. An agent providing a date but the other agent needs a year for the date as well *clarification* versus an agent communicating a seemingly erroneous date such as the tenth day of the seventeenth month *correction*. The message encapsulated by a 'c' before the *clarification* transformation represents in the protocol language that the message has been sent. The 'f' encapsulation represents a message failure which is the requirement for an agent making a *correction* transformation. Clarifications and Corrections have a unique property as they could be said to be attempts to undo a dialogical action that has occurred or modify the common knowledge created during the dialogue. Whereas responses, counters, and continuations are concerned with only the structuring of locutions with respect to other locutions in a generic sense. Clarifications and corrections concern particular instances of messages and in particular messages that have been communicated. It is therefore necessary to have some δ function which can be used to address the content that is being corrected or clarified. For example, a *clarification* of a message by another may in some instance require a smaller set than previously discussed as in this exchange.

A:"Get the thingie."
B:"The thingie?" { *clarification* }
A:"The wrench." { *response* }

The set of thing(ie)s is reduced to the set of wrenches, but it not possible to make a more broad claim that in all circumstances the result of a *clarification* is a smaller set. Therefore the agent would define its δ in such a way that would ensure that the result of a *clarification* is indeed a smaller set. A *correction* is more of a revision. It's point is to nullify some proposition stated by a previous speech act. In this conversation snippet, there are two corrections.

A:"Hand me the wrench."
B:"You mean a Hammer. You need a hammer to hammer a nail." { *correction* }
A:"I know, but the hammer exploded yesterday." {*correction* }

B corrects **A**'s request for the wrench because he knows the hammering of a nail requires a hammer, but **B** does not know the extraordinary fate of the required tool and **A** must correct **B**'s correction. **A** knew that a hammer is for hammering, but also that a wrench used unconventionally could be employed as a substitute.

The transformations described are as generic as the LCC framework. There is no assumption of the rational make up of agents, the ACL involved or the domain ontology. In order for the transformations to make sense for a particular domain, it is necessary to define specific instances of the dialogue structures with respect to the domain being discussed and the locutions being communicated. These serve as synthesis rules. They dictate for the agent what is considered the correct responses, counters, continuations, corrections or clarifications given the ACL and domain of the dialogue.

The rules go one more level of detail. The transformations of figure 5.6 give the generic syntax for synthesising protocols. The rules tie those generic structures to the specific domain and set of locutions. For example, in a *response* the protocol has two messages, one coming in and one going out, separated by the **then** operator. The synthesis rules for the agent say just what locution can be used for a *response* transformation.

response(ask(X),tell(X)**)**.

The synthesis rule above says that the proper *response* for an *ask* locution is a *tell* and their content is the same. Given this synthesis rule, if the agent, we'll call him 'agentA', has a protocol which is just the sending of an ask to agentB, written as

$$agent(Proposition, agentA) ::= ask(Proposition) \Rightarrow agent(Proposition, agentB).$$

then we can synthesise a two step protocol which provides the protocol step to allow agentB to respond.

$$agent(Proposition, agentA) ::= ask(Proposition) \Rightarrow agent(Proposition, agentB)$$
$$\textbf{then}\ tell(Proposition) \Leftarrow agent(Proposition, agentB).$$

As synthesis proceeds using the standard LCC framework described in chapter 3, a boot strapping role and initial message to be sent is still used. For synthesis, the role is

ignored, but is useful for the agent's internal decision process to determine the locution to initiate the dialogue, as well as any initial variables needed.

We take advantage of the common knowledge mechanism in LCC to communicate the synthesis rules. This provides a public representation of the rules that synthesised the protocol and the ability for other agents to employ the rules for synthesising.

How an agent determines which rule is applicable is partly defined by the rule itself, but it also depends on the agent to be able to reason about the rules. The synthesis engine restricts transformations. Figure 5.7 gives an example of two agent clauses of an LCC protocol with five synthesis rules that the agent can employ. *agentA* has yet to send the *ask*. The synthesis engine only allows one transformation to be applied, $response(ask(X) \leftarrow need(X), tell(X))$. The agent still has the freedom to choose whether it wants to perform this transformation, but it will not be allowed to apply any of the others. The $response(ask(X), tell(X))$ rule cannot be used because the first half, $ask(X)$ does not exactly match the existing protocol, $ask(X) \leftarrow need(X)$. The rules $clarification(X, ask(X))$ and $correction(X, ask(X))$ are not usable as defined by transformations the locution on which the synthesis is occurring must be closed or failed. The $response(tell(X), confirm(X))$ can not be used because it requires a $tell(X)$ in the protocol. After $response(ask(X) \leftarrow need(X), tell(X))$ synthesis rule is applied, it will be appropriate.

We now turn to describe the engine that will drive the synthesis, and ensure that the synthesised protocols maintain symmetric clauses. As stated before, the preservation of symmetry in the agents' clauses is essential to avoiding deadlock and failure.

5.3 Synthesising Protocols

The process of synthesis progresses upon the last message in the protocol. This is to prevent transformations such as figure 5.8. The two responses are performed with respect to the first message, *M1*. This could go on indefinitely as the agent repeatedly

$$a(\_, agentA) \texttt{::=} ask(X) \leftarrow need(X) \Rightarrow a(\_, agentB).$$

$$a(\_, agentB) \texttt{::=} ask(X) \Leftarrow a(\_, agentA).$$

$$response(ask(X), tell(X)).$$
$$response(ask(X) \leftarrow need(X), tell(X)).$$
$$clarification(X, ask(X)).$$
$$correction(X, ask(X)).$$
$$response(tell(X), confirm(X)).$$

Figure 5.7: Choosing which Synthesis Rules

applies synthesis rules with respect to *M1*.

$$M1 \Rightarrow \theta \xrightarrow{response(M1,M2)} M1 \Rightarrow \theta \texttt{ then } M2 \Leftarrow \theta$$

$$M1 \Rightarrow \theta \texttt{ then } M2 \Leftarrow \theta \xrightarrow{response(M1,M3)} M1 \Rightarrow \theta \texttt{ then } M3 \Leftarrow \theta \texttt{ then } M2 \Leftarrow \theta$$

Figure 5.8: An illegal Transformation

This is avoided by stepping forward to the last step of the protocol synthesised, and evaluating whether there are any synthesis rules to apply for that message. This way the protocol continues to expand but only in one direction, forward. The synthesis engine maintains a single thread of dialogue rather than attempting to expand all potential conversations. Not only is this more computationally viable as you avoid state explosion and replication, it is more inline with how dialogues work as participants

act and react to one another's contribution to the conversation. This is done by halting further synthesis after the application of any *counter* rules (i.e. The addition of the **or** operator and another message). When the agent has made its choice and the dialogue state has only one thread, synthesis can once again occur.

The does not restrict the set of protocols that can be expressed. Any ordering of messages expressed in LCC can still be expressed using the synthesis engine. This is due to our exhaustive set of meaningful transformations which can be performed iteratively to create the necessary protocols. The protocol in figure 5.8 is no exception. What made that protocol illegitimate was its construction, not its structure. Figure 5.9 shows a revision of the protocol's synthesis. The rules and their order of application are different but the resulting protocol is the same.

$$M1 \Rightarrow \theta \xrightarrow{response(M1,M3)} M1 \Rightarrow \theta \textbf{ then } M3 \Leftarrow \theta$$

$$M1 \Rightarrow \theta \textbf{ then } M2 \Leftarrow \theta \xrightarrow{continuation(M3,M2)} M1 \Rightarrow \theta \textbf{ then } M3 \Leftarrow \theta \textbf{ then } M2 \Leftarrow \theta$$

Figure 5.9: Revision of the Synthesis of Figure 5.8

LCC deals with meta-dialogical (e.g. deontic) concerns in a number of ways, one of which is the use of constraints. The use of constraints also deals context-dependent dialogical issues. The use of constraints with the synthesis rules also provides this functionality. The synthesis rules can be expressed in such a way to convey context-sensitive locutions. For example, a synthesis rule can be written like this:

response($ask(X), tell(X) \leftarrow hasPrivileges(X, \theta)$**)**.

This could be described as the proper response to an *ask* about 'X' is a *tell* about 'X' but only if the agent θ has privileges to that information. The synthesis engine puts

the constraint in the appropriate agent clause in accordance with the syntactical rules of LCC. Take the revised *response* rule below.

$$\textbf{response(}ask(X), known(X) \leftarrow tell(X) \leftarrow hasPrivileges(X, \theta)\textbf{)}.$$

By the definition of LCC, the construction of a constraint on the left hand side of the $\leftarrow$ may only exist upon a received message (e.g. $M1 \Leftarrow \theta$) and having a constraint on the right hand side is for outgoing messages (e.g. $M1 \Rightarrow \theta$). Since this is the case, it is unambiguous for the synthesis engine to place the message and constraint onto the correct agent's clause. Figure 5.10 shows the resulting agent clauses after the *response* rule is applied and the protocol is synthesised.

$$a(\_, agentA)\texttt{::=}$$
$$\quad ask(X) \Rightarrow a(\_, agentB)\ \texttt{then}$$
$$\quad\quad known(X) \leftarrow tell(X) \Leftarrow a(\_, agentB).$$

$$a(\_, agentB)\texttt{::=}$$
$$\quad ask(X) \Leftarrow a(\_, agentA)\ \texttt{then}$$
$$\quad\quad tell(X) \Rightarrow a(\_, agentA) \leftarrow hasPrivileges(X, agentA).$$

Figure 5.10: The Placing of Constraints

With hopefully a sufficient explanation of the idea of synthesising protocols let us turn to its practice for agent communication. Once again, dialogue games are a useful model to implement.

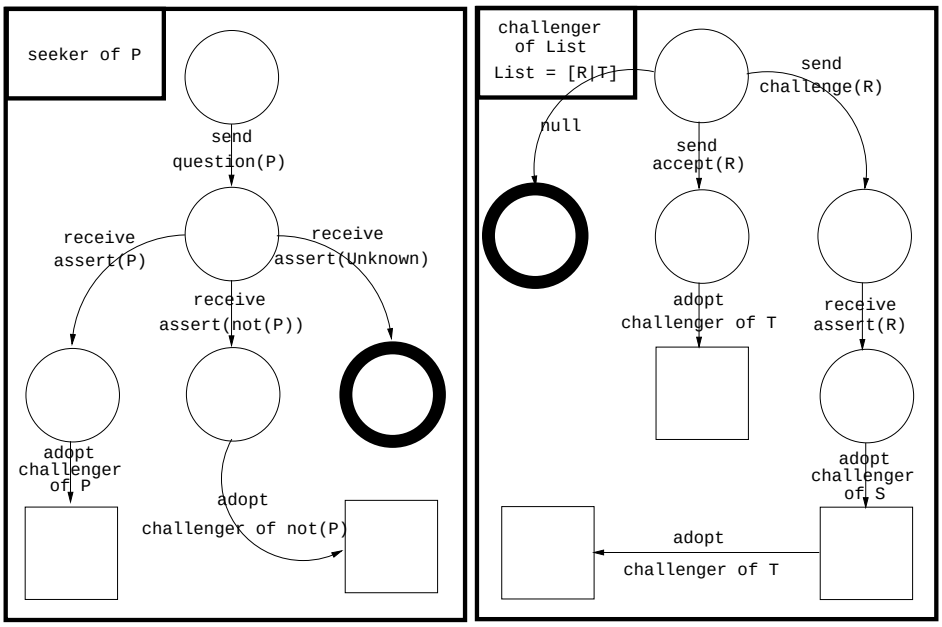

Figure 5.11: *Graphical Representation of the Information seeking Game*

5.4 An Example Using Dialogue Games for Synthesis

This section will use the Information-Seeking dialogue game and show how simple manipulations of the set of synthesis rules provide versatility to the protocol based approach to dialogue. Figure 5.11 is a graphical representation of the protocol used. As a reminder, circles signify states, bold circles are final states and squares are the definition of roles. The arcs are the occurrence of messages or the adoption of roles. Figure 5.12 and 5.13 show the information seeking game protocol developed previously in chapter 4.

Previously, dynamic protocols were achieved by splicing the roles that defined the pariticpant's part in a dialogue game instance into another dialogue game protocol. This created on-the-fly dialogue game combinations from the atomic game protocols. This example will focus on a single game, but the techniques are readily applicable to game combinations. The synthesis approach requires a more robust synthesising agent. Since synthesis is defined in terms of purely dialogical terms (i.e. the relationship between incoming and outgoing messages), there is a loss of meta-dialogical expressivity

$$
\begin{array}{l}
a(seeker(P,B),A)\texttt{::=} \\
question(P) \Rightarrow a(provider(P,A),B)\ \texttt{then} \\
\left(\begin{array}{l} assert(P) \Leftarrow a(provider(P,A),B)\ \texttt{then} \\ a(challenger([P],B),A) \end{array}\right)\ \texttt{or} \\
\left(\begin{array}{l} assert(not(P)) \Leftarrow a(provider(P,A),B)\ \texttt{then} \\ a(challenger([not(P)],B),A) \end{array}\right)\ \texttt{or} \\
assert(\mathcal{U}) \Leftarrow a(provider(P,A),B).
\end{array}
$$

$$
\begin{array}{l}
a(provider(P,A),B)\texttt{::=} \\
question(P) \Leftarrow a(seeker(P,B),A)\ \texttt{then} \\
\left(\begin{array}{l} assert(P) \Rightarrow a(seeker(P,B),A)\ \texttt{then} \\ a(defender([P],A),B) \end{array}\right)\ \texttt{or} \\
\left(\begin{array}{l} assert(not(P)) \Rightarrow a(seeker(P,B),A)\ \texttt{then} \\ a(defender([not(P)],A),B) \end{array}\right)\ \texttt{or} \\
assert(\mathcal{U}) \Rightarrow a(seeker(P,B),A).
\end{array}
$$

Figure 5.12: A Protocol for an Information Seeking Dialogue Game

$$
\begin{array}{l}
a(challenger(List,B),A)\,\mathbf{::=} \\
null \leftarrow (List=[])\ \mathbf{or} \\
\left(\begin{array}{l}
accept(R) \Rightarrow a(defender(List,A),B) \leftarrow (List=[R|T])\ \mathbf{then} \\
a(challenger(T,B),A))
\end{array}\right)\ \mathbf{or} \\
\left(\begin{array}{l}
challenge(R) \Rightarrow a(defender(List,A),B) \leftarrow (List=[R|T])\ \mathbf{then} \\
assert(S) \Leftarrow a(defender(List,A),B)\ \mathbf{then} \\
a(challenger(S,B),A)\ \mathbf{then}\ a(challenger(T,B),A)
\end{array}\right).
\end{array}
$$

$$
\begin{array}{l}
a(defender(List,A),B)\,\mathbf{::=} \\
null \leftarrow (List=[])\ \mathbf{or} \\
\left(\begin{array}{l}
(List=[R|T]) \leftarrow accept(R) \Leftarrow a(challenger(List,B),A)\ \mathbf{then} \\
a(defender(T,A),B)
\end{array}\right)\ \mathbf{or} \\
\left(\begin{array}{l}
(List=[R|T]) \leftarrow challenge(R) \Leftarrow a(challenger(List,B),A)\ \mathbf{then} \\
assert(S) \Rightarrow a(challenger(List,B),A) \leftarrow support(R,S)\ \mathbf{then} \\
a(defender(S,A),B)\ \mathbf{then}\ a(defender(T,A),B)
\end{array}\right).
\end{array}
$$

Figure 5.13: The Recursive Roles for the Information Seeking Dialogue Game Protocol

like explicit role adoption. As such, synthesis results in more tolerant agent clauses. A protocol is said to be more tolerant if it has a more liberal definition of the conversation space. The more tolerant the protocol definition, the more the agent is required to decide for itself the appropriate action to take. In other words, there is a large number of possible paths the participants can take and still be within the protocol. The least tolerant protocols are the more orthodox ones typified by electronic institutions. There is a strict ordering of messages to be exchanged as well as norms to which must be adhered. This is not to say that there is any loss of expressivity for the exchange of messages or combination of messages. Any legal LCC sequence of messages can also be expressed using synthesis.

a) **response(**$question(P), assert(P)$**)**.

b) **counter(**$assert(P), assert(not(P))$**)**.

c) **counter(**$assert(not(P)), assert(unknown)$**)**.

d) **response(**$assert(R), accept(R)$**)**.

e) **counter(**$accept(R), challenge(R)$**)**.

f) **response(**$challenge(R), assert(S) \leftarrow support(R,S)$**)**.

g) **response(**$assert(S), accept(R) \leftarrow memberOf(R,S)$**)**.

h) **counter(**$accept(R), challenge(R) \leftarrow memberOf(R,S)$**)**.

i) **continuation(**$accept(R), accept(T) \leftarrow memberOf(T,S)$ **and** $R \neq T$**)**.

Figure 5.14: Synthesis Rules for an Information Seeking Game

Figure 5.14 shows our set of synthesis rules which can reproduce the dialogue game protocol in figures 5.12 and 5.13. Although we will step through the protocol from the perspective of the initiator of the dialogue (i.e. the seeker) the protocol synthesised produces the symmetric clause for the dialogical partner.

This particular set of rules was derived directly from the protocols written in chapter 4, and repeated in figures 5.12 and 5.13. This makes the task much easier as the

interpretation of the game description is already defined in terms of LCC. Choosing the synthesis rule which is to be used to recreate a segment of protocol becomes a simple matching task between the protocol and the vocabulary of transformations shown in figure 5.6.

It is a more difficult task to go from a semi-formal game description to a set of synthesis rules. However it isn't very different from the process of writing static LCC protocols. It is a trial and error process. This is a problem in general with the use of protocols in agent communication. There needs to be more formal analysis and model checking. The "specification as implementation" approach described here is a step in the right direction as it makes analysis easier to do.

Rule *a* in figure 5.14 is the synthesis rule for a *response* producing a message going out, *question(P)*, followed by a message coming in, *assert(P)*, separated by the operator ' `then` '. According to the protocol an agent could also respond with an *assert* of the negation as well as asserting *unknown*. A protocol can also be synthesised with these steps by rules *b* and *c*. In the original protocol the assertion of *unknown* ended the conversation. This strictness is not preserved by the response rules of *d*. A more specific rule could have been defined to disallow this step such as:

d') **response(**$assert(R), accept(R) \leftarrow R \neq unknown$**)**.

Without the constraint, the synthesis' laxity is due to the uniqueness and transitivity of variables in the synthesis rules. This uniqueness is because the variables in the individual rules only refer to the same variable within that rule. Their scope does not extend beyond that rule. The *P* in rule *a* is not the same *P* as in rule *b*. What does make them the same is that if the rules are applied iteratively and the *assert(P)* of the second part of rule *a* is the *assert(P)* of the first part of rule *b*. By transitivity *P* becomes the same throughout both transformations. This is why one could apply rule *d* to an *assert(unknown)* message. Such flexibility puts the burden on the agent not to perform such an operation if it is deemed to be prohibited. It also becomes a weakness when wanting to define recursive, or persistent meta-dialogical behaviour.

Not only can an agent *accept* an assertion but it should be able to *challenge* one. Rule *e* enables that. Rules *f*, *g*, and *h* provide an example of how to constrain the transformations given some condition in the conversation. An agent can respond to a challenge with an assertion of the grounds for the argument as long as it can satisfy the constraint that those grounds are the support for the proposition that is challenged. Rules *d* and *e* deal with the correct responses for a single proposition. Rules *g* and *h* deal with the responses to a set of propositions with the added constraint that a proposition that is accepted or challenged is a member of that set of propositions. These rules differ from rules *d* and *e* because the constraint ensures *S* is a list rather than a single proposition.

The final rules give the ability to respond to all the propositions under consideration. An agent accepting one proposition can *continue* to consider communicating acceptance of some other proposition. The *accept* could be subject to the counter rule *e* which enables the agent to consider all the supporting arguments. In the protocol this was done through recursion enabled by the use of roles. Synthesis is driven by the locutions and their relationships and as such does not have some encapsulating data structure that can force iteration over the set of supporting arguments. The result is a synthesised protocol that is more tolerant than the original and depends on the discretion of the agent doing the synthesis.

Figure 5.15 shows the resulting dialogue state for the initiating agent of the information seeking dialogue game using the protocol of figures 5.12 and 5.13. The various alternative messages (i.e. the `or` branches not taken) do not appear as the dialogue state only shows the choices made during the conversation. In this example the agent questioned the proposition ρ. The other agent replied with its assertion. The first agent challenged the assertion to which it received the reply of the set of the propositions α, β, and γ. This set being the support for the original proposition. Obligingly, the agent accepts all supporting propositions for ρ.

Figures 5.16 and 5.17 is the process and the construction of the same instance of

$a(seeker(\rho, agentB), agentA)$ **::=**

$c(question(\rho) \Rightarrow a(provider(\rho, agentA), agentB))$ **then**

$c(assert(\rho) \Leftarrow a(provider(\rho, agentA), agentB))$ **then**

$a(challenger([\rho], agentB), agentA)$ **::=**

$c(challenge(\rho) \Rightarrow a(defender([\rho], agentA), agentB))$ **then**

$c(assert([\alpha, \beta, \gamma]) \Leftarrow a(defender([\rho], agentA), agentB))$ **then**

$a(challenger([\alpha, \beta, \gamma], agentB), agentA)$ **::=**

$c(accept(\alpha) \Rightarrow a(defender([\alpha, \beta, \gamma], agentA), agentB))$ **then**

$a(challenger([\beta, \gamma], agentB), agentA)$ **::=**

$c(accept(\beta) \Rightarrow a(defender([\beta, \gamma], agentA), agentB))$ **then**

$a(challenger([\gamma], agentB), agentA)$ **::=**

$c(accept(\gamma) \Rightarrow a(defender([\gamma], agentA), agentB))$ **then**

$a(challenger([], agentB), agentA)))$ **then**

$a(challenger([], agentB), agentA).$

Figure 5.15: Resulting Dialogue State Using the Information Seeking Protocol

Rule a, b, and c applied

(1) $question(\rho) \Rightarrow a(\_, agentB)$ **then** $assert(\rho) \Leftarrow a(\_, agentB)$ **or**
$assert(not(\rho)) \Leftarrow a(\_, agentB)$ **or** $assert(unknown) \Leftarrow a(\_, agentB)$

Rule d and e applied

(2) $c(question(\rho)) \Rightarrow a(\_, agentB))$ **then** $c(assert(\rho) \Leftarrow a(\_, agentB))$ **then**
$accept(\rho) \Rightarrow a(\_, agentB))$ **or** $challenge(\rho) \Rightarrow a(\_, agentB)$

Rule f, g, and h applied

(3) $c(question(\rho) \Rightarrow a(\_, agentB))$ **then** $c(assert(\rho) \Leftarrow a(\_, agentB))$ **then**
$c(challenge(\rho) \Rightarrow a(\_, agentB))$ **then** $assert([\alpha, \beta, \gamma]) \Leftarrow a(\_, agentB)$ **then**
$accept(\alpha) \Rightarrow a(\_, agentB)$ **or** $challenge(\alpha) \Rightarrow a(\_, agentB)$

Rule i, and e applied

(4) $c(question(\rho) \Rightarrow a(\_, agentB))$ **then** $c(assert(\rho) \Leftarrow a(\_, agentB))$ **then**
$c(challenge(\rho) \Rightarrow a(\_, agentB))$ **then** $c(assert([\alpha, \beta, \gamma]) \Leftarrow a(\_, agentB))$ **then**
$accept(\alpha) \Rightarrow a(\_, agentB)$ **then**
$accept(\beta) \Rightarrow a(\_, agentB)$ **or** $challenge(\beta) \Rightarrow a(\_, agentB)$

Figure 5.16: Synthesis and Expansion of the Same Information Seeking Dialogue Game

Rule i, and e applied

(5) $c(question(\rho) \Rightarrow a(\_, agentB))$ **then** $c(assert(\rho) \Leftarrow a(\_, agentB))$ **then**
$c(challenge(\rho) \Rightarrow a(\_, agentB))$ **then** $c(assert([\alpha, \beta, \gamma]) \Leftarrow a(\_, agentB))$ **then**
$c(accept(\alpha) \Rightarrow a(\_, agentB))$ **then** $accept(\beta) \Rightarrow a(\_, agentB)$ **then**
$accept(\gamma) \Rightarrow a(\_, agentB)$ **or** $challenge(\gamma) \Rightarrow a(\_, agentB)$

Rule i, and e applied

(6) $c(question(\rho) \Rightarrow a(\_, agentB))$ **then** $c(assert(\rho) \Leftarrow a(\_, agentB))$ **then**
$c(challenge(\rho) \Rightarrow a(\_, agentB))$ **then** $c(assert([\alpha, \beta, \gamma]) \Leftarrow a(\_, agentB))$ **then**
$c(accept(\alpha) \Rightarrow a(\_, agentB))$ **then** $c(accept(\beta) \Rightarrow a(\_, agentB))$ **then**
$accept(\gamma) \Rightarrow a(\_, agentB)$

No more protocol is synthesised

(7) $c(question(\rho) \Rightarrow a(\_, agentB))$ **then** $c(assert(\rho) \Leftarrow a(\_, agentB))$ **then**
$c(challenge(\rho) \Rightarrow a(\_, agentB))$ **then** $c(assert([\alpha, \beta, \gamma]) \Leftarrow a(\_, agentB))$ **then**
$c(accept(\alpha) \Rightarrow a(\_, agentB))$ **then** $c(accept(\beta) \Rightarrow a(\_, agentB))$ **then**
$c(accept(\gamma) \Rightarrow a(\_, agentB))$

Figure 5.17: Synthesis and Expansion of the Same Information Seeking Dialogue Game[cont'd]

the information seeking dialogue game. Rather than using the prefabricated dialogue game protocol, the agent constructs the game during the interaction as defined by the synthesis rules of figure 5.14. At step one, the agent applies the synthesis rules *a, b,* and *c* stopping after the *counter* rule. Nothing has been communicated yet and the now synthesised protocol resembles the conversational choices provided by the *seeker* and *provider* roles. One agent can ask a question and the other can reply with an assertion, an assertion of the negation, or an assertion of *unknown*. In our example, two messages are then passed, question(ρ) and assert(ρ). These messages are recorded as closed in the dialogue state and the alternative locution choices are no longer shown.

For step two, the rules *d* and *e* are applied allowing the agent to either accept or challenge the other agent's assertion. This is the same behaviour allowed by the adoption of the *challenger* role.

In step three, the agent chooses to challenge but before the message is sent the rules *f, g,* and *h* are applied. This provides *agentB* with the ability to assert the supporting propositions, α, β, and γ and for *agentA* himself to respond with an acceptance or challenge upon an element of that proposition set. The current synthesis rules depend on the agent to decide which proposition to consider for acceptance or not, whereas the protocol of figures 5.12 and 5.13 gave the agent no choice and ensured that all propositions are considered.

Step four occurs after the challenge is sent and the assertion of the support is received. The rules *i*, and *e* are used to allow the agent to accept or challenge one of the other propositions of the support. Like the dialogue in figure 5.15, step five, six and seven repeatedly use the rules *i*, and *e* to enable *agentA* to accept each supporting proposition. Figure 5.17 starts with step five.

Figure 5.18 shows a graphical representation of the final dialogue state. The protocolled approach is on the right and the synthesised approach is on the left. The right-hand side has the roles, the shaded boxes, being expanded and the locutions being sent and received. The left-hand side shows the dialogue state from synthesis. There are no

role adoptions, but the locutions and their content are the same.

If necessary by the addition of one more rule we can have the ability to embed the information seeking dialogue games just as was done in the previous chapter. This allows more complex dialogue games consisting of more than one instance of an information seeking game. It simply requires an additional synthesis rule. Of course, this simple addition is relying on the assumption that our agent has the intelligence to use the rule appropriately.

k) **response(**_,question(P)**)**.

By allowing the *response* to any message to be the first message of the information seeking game (i.e. the commencement rule), an agent can initiate that type of game at any point within another. The same could be done to initiate the other atomic game types as well, but what if this is too much flexibility and a more regimented approach is needed. The use of constraints express context sensitive information, but they have a limited scope.

5.5 Chapter Summary

Synthesis using dialogue structures has provided a novel mechanism for dynamic distributed protocols. It had the advantage over the dynamic protocols described in the previous chapter of having traceability. Given a protocol and a set of synthesis rules, you could reconstruct the protocol or vise versa (i.e. determine a set of synthesis rules that created the protocol.) The cost for this was an increased reliance on an agent's understanding of the dialogue state. It was not possible to express explicit changes in role in the synthesised protocols. The introduction of synthesised role adoption would sacrifice the traceability, because the result of the expansion of the role into an agent clause cannot be guaranteed. It now depended on the agent to ensure all supporting

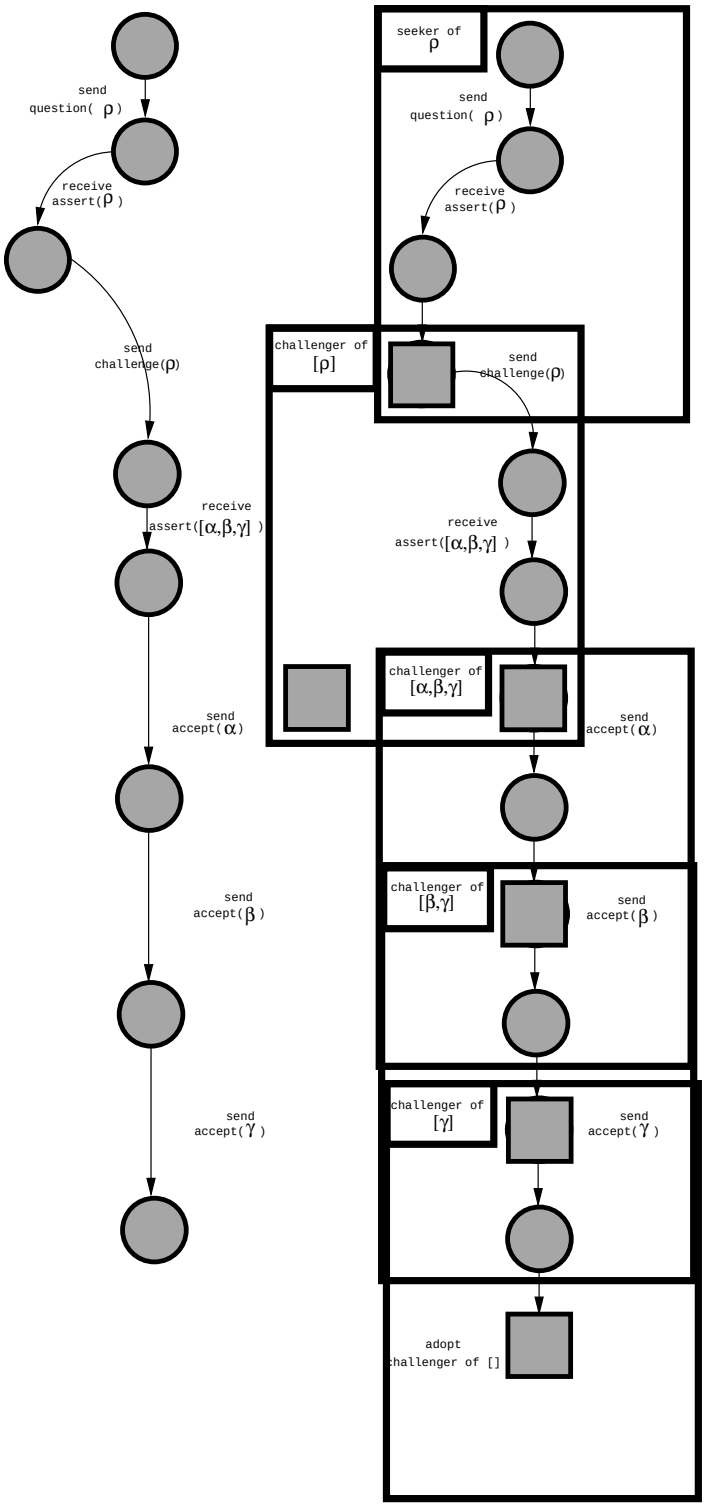

Figure 5.18: *Two Versions of the Dialogue State*

arguments were considered. The semantics of the protocols become compositional by the predicates that tie two domain specific locutions to a particular LCC syntactical construction. These predicates can can then become part of more complex predicates and therefore more complex protocol constructions. Functionally there is no difference and, as seen in the example, the same conversation spaces can be covered.

One possible way to keep track of any recursion or role adoption can be done is a manipulation of the common knowledge through the assertion and retraction of propositions. This solution though viable presents additional problems that should be considered. One attractive feature of LCC protocols is they are defined declaratively. This is lost by the assertion and retraction of facts in the knowledge base. Agent clauses become dependant on others. Since these clauses exist within the context of a multiagent system this non-monotonicity is distributed. This is a complication best avoided. For example, the situation could arise that the contributing fact for a message being sent could become unknown because of another agent's actions, but the sending agent is unaware of the retraction.

Another solution is the overloading of the messages to contain the information necessary to have recursion and role adoption. This has not been advocated as this can not be generally applicable and muddles the separate layers of communication. Despite this shortcoming, protocol synthesis has a number of potentially useful applications.

This synthesis of distributed protocols could be useful for norm convergence or evolutionary development of static protocols. An agent could reason about the synthesis rules that it applies given a certain state or conversational partner. For example, if synthesis rules x, y, and z are always triggered for a given interaction, the agent could store the produced protocol thus gaining a computational saving upon the next encounter with that agent rather than wasting computational cycles synthesising the same protocol over and over. Similarly, using synthesis agents could negotiate the protocol for communication by suggesting various sequences of rules to be applied.

Having described the contributions of this thesis, we now turn to the more prag-

matic discussion of implementation. Through out this project, the ideas developed have been reinforced by a proof-of-concept implementation built with a Prolog and Linda based program developed for the LCC framework. This has proven not only to be an excellent testing ground, but also served to show just how lightweight and modular the extensions I have described are.

Chapter 6

System Design and Implementation

This chapter describes the details of an implementation for protocol adaptations and protocol synthesis. An important contribution of this thesis is not only developing the ideas of dynamic and distributed protocols, but also providing a practical and computational solution. The explanation of this chapter is facilitated by the implementation in a high level declarative language. The protocol language, expansion engine, adaptation and synthesis engine are all written in Sicstus Prolog [the Intelligent Systems Laboratory, 1987] and the message passing system was implemented in LINDA [Carriero and Gelernter, 1989]. Firstly, the basic framework described in chapter 3 is reviewed in section 6.1. Following the progression of the thesis' chapters, section 6.2 illustrates the additions to the basic framework necessary for creating the dynamic protocols described in section 4. Through a simple addition to the interface between the agent's rationalisations and the protocol expansion engine, dynamic protocols can be achieved. Section 6.3 relates the similar functionality used for the protocol synthesis from chapter 5. Though the modifications to the interface are similar, the consequences of using synthesis has a fundamental effect upon the LCC approach.

6.1 Basic Framework

An important engineering question at the start of this work was whether the existing language and mechanism, with its virtue of simplicity and light-weight requirements for use, could accommodate the idea of adaptable and executable interaction protocols. It was necessary to answer whether this new mechanism of communication could be achieved without significant loss of expressivity and without requiring a drastic change to the underlying mechanism.

One of the significant results of this work was to confirm the suitability of LCC as a language and framework for implementing dynamic and distributed protocols. Figure 6.1 provides a simple diagram to illustrate the process. Firstly, the agents are loaded with their knowledge bases. For this implementation, agents are defined by a set of Prolog predicates and rules to govern the decision-making processes and satisfaction of any constraints that may occur in the agent clauses. Dialogues are initiated by an agent with a bootstrapping mechanism that requires a unique agent identifier, a role, and the name of a file which contains the protocol that it wants to use. The file defining the protocol in terms of agents' clauses and the common knowledge is read and loaded into memory for the agent to use. The next step is for the agent to identify the appropriate clause for its role and attempt to perform the actions defined for that role.

The expansion engine does this by trying to satisfy one of the rules in figure 3.2 described in chapter 3. The expansion engine either halts, reaches the end of the protocol and terminates, or returns a set of locutions to be sent. The agent then decides whether to send a message or not. If the agent decides not to send the locution or locutions, the expansion engine attempts to explore another path in the protocol and find another locution that can be sent. The agent once again gets to decide upon the appropriateness of the message. This continues until either the agent finds an acceptable locution to be sent or the expansion engine exhausts the possibilities and the protocol fails. Each time the expansion engine finds a locution to be sent or encounters an incoming locution,

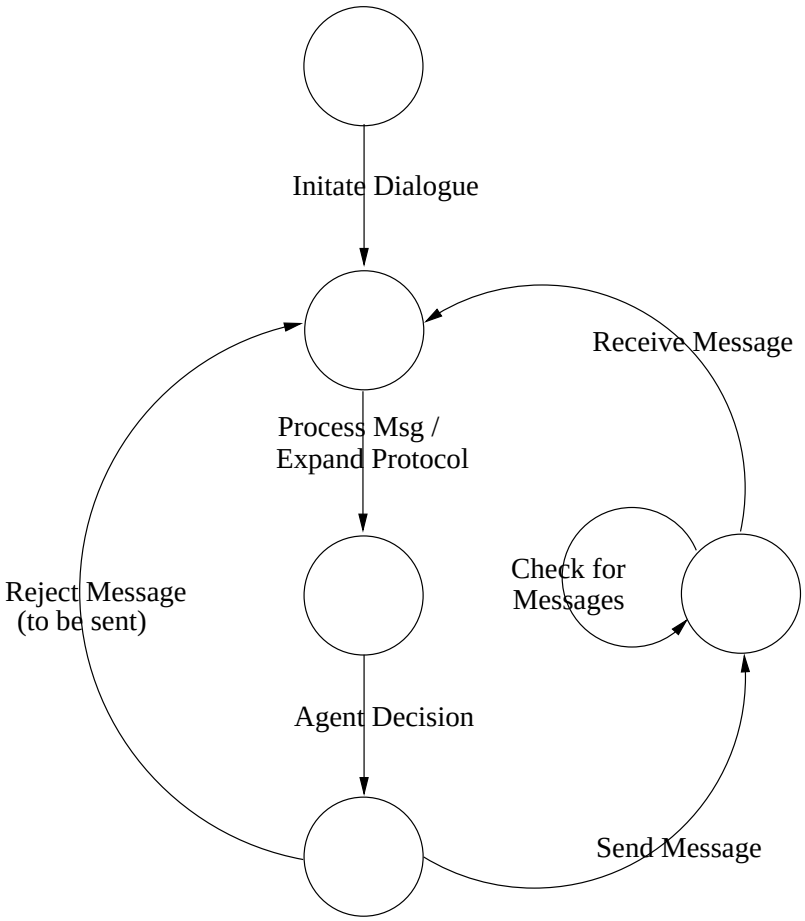

Figure 6.1: Basic LCC Framework

the corresponding portions of the dialogue state are marked to reflect those activities occurring.

The message containing the agent clauses, the common knowledge base, the marked agent clauses of the dialogue state that the expansion engine created, and the locution is sent to the LINDA server. The LINDA server uses a blackboard approach to communication and the message is left on the server addressed to the recipient as specified by the protocol. LINDA was chosen because of its simplicity and ease of use. The low level mechanics of message passing has no effect on the issues addressed by this thesis. If the system were to be developed further, it would be important to use a system in which the agents do not rely on a third party in order to gain the most advantage from the peer to peer nature of LCC and its framework.

The dialogue continues with the agents checking the LINDA server for messages addressed to its identifier. Taking the received protocol with all its constituents, the agent applies the expansion process once again. Firstly, it processes the received locutions and marks its dialogue state. Next the expansion proceeds as described before in order to find a reply that both satisfies the agent definition including any constraints and the agent's own discretion. The protocol bounces between the agents (via the blackboard mechanism of LINDA) until no part of the protocol remains open and the dialogue ends successfully. In order to have dynamic protocols or protocol synthesis this process must include a way to amend existing protocols. The surprising result is that this is the only extension required. Adaptations can be achieved without requiring any extra machinery to the framework or language save for an addition to the interface which allows the agent to adapt the protocol and return it to the expansion engine.

6.2 Creating Dynamic Protocols

Figure 6.2 shows the simple addition to the basic framework as well as illustrates its modularity. The algorithm for insertions of protocol code into an existing dialogue

state starts by dissecting the initial protocol. It identifies the agents' clauses and the corresponding clauses in the dialogue state. The head of the list in the dialogue state will be the clause for the agent who last performed an action and updated the dialogue state. Insertions are done to the dialogue state rather than the protocol itself as adaptations are meant to modify the current instance of protocol rather than the template for the current dialogue. The current position of the dialogue state is identified (i.e. after the closed portions of the agent clause).

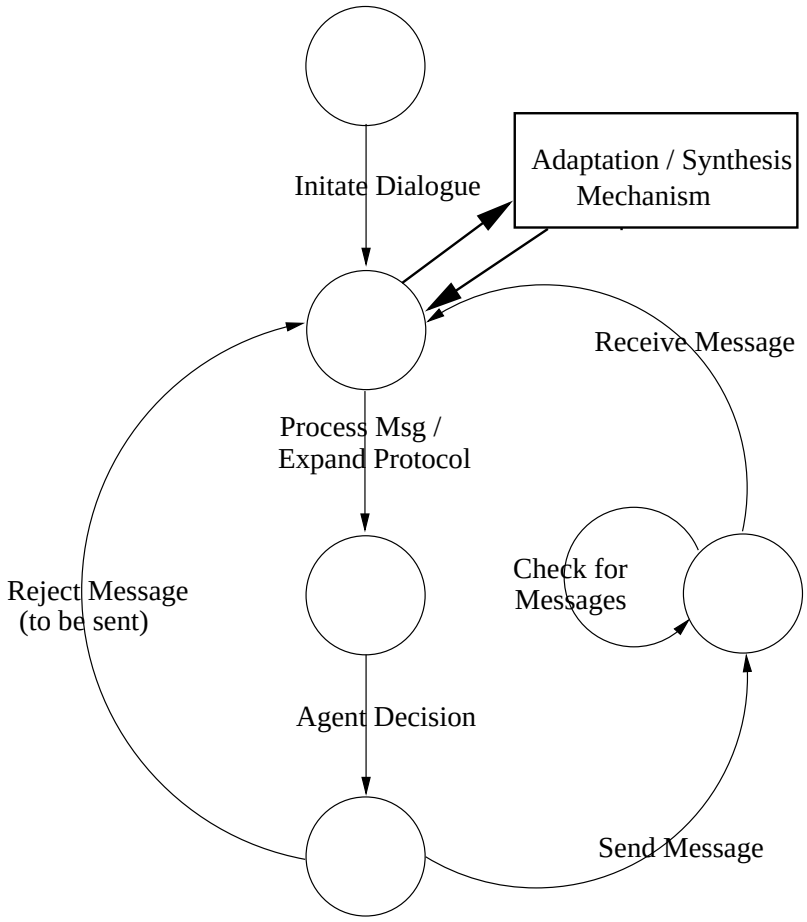

Figure 6.2: Dynamic LCC Framework

The new code is spliced in with an **then** operator. Any remaining protocol operations are then added after the newly introduced operation with is also separated by the **then** operator. This is done for both the agent performing the insertion and its conversational partner. The newly adapted protocol is then returned to the

calling predicate which can be utilised by the expansion engine and the agent can take advantage of the newly introduced protocol steps.

The only addition to the framework necessary is to introduce the ability for the agent to not only decide whether to send a message but also to evaluate whether to make an adaptation to the protocol. To achieve this, a step is introduced into the framework that provides the agent with the current dialogue state and protocol and expects a protocol to be returned. This is done immediately after the agent receives the protocol from the LINDA server. Whether the agent decides to make a transformation or not, the expansion engine proceeds regardless. All the code for this is given in sections B.1 and B.3 from appendix B.

$$
\begin{array}{ll}
1) & a(control\_layer\_a, A) \texttt{::=} \\
2) & begin(Game, P) \Rightarrow a(control\_layer\_b, B) \leftarrow choosegame(Game, P, B) \texttt{ then} \\
3) & \left(\begin{array}{l} reject(Game, P) \Leftarrow a(control\_layer\_b, B) \texttt{ or} \\ startgame(Game, P) \leftarrow accept(Game, P) \Leftarrow a(control\_layer\_b, B) \end{array} \right).
\end{array}
$$

$$
\begin{array}{ll}
4) & a(control\_layer\_b, B) \texttt{::=} \\
5) & begin(Game, P) \Leftarrow a(control\_layer\_a, A) \texttt{ then} \\
6) & \left(\begin{array}{l} reject(Game, P) \Rightarrow a(control\_layer\_a, A) \texttt{ or} \\ accept(Game, P) \Rightarrow a(control\_layer\_a, A) \end{array} \right).
\end{array}
$$

Figure 6.3: A Protocol to Define the Control Layer

For example, in chapter 4, we illustrated a complex dialogue game framework using dynamic LCC. The game started at a control layer where the type of individual game is agreed between the two agents. Figure 6.3 is the agent clauses for this control layer. In the example from chapter 4, the agent playing the role of *control_layer_a* chooses to play a game of persuasion after the agent playing *control_layer_b* agrees to

join it. The protocol for this dialogue game can be found in chapter 4, figures 4.12 and 4.13. Initially the example runs without use of the adaptation engine. The agents are queried as to whether they want to make any changes to the protocol at each step. In the implementation, the agent will make a transformation to the protocol when it matches the current dialogue state to a internal knowledge base of dialogue states which it considers ideal to make an adaptation to the protocol. If no match occurs, it returns the protocol back to the expansion engine unadulterated. The expansion engine then begins its work. In chapter 3 we described the basics of the expansion engine. Figure 6.4 reiterates the expansion rules used by the engine.

For our example, the agent who initiates the dialogue is named *agentA* and its internal rationalisation decides to input to the expansion engine the protocol to be used[1], its identity, and the role it will be playing within the protocol (i.e. control_layer_a). The expansion engine will the succeed to satisfy this rule from figure 6.4:

$$1) \quad A\texttt{::=}B \xrightarrow{M_i,M_o,\mathcal{P},O} A\texttt{::=}E$$
$$if \quad B \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

on the agent clause below which was taken from figure 6.3 (A maps to line 1 and B maps to the remainder):

$$\begin{array}{l} 1) a(control\_layer\_a, A)\texttt{::=} \\ 2) begin(Game, P) \Rightarrow a(control\_layer\_b, B) \leftarrow choosegame(Game, P, B)\ \texttt{then} \\ 3) \left(\begin{array}{l} reject(Game, P) \Leftarrow a(control\_layer\_b, B)\ \texttt{or} \\ startgame(Game, P) \leftarrow accept(Game, P) \Leftarrow a(control\_layer\_b, B) \end{array} \right). \end{array}$$

This is true because the expansion engine can satisfy this rule (A_1 maps to lines 1 and 2 and A_2 maps to line 3 of the control_layer_a agent clause):

$$4) \quad A_1\ \texttt{then}\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E\ \texttt{then}\ A_2$$
$$if \quad A_1 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

[1]This includes all the agent clauses that are part of the framework for complex dialogue games which includes the information seeking game, persuasion game, inquiry game, control layer, interrupt, etc.

$$1)\quad A\texttt{::=}B \xrightarrow{M_i,M_o,\mathcal{P},O} A\texttt{::=}E$$
$$if\quad B \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

$$2)\quad A_1\ \texttt{or}\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$
$$if\quad \neg closed(A_2) \wedge A_1 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

$$3)\quad A_1\ \texttt{or}\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$
$$if\quad \neg closed(A_1) \wedge A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

$$4)\quad A_1\ \texttt{then}\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E\ \texttt{then}\ A_2$$
$$if\quad A_1 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

$$5)\quad A_1\ \texttt{then}\ A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} A_1\ \texttt{then}\ E$$
$$if\quad closed(A_1) \wedge A_2 \xrightarrow{M_i,M_o,\mathcal{P},O} E$$

$$6)\quad C \leftarrow M \Leftarrow A \xrightarrow{M_i,M_i-\{M \Leftarrow A\},\mathcal{P},\emptyset} c(M \Leftarrow A)$$
$$if\quad (M \Leftarrow A) \in M_i \wedge satisfy(C)$$

$$7)\quad M \Rightarrow A \leftarrow C \xrightarrow{M_i,M_o,\mathcal{P},\{M \Rightarrow A\}} c(M \Rightarrow A)$$
$$if\quad satisfied(C)$$

$$8)\quad null \leftarrow C \xrightarrow{M_i,M_o,\mathcal{P},\emptyset} c(null)$$
$$if\quad satisfied(C)$$

$$9)\quad agent(r,id) \leftarrow C \xrightarrow{M_i,M_o,\mathcal{P},\emptyset} a(R,I)\texttt{::=}B$$
$$if\quad clause(\mathcal{P}, a(R,I)\texttt{::=}B) \wedge satisfied(C)$$

Figure 6.4: Rules for Expanding an Agent Clause

This in turn is true because this rule succeeds:

$$7) \quad M \Rightarrow A \leftarrow C \xrightarrow{M_i, M_o, \mathcal{P}, \{M \Rightarrow A\}} c(M \Rightarrow A)$$
if satisfied(C)

This is because the *agentA* can satisfy the constraint *choosegame(Game,P,B)* and put the *begin* locution in the set of outgoing messages which will be sent along with the protocol to *agentB* whose identity was returned as part of satisfying the *choosegame(persuasion,thursday_best_meeting_day,agentB)* constraint. The *agentB* will the receive the dialogue state shown in figure 6.5.

$$a(control\_layer\_a, agentA) \texttt{::=}$$
$$c(begin(persuasion, thursday\_best\_meeting\_day) \Rightarrow$$
$$a(control\_layer\_b, agentB)) \texttt{ then}$$
$$\left(\begin{array}{l} reject(Game, P) \Leftarrow a(control\_layer\_b, B) \texttt{ or} \\ startgame(Game, P) \leftarrow accept(Game, P) \Leftarrow a(control\_layer\_b, B) \end{array} \right).$$

Figure 6.5: Dialogue State after agentA's expansion

An LCC message consists of several parts including the locutions being sent as well as all the agent clauses being used and the current dialogue state (shown in figure 6.5. There is no dialogue state clause for *agentB* because it has not participated in the conversation yet. Once *agentB* does receive the message, it will first look at the dialogue state to find a clause for which it can satisfy one of the rules shown in figure 6.4. At this point in the dialogue, it will fail to do that. It will then proceed to examine the agent clauses sent with the protocol for a clause which is satisfiable. It will succeed when it finds:

$$4) a(control\_layer\_b, B) \texttt{::=}$$
$$5) begin(Game, P) \Leftarrow a(control\_layer\_a, A) \texttt{ then}$$
$$6) \left(\begin{array}{l} reject(Game, P) \Rightarrow a(control\_layer\_a, A) \texttt{ or} \\ accept(Game, P) \Rightarrow a(control\_layer\_a, A) \end{array} \right).$$

A similar set of steps occurs as with *agentA*. No synthesis occurs. Rule 1 from figure 6.4 is satisfied by A being line 4 and B being lines 5 and 6. Rule 4 is satisfied

with A_1 being mapped to lines 4 and 5 and A_2 being mapped to 6. Next rule 5 is also satisfied because *agentA* sent the locution expected and that part of the dialogue state was marked as closed. The A_1 for this rule is line 5 and A_2 is line 6. The reason rule 5 succeeds is because with no explicit constraints the agent is free to choose one branch of the **or** operator or the other. In this example, *agentB* chooses the second half and therefore can satisfy this rule:

$$3) \quad A_1 \ \mathbf{or} \ A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E$$
$$if \quad \neg closed(A_1) \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E$$

The message is put in the set of outgoing messages and it is returned to *agentA*, the addressee of the locution. The dialogue state now is shown in figure 6.6.

$$1) a(control\_layer\_a, agentA) \mathbf{::=}$$
$$2) c(begin(persuasion, thursday\_best\_meeting\_day) \Rightarrow a(control\_layer\_b, agentB)) \ \mathbf{then}$$
$$3) \left(\begin{array}{l} reject(persuasion, thursday\_best\_meeting\_day) \Leftarrow a(control\_layer\_b, agentB) \ \mathbf{or} \\ startgame(persuasion, thursday\_best\_meeting\_day) \leftarrow accept(persuasion, thursday\_best\_meeting\_day) \\ \qquad \Leftarrow a(control\_layer\_b, agentB) \end{array} \right).$$
$$4) a(control\_layer\_b, agentB) \mathbf{::=}$$
$$5) c(begin(persuasion, thursday\_best\_meeting\_day) \Leftarrow a(control\_layer\_a, agentA)) \ \mathbf{then}$$
$$6) c(accept(persuasion, thursday\_best\_meeting\_day) \Rightarrow a(control\_layer\_a, agentA)).$$

Figure 6.6: Dialogue State after *agentB*'s response

The *agentA* receives this response with the locution and updated dialogue state. Initially the agent makes no adaptations. Once the expansion engine begins its work satisfying the expansion rule:

$$6) \quad C \leftarrow M \Leftarrow A \xrightarrow{M_i, M_i - \{M \Leftarrow A\}, \mathcal{P}, \emptyset} c(M \Leftarrow A)$$
$$if \quad (M \Leftarrow A) \in M_i \wedge satisfy(C)$$

The agent has been engineered to be able to satisfy the constraint. The *agentA* has been encoded with a predicate where upon the satisfaction of a *startgame* constraint on in its protocol, it will adopt the intention to introduce a protocol adaptation with the needed roles defined in *startgame* (i.e. for our example this is a persuasion game).

The adaptation engine must maintain the protocol symmetry. Internally, the agent has been defined with the roles that constitute the starting roles for the atomic dialogue games. Its initial role will that of the *persuader* and its partner will be playing the role of *listener*. It is assumed that any protocol being used is symmetric and the dialogue partners' agents clauses compliment each other. If this assumption holds, it is guaranteed to hold after any protocol adaptation. The adaptation engine will place the new message at the end the current point of the dialogue.

$$c(A) \textbf{ then } Na \wedge c(B) \textbf{ then } Nb \leftarrow c(A) \wedge c(B)$$

Here A represents the protocol operations for the agent whose is adapting the protocol and B represents those for its dialogical partner. Na and Nb are the newly introduced operations. The result for our example would look like figure 6.7. Afterwards, the adaptation engine would return the modified dialogue state to the expansion engine which in turn would expand the newly introduced operations (i.e. unfold the persuader role) and the dialogue would continue like it was described in chapter 4.

Conceptually the adaptation process is not much different from the basic framework. As it still relies on a library of predefined protocols that can be spliced in by the adaptation engine. The protocols do not come *ex nihilo*. This is not the case when the agent uses synthesis.

1) $a(control\_layer\_a, agentA)$ **::=**

2) $c(begin(persuasion, thursday\_best\_meeting\_day) \Rightarrow$
$a(control\_layer\_b, agentB))$ **then**

3) $c(accept(persuasion, thursday\_best\_meeting\_day)$
$\Leftarrow a(control\_layer\_b, agentB))$ **then**

3.5) **a(persuader(thursday_best_meeting_day,agentB),agentA)**.

4) $a(control\_layer\_b, agentB)$ **::=**

5) $c(begin(persuasion, thursday\_best\_meeting\_day) \Leftarrow$
$a(control\_layer\_a, agentA))$ **then**

6) $c(accept(persuasion, thursday\_best\_meeting\_day)$
$\Rightarrow a(control\_layer\_a, agentA))$ **then**

7) **a(listener(thursday_best_meeting_day,agentA),agentB)**.

Figure 6.7: Dialogue State after *agentB*'s response

6.3 Synthesising Protocols

Synthesising protocols reuses the mechanism that enabled adaptations to be performed to existing protocols. The predicates from section B.1 can be used unchanged. However the nature of the adaptation has changed completely. We are no longer adapting existing protocols. Instead we must let the agent's synthesis rules dictate the incremental adaptations. This requires a modification to the interface.

Several iterations of adaptations will be made as the synthesising agent applies its domain rules. Previously, the agent could make sure its adaptations include the additional steps required for the agent to make the appropriate reply. Due to the incremental way the protocols are created and the importance of building upon each previous action taken in the dialogue after each choice the agent makes, the loop provides the agent with an opportunity to expand and synthesise the protocol until neither is possible and the agent sends its messages or the conversation concludes.

This requires additional code as shown in section B.2 as well as a modified interface, section B.4. The dialogue starts by an agent choosing a locution with which to begin the conversation. The synthesis engine takes this locution and attempts to apply any matching synthesis rules that have been defined for the domain. The synthesis rules are applied, and the output is the newly synthesised steps which are amended to the protocol. When the agent applies a domain rule the synthesis engine first translates the rule to the appropriate LCC syntax as shown in figure 5.6 from chapter 5. This new protocol is expanded by the synthesis engine as well as allowing the agent to make any decision about which message to send. Unlike the previous approach, it is necessary to return the protocol to the synthesis engine as it may need to be amended given the dialogical choice the agent just made. In particular, it may be required to synthesise the response steps for its dialogical partner. Once the cycle of expansion and synthesis is exhausted and no more protocol can be amended or traversed, the message is sent as normal.

By this simple addition to the basic framework, the functionality of distributed

protocol communication is changed. Synthesis entirely replaces the need for protocol libraries and becomes the driving force of the interaction. The agents involved no longer have any of the agent clauses that were one of the main components of the LCC message. Instead, there is only the dialogue state onto which additional protocol steps are synthesised. We achieve "protocol-esque" communication in a way that the synthesising agent itself can autonomously and automatically create. This increased functionality comes at a minimal additional cost in code and modification to the existing framework.

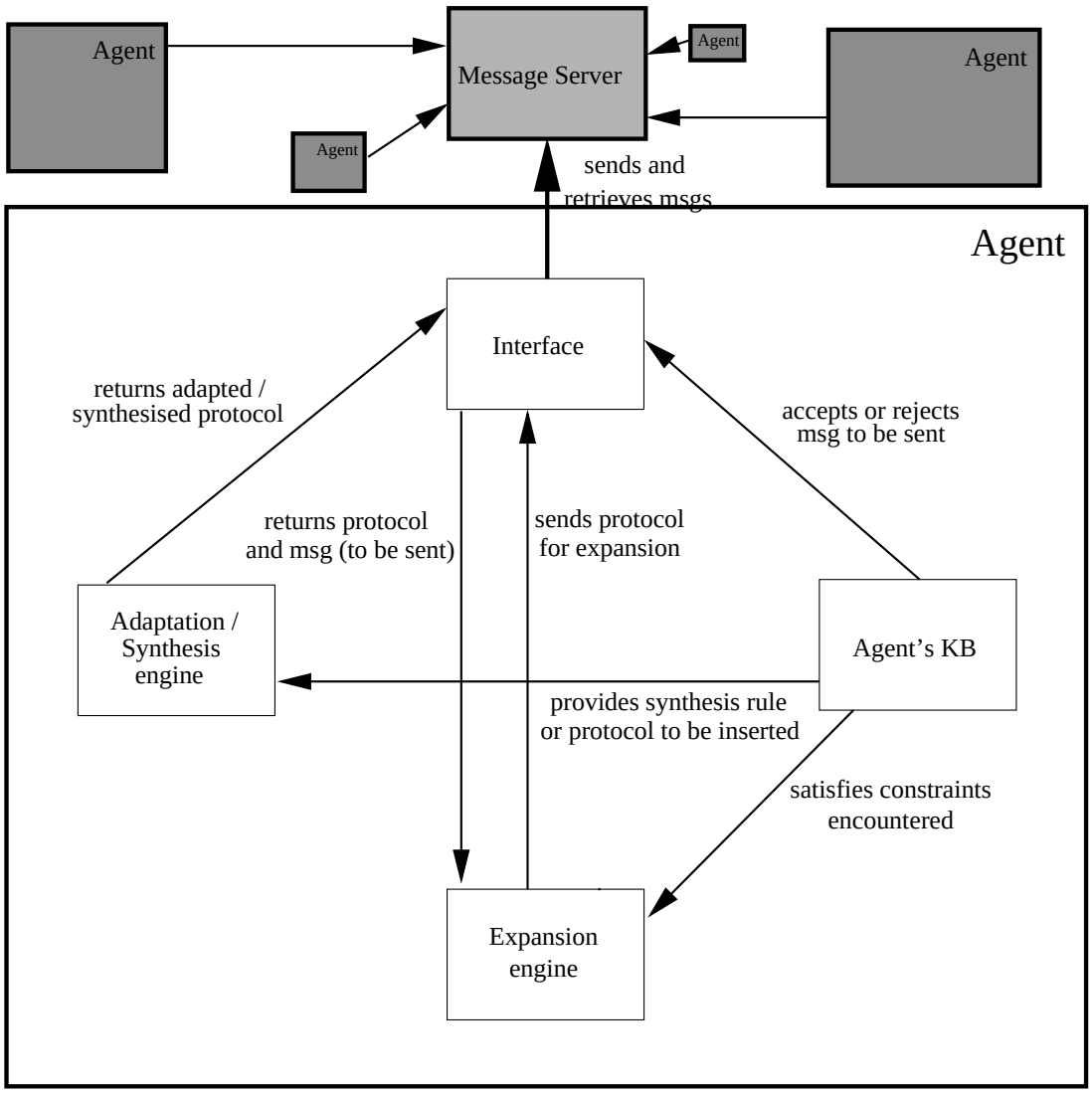

Figure 6.8: Relationship Diagram Between Components

Figure 6.8 shows a diagram of the relationships between the components of our communicating agent. The diagram focuses on one agent from the system. It has several key parts; the expansion engine, the agent's knowledge base, the synthesis engine, the adaptation engine and the interface. The interface connects all the components together and is equivalent to the main class from object oriented programming. The knowledge base contains all the agent's preferences and decision-making machinery. The expansion engine refers to the knowledge base to satisfy any constraints that it encounters in the expansion of protocols. If the constraint cannot be satisfied by this knowledge base that operation dependant on that constraint cannot succeed, as defined by the expansion rules from figure 3.2. The interface calls upon the expansion engine to expand a protocol and a dialogue state that is amiable to the agent's rationalisations. The agent's rationalisation might also choose to adapt / synthesise the existing protocol and the aptly named adaptation / synthesis engine provides that functionality.

6.4 Chapter Summary

The modifications necessary to satisfy the ultimate goal of dynamic protocols were surprisingly unobtrusive upon the language of LCC and its associated expansion engine. The ability to modify protocols during the execution of an agent conversation was achieved in a wholly modular way allowing the functionality to be used or not without consequence to the protocols or the interaction. Certain engineering decisions were made concerning this implementation such as the use of Prolog and Linda.

The ideas explained in this thesis are not contingent on any particular language or technology. Though I have more experience in other languages, in particular JAVA, the choice to use Prolog was in order to take advantage of existing code for the basic framework and LCC expansion engine. The high level nature of the language surely facilitated the translation, implementation, and testing of the concepts developed. It was important to show that a practical implementation was possible without the ap-

proach being dependant on it. However, I am not claiming that the implementation is of industrial strength in that it has been through a full cycle of quality assurance testing. It is not a very user oriented or aesthetically pleasing implementation either. The inherited code for the basic LCC framework did have a graphical front end, but in order to truly identify and remedy bugs, it was simpler for myself to interact with the lowest level of the system without the GUI sugar coating.

Despite these issues, the implementation is useful as a proving grounds for the work developed in the thesis as well as providing an example for other implementations and agent system frameworks. Indeed, the work developed is not restricted solely to that of agents. The concepts developed here also have purchase in the related field of the semantic web and web services [Walton and Barker, 2004].

Chapter 7

Conclusions

There are a great number of issues to address before computers can communicate in an equally robust and complex manner as their human creators. Computers already have a brute force advantage over humans when it comes to computation or multi-tasking. If computational power can indeed emulate our innate ability to infer meaning and context from the scantest of information, follow the capricious flow of human conversations or even deceive, the consequences would be extremely profound. One could imagine not only complex negotiations being handled in seconds, but thousands of such negotiations in parallel. For example, the negotiation of international mobile phone call charges could be completed between your telecom and the host country's before the person you are calling answers. Fanciful speculation aside, the contributions of this thesis make a small measure of progress toward that grand challenge of more human than human communication.

7.1 Summary

The agent paradigm is clearly well suited for the massively distributed peer-to-peer environment that the ubiquity of the internet has given us. Those day dreams of thousands of simultaneous complex negotiations will undoubtedly take place upon the web with source computers spread across the globe. Agent communication will be an es-

sential component. The purpose of this thesis was to explore and hopefully expand the understanding of agent communication. At the start of this investigation, a gap was perceived. There was the strict and static protocols of the 'top-down' institutional approaches to multiagent interactions. The approach sacrificed agent autonomy for the reliable and predictable social system into which they would participate.

On the other side were the 'bottom-up' approaches. They were models of communication for individual agents that would hopefully arm them sufficiently with the ability to reason about the communicative actions of other agents and allow them to reply appropriately, or understand their own desires and the correct messages to communicate to convince others to help satisfy those desires. From this, it is hoped that the community would emerge from these individuals. Reliability was sacrificed to autonomy and flexibility.

The goal of the project became to address this gap, to find a way to utilise advantages from both approaches. The use of the LCC language and frameworks developed in [Robertson, 2004b, Robertson, 2004a, Robertson, 2004c, Walton and Robertson, 2002] for distributed protocols facilitated this. Agents were now able to adapt their interaction protocol and communicate those changes to their dialogical partner. The following sections present an evaluation of this project's hypotheses and goals, as well as speculation on how to address some of the open issues that remain given this research domain.

7.2 Evaluation of Contribution

This thesis has shown that dynamic protocols are indeed possible. Using a distributed language and an abstract model of communication, it is possible to generate dynamic protocols without compromising either LCC language or the abstract communicative model being implemented. This additional functionality has been done in a modular way that does not impede on the functionality of the expansion engine used. The thesis'

main contribution is the movement of locus and timing of protocol agreement from the agent engineer and specification phase to the agents themselves at execution time.

One of the initial questions raised by this thesis is whether the representation of agent-centric approaches of communication could be expressed as LCC protocols. We showed that is was possible, but we also showed LCC to be inappropriate for the communicative acts of FIPA's BDI-based model. This is because of the conflict between the explicit and public representation and communication of the protocols and dialogue states inherit in the model of LCC versus FIPA's reliance on mentalistic and private grounding for their locutions.

Given a model of communication represented as LCC, was it possible to alleviate some of the burden for communicative partners? In other words, could an agent receive a protocol developed in accordance with some communicative model such as dialogue games and participate in the interaction without the need to understand the model represented? This approach certainly alleviates this dialects problem. The problem when two agent share a common model of communication (e.g. dialogue games), but differ on a specific implementation. It is unavoidable that communicating agents will be required to share some common knowledge and representation of the topic of conversation. Our approach helps to further elucidate the particular semantics of the locutions and social norms for conversing. Dynamic distributed protocols can also do this in response to run time requirements of the conversation itself rather than rely on an *a priori* understanding of all the possible paths of discourse.

The core contributions of this thesis come from chapters 4 and 5. In chapter 4, we first explored the use of BDI based protocols by using FIPA specifications. This approach addressed some of the issues associated with using those specifications as a basis for communication. Firstly, this project achieved a means of representing FIPA's norms for the use of communicative acts as a protocol, but in a dynamic way that would allow an agent using the communicative acts to still decide when and where to use those acts without being confined to a predefined and static protocol. The use of

dynamic LCC protocols allow agents to communicate the dialogical context as well as the expectation for other's communicative actions.

The representation of FIPA's communicative act library and their AUML interaction protocol specifications as LCC clarifies any ambiguities in those specifications as well as provides a format that is much more easily evaluated, verified and, if needed, modified or corrected. Although these advantages can be gained by the use of our dynamic additions to LCC and FIPA's ACL, it is FIPA's basing communication on the occurrence of internal and private mental states that makes this approach unsuitable. The representation as LCC constraints of these mental states in the protocols have the benefit of explicitly making those states accountable, it does not address the fundamental issue of sincerity and verifiability. However this is not a criticism of the FIPA approach but instead shows that not all models of communication are convivial to the distributed protocol approach even though a representation with dynamic and distributed protocols is possible.

Section 4.2 proposes using dialogue games as a suitable model of communication to be implemented as dynamic LCC. This provided the benefit of a practical implementation platform for which dialogue games could not only be developed but executed. Similar to section 4.1 the model of dialogue games for communication benefits from being represented as LCC by providing a mechanism to communicate its social norms to other agents during their interactions as well as providing a format that can be evaluated, verified, and modified automatically.

Chapter 5 extends this work to an even more novel approach. Rather than modifying existing protocols, we attempted to synthesise protocols by finding a small but exhaustive set of transformations that could then be applied in reaction the current dialogue state. We evaluated all the permutations of transformations, identifying and discarding the objectionable ones, and finding matches to phenomena identified for human communication for the remainder. This resulted in a small minimal set of transformations that still provided the same expressiveness of normal LCC protocols. This

minimal set of synthesis rules was shown to be complete (i.e. it could create all possible protocols given the restrictions imposed) and sound in that it was not possible to introduce syntactically erroneous protocols using synthesis.

Through a relatively small modification of code, we we able to achieve a fundamental change in the functionality of dynamic and distributed protocols. No longer do agents require libraries of protocols to address all the possible interactions they are capable of. Instead, the agents synthesise the protocols by the application of a set of rules that translate to syntactically correct protocols.

One unexpected result was the addition of the dynamic functionality of protocols was all achieved without needing to modify the LCC language or expansion engine. This increased amount of flexibility and dynamism was achieved in a completely modular way and only required the framework to provide an opportunity to the agent to make any needed transformations from which the resulting protocols are identical to those defined in the more traditional way (i.e. a static definition defined by an engineer prior to the interaction).

Overall, the line that separates the "agent-centric" and protocol led models of communication has been blurred. It is now possible to represent the capricious nature of the "agent-centric" communication as the more reliable and accountable protocols without requiring the agent to be dictated to by the protocol. Instead, the agent can still use its reasoning capabilities and communicative model to modify the LCC to reflect avenues of dialogue it wants to explore. To summarize, below is a bullet pointed list of the thesis' contributions (in no particular order):

- For an interesting set of agent conversations, we can produce dynamic and distributed protocols with out compromising the communicative models they represent

- A shift in the locus and timing of agreement toward agents and execution rather than designer and specification time.

- A movement from specifications from natural language, semi-formal specifications to specifications that are computable, formal, and executable.
- A shift for dialogue games from the theoretic to the practical
- Introduction of a “third-way” (i.e. mutable protocols)
- Provided one possible implementation approach for the ideas described in the thesis
- Provide a means and a method for translating agent centric approaches into dynamic protocols
- Show the advantages for using an explicit computational social norm which can be communicated by the agents using them.
- Show dynamic distributed protocols can relax communication requirements on agents

7.3 Future Work

Although this research has achieved much of its expected contribution, there are some remaining questions that could extend and enhance the results of this thesis. It would be good to further mitigate the restrictions on the class of agent conversations. It was necessary to restrict ourselves for the purposes of protocol synthesis to dialogues or serialisable multi-party conversations. An immediate improvement upon this work would be to open that Pandora’s box of agent communication and look into protocol synthesis for multi-party dialogues.

Our results make the possibility of ”protocols about protocols” a very real possibility. Agents could negotiate the protocols as they use them. This capability was explored in chapter 4 using the McBurney/Parsons frame-

work [McBurney and Parsons, 2002] but with agents only being able to introduce predefined (and now executable) dialogue game protocols.

For synthesis we have moved much more responsibility to the agent for the successful execution of the protocol. What needs to be explored further is the computational consequences of this move and can there be some recovery of this responsibility back to the distributed protocol that synthesis produces. This should also include a more in-depth investigation into the computational savings of using distributed and dynamic protocols.

The future work described in this section directly result from the findings of this thesis. They are the questions that were raised by this thesis' objectives. The next section mentions future work of a more tangential and further afield nature.

7.4 Possible Extensions

Distributed interaction protocols is a relatively unplumbed field of research within the agent communication community. Making these protocols dynamic has been usually left as speculation in the final sections of existing literature [de Silva, 2002, Nodine and Unruh, 1997, Martin et al., 1996, Freire and Botelho, 2002]. Due to this domain's incipient nature, there are more questions that remain than can possibly fit into the short span of research time allotted to this thesis. We have focused on addressing the fundamental issues for implementing mutability in protocol led agent communication, but there remains a number of channels for continued exploration.

7.4.1 Trust

When is it okay to trust agents to modify protocols? Giving agents providence over the social norms as they are using those norms is to provide the ability to subvert them too. Previous research into static distributed protocols focused solely on the topic of trust [de Silva, 2002] and how an agent might evaluate a received protocol. Trust is

an open issue for agency in general [Ramchurn et al., 2004] and research literature is applicable for use with the distributed dynamic protocols as well. The unique difficulty is the need to ensure desirable properties are some how guarded from exclusion by any adaptations performed. This includes the more mundane properties such as deadlock or termination as well as more abstract and domain specific properties (e.g. highest bidder always wins the auction). Model checking techniques have been proposed to address these issues and their findings are appropriate for static as well as dynamic protocols [Osman et al., 2006].

7.4.2 Permission

The literature has already described a number of permissive strategies concerning the utterances of individual locutions, and participation in a multiagent system [Noriega, 1997, Vázquez-Salceda, 2003, Esteva et al., 2002, Esteva et al., 2000, Estava et al., 2001, Dignum, 2003, Cortés, 2004, Esteva et al., 2004]. It is likely that there is the need for mechanisms to control who has the right to modify the protocols as well as when and how much of the protocol should be adapted.

7.4.3 Possible Collaborative Technologies

This thesis has focused on the use of dialogue games to illustrate its approach. Their use is not necessary to realising dynamic protocols, but are due to their similarity in view of agent communication. They were an excellent complement. Another model that to complement protocol synthesis in particular is the work being done on eco-grammars adapted to the agency paradigm [Paun and Salomaa, 1999].

Researchers have begun to investigate the use of eco-grammars for application in agent communication [Bel-Enguix and Jiménez-López, 2005]. Using theories of conversational grammar systems (CGS), the researchers attempt to develop a formal and computational theory of dialogue by an extension of eco-grammars. The classical ap-

proach to formal language theory uses a *single* grammar to generate a language. In an eco-grammar system there are *several* grammars used to generate the one language. Using this framework the authors have attempted to make correlations to a multiagent model of communication.

Though the authors have an eye toward implementation in multiagent systems, the research is theoretical in nature and has not fully converted its concepts to the difficulties inherit in multiagent systems and communication. It currently lacks a translation mechanism to convert the results of the grammar rules to a format that an agent can utilize for communication in a multiagent system. The research of this thesis can nicely fill this developmental gap. It is simple to imagine connecting the grammar rules to adjacency pairs and the LCC synthesis rules that have just been described. One could envision run time synthesis and execution of the interaction protocol using the communicating agents' grammar rules expressed as LCC protocols.

Appendix A

Complete Trace of the Dialogue State from 4.2.2.2

$a(control\_layer\_a, agentA)$ **::=**

$c(begin(persuasion, thursday\_best\_meeting\_day) \Rightarrow$

$a(control\_layer\_b, agentB))$ **then**

$c(accept(persuasion, thursday\_best\_meeting\_day) \Leftarrow$

$a(control\_layer\_b, agentB))$.

a(persuader(thursday_best_meeting_day,agentB),agentA)::=

$c(know(thursday\_best\_meeting\_day) \Rightarrow$

$a(listener(thursday\_best\_meeting\_day, agentA), agentB))$ **then**

$c(assert(thursday\_best\_meeting\_day) \Rightarrow$

$a(listener(thursday\_best\_meeting\_day, agentA), agentB))$ **then**

$a(defender([thursday\_best\_meeting\_day], agentB), agentA)$ **::=**

$c(assert(not(thursday\_best\_meeting\_day)) \Leftarrow$

$a(challenger([thursday\_best\_meeting\_day], agentB), agentA))$ **then**

$a(challenger([not(thursday\_best\_meeting\_day)], agentA), agentB)$ **::=**

$*\quad c(challenge(thursday\_best\_meeting\_day) \Rightarrow$

$a(defender([thursday\_best\_meeting\_day], agentA), agentB))$ **then**

$c(assert([room4\_unavailable]) \Leftarrow$

$a(defender([thursday\_best\_meeting\_day], agentA), agentB))$ **then**

$\Big($

a(interrupt_a,agentA)::=

$c(propose\_return\_control \Rightarrow a(interrupt\_b, agentB))$ **then**

$c(accept(return\_control) \Leftarrow a(interrupt\_b, agentB))$ **then**

$a(control\_layer\_a, agentA)$**::=**

$c(begin(info\_seek, room4\_has\_white\_board) \Rightarrow$

$a(control\_layer\_b, agentB))$ **then**

$c(accept(info\_seek, room4\_has\_white\_board) \Leftarrow$

$a(control\_layer\_b, agentB))$ **then**

$a(seeker(room4\_has\_white\_board, agentB), agentA)$**::=**

$c(question(room4\_has\_white\_board) \Rightarrow$

$a(provider(room4\_has\_white\_board, agentA), agentB))$ **then**

$c(assert(room4\_has\_white\_board) \Leftarrow$

$a(provider(room4\_has\_white\_board, agentA), agentB))$ **then**

$a(challenger([room4\_has\_white\_board], agentB), agentA)$**::=**

$c(accept(room4\_has\_white\_board) \Rightarrow$

$a(defender([room4\_has\_white\_board], agentA), agentB))$ **then**

$a(challenger([], agentB), agentA))$**::=** *null* **then**

$\Big)$

$a(challenger([room4\_unavailable], agentB), agentA)$**::=**

$c(accept(room4\_unavailable) \Rightarrow$

$a(defender([room4\_unavailable], agentA), agentB))$ **then**

$a(challenger([], agentB), agentA)$ **then**

$a(challenger([], agentB), agentA).$

Appendix B

Prolog Code

B.1 insert.pl

This is the code for adapting existing protocols. These predicates are reused for both dynamic protocols and protocol synthesis.

```
do_insert(  Af,At,AfNC,AtNC,Prot,
            def([Af ::= NewAfDS,At ::= NewAtDS],AC,CK)) :-
              Prot = def(DS,AC,CK),
              DS = [Af ::= AfDS,At ::= AtDS],
              insert_for_At(AtDS,AtNC,NewAtDS),
              insert_for_Af(AfDS,AfNC,NewAfDS).
```

```
do_insert(  Af,At,AfNC,AtNC,Prot,
            def([Af ::= NewAfDS,At ::= NewAtDS],ACC,CK)) :-
              Prot = def(DS,AC,CK),
              DS = [Af ::= AfDS],
              insert_for_Af(AfDS,AfNC,NewAfDS),
              copy_term(AC,ACC),
              getDSforAt(At,AC,AtDS),
              insert_for_At(AtDS,AtNC,NewAtDS).

insert_for_Af(  c(X) then Rem,NewClause,c(X) then Ret):-
                  insert_for_Af(Rem,NewClause,Ret).

insert_for_Af(  Ad ::= Rem,NewClause,Ad ::= Ret):-
                  insert_for_Af(Rem,NewClause,Ret).

insert_for_Af(  First_Open_Msg then Rem,NewClause,
                NewClause then First_Open_Msg then Rem).

insert_for_Af(  Last_Message,NewClause,
                NewClause then Last_Message).

insert_for_At(  c(X) then Rem,NewClause,c(X) then Ret):-
                  insert_for_At(Rem,NewClause,Ret).

insert_for_At(  Ad ::= Ac,NewClause,Ad ::= Ret) :-
                  insert_for_At(Ac,NewClause,Ret).

insert_for_At(  First_Open_Msg then Rem, NewClause,
                First_Open_Msg then NewClause then Rem).
```

B.2 syn.pl

This code enables protocol synthesis by translating the agent's selection of a synthesis rule into the correct LCC code.

```
syn(  A1,def([A1 ::=Clause1,A2 ::=Clause2],AC,CK),R,
      def([A1 ::=NClause1,A2 ::=NClause2],AC,CK)):-
        syn(A1,Clause1,Clause2,R,NClause1,NClause2).

syn(  A1,def([A2 ::=Clause2,A1 ::=Clause1],AC,CK),R,
      def([A2 ::=NClause2,A1 ::=NClause1],AC,CK)):-
        syn(A1,Clause1,Clause2,R,NClause1,NClause2).

syn(  A1,def([],[A1 ::=Clause1,A2 ::=Clause2],CK),R,
      def([A1 ::=NClause1,A2 ::=NClause2],
      [A1 ::=Clause1,A2 ::=Clause2],CK)):-
        syn(A1,Clause1,Clause2,R,NClause1,NClause2).

syn(  A1,def([],[A2 ::=Clause2,A1 ::=Clause1],CK),R,
      def([A2 ::=NClause2,A1 ::=NClause1],
      [A2 ::=Clause2,A1 ::=Clause1],CK)):-
        syn(A1,Clause1,Clause2,R,NClause1,NClause2).

syn(  A1,(Msg ⇒ A2 ← Constraint),(Msg ⇐ A1),
      response(Msg ← Constraint,NMsg ← Constraint2),
      (Msg ⇒ A2 ← Constraint)  then (NMsg ⇐ A2),
      (Msg ⇐ A1)  then (NMsg ⇒ A1 ← Constraint2)).
```

```
syn(  A1,(Msg ⇒ A2 ← Constraint),(Msg ⇐ A1),
      response(Msg ← Constraint,NMsg),
      (Msg ⇒ A2 ← Constraint)  then (NMsg ⇐ A2),
      (Msg ⇐ A1)  then (NMsg ⇒ A1)).

syn(  A1,(Msg ⇒ A2),(Msg ⇐ A1),
      response(Msg,NMsg ← Constraint),
      (Msg ⇒ A2)  then (NMsg ⇐ A2),
      (Msg ⇐ A1)  then (NMsg ⇒ A1 ← Constraint)).

syn(  A1,(Msg ⇒ A2),(Msg ⇐ A1),
      response(Msg,NMsg),
      (Msg ⇒ A2)  then (NMsg ⇐ A2),
      (Msg ⇐ A1)  then (NMsg ⇒ A1)).
```

B.3 rawinterface.pl

This is the interface for the expansion engine, the agent's deliberations, and the mechanisms for protocol adaptations. This is adapted code from the basic framework. Debugging and Console predicates have been removed for brevity.

```
load_agent(  Name) :-
               agent_dir(Path),
               concat_list([Path,'/',Name,'.agent'],File),
               see(File),
               read_agent,
               seen.

read_agent  :-
              read(Clause),
              \+ Clause = end_of_file, !,
              assertz(Clause),
              read_agent.
              read_agent.

institution(  I, Role, Id) :-
                load_institution(I, Prot),
                postit(a(Role,Id), [], Prot).

switcheroo(  [m(Af,M ⇒ At)|T],Af,At,[m(At,M ⇐ Af)|NT]) :-
               switcheroo(T,_,_,NT). switcheroo([],_,_,[]).

agent_id_from_role(  a(_,Id), Id).
```

```
send_prot_msgs(  [m(Af,M ⇒ At)|T], Prot) :-
                   agent_id_from_role(Af, From), nonvar(From),
                   agent_id_from_role(At, To), nonvar(To),
                    send_msg(  From, To,
                               protocol(m(Af,M ⇒ At),Prot)),
                   send_prot_msgs(T, Prot).
                   send_prot_msgs([], _).

send_msg(  From, To, Message) :-
             find_server(Server, PID),
             add_msg(Server, PID, From, To, Message).

retrieve_msgs(  From, To, List):-
                  find_server(Server,PID),
                  linda_client(Server:PID),
                  bagof_in_noblock(Server,PID,From,To,List),
                  close_client,!.

replace_msgs(  From,To,[H|T]) :-
                 find_server(Server,PID),
                 add_msg(Server,PID,From,To,H),
                 replace_msgs(From,To,T).

replace_msgs(  _,_,[]).
```

```
react(  From,Id) :-
          retrieve_msgs(From,Id,List),
          List = [protocol(_,Prot)|_],
          bagof(M,P^member(protocol(M,P),List),MS),
          switcheroo(MS,Af,At,NewMS),
          postit(At,NewMS, Prot).

postit(  Role, IMsgs, Prot) :-
           adaptprotocol(Role,Prot,NewProt),
            expansion(  Role, IMsgs, [], NewProt,
                        RMsgs, Msgs, EProt),
           RMsgs = [],
           agent_decision(EProt),
           send_prot_msgs(Msgs, EProt).

adaptprotocol(  Role,Prot,TProt) :-
                  decide2adapt(Af,Role,AfNC,AtNC),
                  do_insert(Af,Role,AfNC,AtNC,Prot,TProt),

adaptprotocol(  _,Prot,Prot).
```

B.4 syninterface.pl

This is the interface between the LCC framework and expansion engine, the agent's deliberations, and the protocol synthesis mechanism. This is adapted code from the basic framework. Debugging and Console predicates have been removed for brevity.

```
institution(  I, Role, Id) :-
                load_institution(I, Prot),
                postit(a(Role,Id), [], Prot).

react(  From,Id) :-
          retrieve_msgs(From,Id,List),
          List = [protocol(_,Prot)|_],
          bagof(M,P^member(protocol(M,P),List),MS),
          switcheroo(MS,Af,At,NewMS),
          postit(At,NewMS, Prot).

switcheroo(  [m(Af,M ⇒ At)|T],Af,At,[m(At,M ⇐ Af)|NT]) :-
              switcheroo(T,_,_,NT). switcheroo([],_,_,[]).

synandexpand(  Role,IMsgs, RMsgs, Msgs,Prot,EProt) :-
                question(Role,Prot,SProt),
                Prot = SProt,
                 expansion(  Role, IMsgs, [], SProt,
                             RMsgs, Msgs, EProt),
                SProt = EProt.
```

```
synandexpand(  Role,IMsgs, RMsgs, Msgs,Prot,SEProt) :-
                question(Role,Prot,SProt),
                 expansion(  Role, IMsgs, [], SProt,
                             RMsgs, Msgs, EProt),
                 synandexpand(  Role,IMsgs, RMsgs,
                                Msgs,EProt,SEProt).

postit(  Role, IMsgs, Prot) :-
           synandexpand(Role,IMsgs, RMsgs, Msgs,Prot,SEProt),
           RMsgs = [],
           agent_decision(SEProt),
           send_prot_msgs(Msgs, SEProt).

agent_id_from_role(  a(_,Id), Id).

send_prot_msgs(  [m(Af,M ⇒ At)|T], Prot) :-
                  agent_id_from_role(Af, From), nonvar(From),
                  agent_id_from_role(At, To), nonvar(To),
                   send_msg(  From, To,
                              protocol(m(Af,M ⇒ At),Prot)),
                  send_prot_msgs(T, Prot).
                  send_prot_msgs([], _).

send_msg(  From, To, Message) :-
             find_server(Server, PID),
             add_msg(Server, PID, From, To, Message).
```

```
retrieve_msgs(  From, To, List):-
                  find_server(Server,PID),
                  linda_client(Server:PID),
                  bagof_in_noblock(Server,PID,From,To,List),
                  close_client,!.

replace_msgs(  From,To,[H|T]) :-
                 find_server(Server,PID),
                 add_msg(Server,PID,From,To,H),
                 replace_msgs(From,To,T).

replace_msgs(  _,_,[]).

question(  Role,Prot,SProt) :-
             dosyn(Role,Prot,SProt).

dosyn(  Af,Prot,Final) :-
          decide2synthesise(Prot,Rule),
          syn(Af,Prot,Rule,TProt),
          question(Af,TProt,Final).

dosyn(  _,Prot,Prot).

bagof_in_noblock(  Server,PID,From,To,[H|T]):-
                     in_noblock(msg(From,To,H)),!,
                     bagof_in_noblock(Server,PID,From,To,T).

bagof_in_noblock(  _,_,_,_,[]).
```

Bibliography

[Amgoud et al., 2000] Amgoud, L., Maudet, N., and Parsons, S. (2000). Modeling dialogues using argumentation. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, page 31. IEEE Computer Society.

[Aristotle, 1997] Aristotle (1997). *Topics*. Clarendon Press.

[Asher and Gillies, 2003] Asher, N. and Gillies, A. (2003). Common ground, corrections and coordination. *Argumentation*, 17(4):481–512.

[Austin, 1962] Austin, J. L. (1962). *How to do Things With Words*. Oxford University Press.

[Bel-Enguix and Jiménez-López, 2005] Bel-Enguix, G. and Jiménez-López, M. D. (2005). A multi-agent system model of dialogue.

[Bellifemine et al., 1999] Bellifemine, F., Poggi, A., and Rimassa, G. (1999). JADE - a FIPA-compliant agent framework. In *Proceedings of PAAM'99*, pages 97–108, LONDON. PAMM.

[Besana et al., 2005] Besana, P., Robertson, D., and Rovatsos, M. (2005). Exploiting interaction contexts in p2p ontology mapping. Accepted for Workshop P2PKM'05.

[Bodenstaff et al., 2006] Bodenstaff, L., Prakken, H., and Vreeswijk, G. (2006). On formalising dialogue systems for argumentation in the event calculus. In *Proceedings of the 11th Workshop on Nonmonotonic Reasoning*, pages 374–382.

[Carriero and Gelernter, 1989] Carriero, N. and Gelernter, D. (1989). Linda in context. In *Comm. of the ACM*, volume 32:4, pages 444–458. ACM Press.

[Chesnevar et al., 2005] Chesnevar, M. C. C., McGinis, J., Rahwan, I., Reed, C., Modgil, S., Simari, G., South, M., Vreeswijk, G., and Willmott, S. (2005). Aif: Argumentation interchange format strawman model. Technical report, Agentlink.

[Cohen and Levesque, 1995] Cohen, P. R. and Levesque, H. J. (1995). Communicative actions for artificial agents. In Lesser, V. and Gasser, L., editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 65–72, San Francisco, CA, USA. The MIT Press: Cambridge, MA, USA.

[Core and Allen, 1997] Core, M. G. and Allen, J. F. (1997). Coding dialogues with the DAMSL annotation scheme. In Traum, D., editor, *Working Notes: AAAI Fall Symposium on Communicative Action in Humans and Machines*, pages 28–35, Menlo Park, California. American Association for Artificial Intelligence.

[Cortés, 2004] Cortés, U. (2004). Electronic institutions and agents. *AgentLink News,*, (15):14–15.

[Dash et al., 2003] Dash, R., Parkes, D., and Jennings, N. (2003). Computational mechanism design: A call to arms.

[Dastani, 2001] Dastani, M. (2001). Negotiation protocols and dialogue games. In Müller, J. P., Andre, E., Sen, S., and Frasson, C., editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 180–181, Montreal, Canada. ACM Press.

[de Silva, 2002] de Silva, L. P. (2002). Extending agents by transmitting protocols in open systems. Honours, RMIT University, Melbourne, Australia.

[Dignum, 2003] Dignum, V. (2003). *A model for organizational interaction. Based on Agents, Founded in logic*. Phd thesis, Utrecht University.

[Dung, 1995] Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358.

[Estava et al., 2001] Estava, M., Rodriguez, J. A., Sierra, C., Garcia, P., and Arcos, J. L. (2001). On the formal specifications of electronic institutions. *LNAI*, pages 126–147.

[Esteva et al., 2002] Esteva, M., de la Cruz, D., and Sierra, C. (2002). Islander: an electronic institutions editor. In *Proceeding of the first International joint conference on Automomous agents and multiagent systems*, pages 1045–1052, Bologna, Italy. ACM press.

[Esteva et al., 2000] Esteva, M., Rodríguez-Aguilar, J. A., Arcos, J. L., Sierra, C., and Garcia, P. (2000). Institutionalising open multi-agent systems. In *proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS'2000)*, pages 381–83, Boston. ICMAS.

[Esteva et al., 2004] Esteva, M., Rosell, B., Rodriguez-Aguilar, J. A., and Acros, J. L. (2004). Ameli: An agent-based middleware for electronic institutions. In *Proceedings of the International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*.

[Finin et al., 1994] Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an Agent Communication Language. In Adam, N., Bhargava, B., and Yesha, Y., editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA. ACM Press.

[FIPA, 2001] FIPA (2001). Communicative act library specification.

[Flores and Kremer, 2002] Flores, R. A. and Kremer, R. (2002). To commit or not to commit: Modelling agent conversations for action. *Computational Intelligence*, 18(2):120–173.

[Freire and Botelho, 2002] Freire, J. and Botelho, L. (2002). Executing explicitly represented protocols.

[Gautama, 2003] Gautama, A. (2003). *Nyaya Sutras of Gotama*. Munshiram Manoharlal Publishers.

[Genesereth and Fikes, 1992] Genesereth, M. R. and Fikes, R. E. (1992). Knowledge interchange format, version 3.0 reference manual. Technical Report Technical Report Logic-92-1, Standford University.

[Gilbert, 1994] Gilbert, M. (1994). Multi-modal argumentation. In *Philosophy of the Social Sciences*, pages 159–177. SAGE publications.

[Ginzburg, 1996] Ginzburg, J. (1996). Dynamics and the semantics of dialogue. In Seligman, J. and Westerståhl, D., editors, *Logic, Language, and Computation*, pages 221–237. CSLI, Stanford, Ca.

[Grando and Walton, 2006] Grando, A. and Walton, C. (2006). Mapa: a language for modelling conversations in agent environments. In *Proceedings of the Intelligent Information Processing and Web Mining Conference 2006 (IIPWM06)*, Ustron, Poland.

[Greaves et al., 2000] Greaves, M., Holmback, H., and Bradshaw, J. (2000). What is a conversation policy? In Dignum, F. and Greaves, M., editors, *Issues in Agent Communication*, pages 118–131. Springer-Verlag: Heidelberg, Germany.

[Habermas, 1991] Habermas, J. (1991). Comments on john searle: 'meaning, communication and representation'. In *John Searle and his Critics*. Cambridge Press.

[Hassan et al., 2005] Hassan, F., Robertson, D., and Walton, C. (2005). Addressing constraint failures in an agent interaction protocol. In *In Proceedings of the 8th Pacific Rim International Workshop on Multi-Agent Systems*, Kuala Lumpur.

[Johnson et al., 2002] Johnson, M. W., McBurney, P., and Parsons, S. (2002). When are two protocols the same?

[Kakas et al., 2004] Kakas, A. C., Torroni, P., and Demetriou, N. (2004). Agent planning, negotiation, and control of operation. In Lopez de Mantaras, R. and Saitta, L., editors, *Proceedings of the Sixteenth European Conference on Artificial Intelligence, Valencia, Spain (ECAI 2004)*, pages 28–32. IOS Press.

[Labrou and Finin, 1997] Labrou, Y. and Finin, T. (1997). Comments on the specification for fipa '97 agent communication language. http://www.cs.umbc.edu/kqml/papers/fipa/comments.shtml.

[Labrou et al., 1999] Labrou, Y., Finin, T., and Peng, Y. (1999). Agent communication languages: The current landscape. *IEEE Intelligent Systems*, 14(2):45–52.

[Lambert and Robertson, 2005] Lambert, D. and Robertson, D. (2005). Matchmaking multi-party interactions using historical performance data. In *AAMAS*, pages 611–617.

[Lebbink et al., 2003] Lebbink, H., Witteman, C., and Ch, J. (2003). Dialogue games for inconsistent and biased information.

[López, 2003] López, F. (2003). *Social Power and Norms: Impact on agent behaviour*. Phd thesis, University of Southampton.

[Martin et al., 1996] Martin, F. J., Plaza, E., Rodríguez-Aguilar, J. A., and Sabater, J. (1996). Jim - a java interagent for multi-agent systems.

[Maudet and Chaib-draa, 2002] Maudet, N. and Chaib-draa, B. (2002). Commitment-based and dialogue-game based protocols: new trends in agent communication languages. *The Knowledge Engineering Review*, 17(2).

[Maudet and Evrard, 1998] Maudet, N. and Evrard, F. (1998). A generic framework for dialogue game implementation.

[Mayfield et al., 1995] Mayfield, J., Labrou, Y., and Finin, T. (1995). Desiderata for agent communication languages. In *AAAI Spring Symposium on Information Gathering*.

[McBurney and Parsons, 2001] McBurney, P. and Parsons, S. (2001). Representing epistemic uncertainty by means of dialectical argumentation. *Annals of Mathematics and Artificial Intelligence*.

[McBurney and Parsons, 2002] McBurney, P. and Parsons, S. (2002). Games that agents play: A formal framework for dialogues between autonomous agents. *Journal of Logic, Language and Information*, 11(3):315–334.

[McBurney and Parsons, 2003] McBurney, P. and Parsons, S. (2003). Dialogue games in multi-agent systems. *Informal Logic. Special Issue on Applications of Argumentation in Computer Science*.

[McBurney and Parsons, 2004] McBurney, P. and Parsons, S. (2004). Locutions for argumentation in agent interaction protocols. In *Proceedings of the Workshop on Agent Communication*, LNAI. Springer-Verlag.

[McBurney et al., 2002a] McBurney, P., Parsons, S., and Wooldridge, M. (2002a). Desiderata for agent argumentation protocols.

[McBurney et al., 2002b] McBurney, P., van Eijk, R., Parsons, S., and Amgoud, L. (2002b). A dialogue-game protocol for agent purchase negotiations. *Journal of Autonomous Agents and Multi-Agent Systems. (In press)*.

[McGinnis and Robertson, 2004a] McGinnis, J. and Robertson, D. (2004a). Dynamic and distributed interaction protocols. In *Proceedings of the AISB 2004 Convention*, pages 45–54.

[McGinnis and Robertson, 2004b] McGinnis, J. and Robertson, D. (2004b). Realizing agent dialogues with distributed protocols. In *Developments in Agent Communication*, volume 3396 of *LNAI*. Springer-Verlag.

[McGinnis et al., 2003] McGinnis, J., Robertson, D., and Walton, C. (2003). Using distributed protocols as an implementation of dialogue games. Presented EUMAS 2003.

[Nodine and Unruh, 1997] Nodine, M. H. and Unruh, A. (1997). Facilitating open communication in agent systems: the infosleuth infrastructure. In *Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages*.

[Noriega, 1997] Noriega, P. (1997). *Agent-Mediated Austions:The Fishmarket Metaphor*. PhD thesis, Institut d'Investigacio en Intelligencia Artificial (IIIA).

[Osman et al., 2006] Osman, N., Robertson, D., and Walton, C. (2006). Run-time model checking of interaction and deontic models for multi- agent systems. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS06)*, Hakodate, Japan.

[Parsons, 2002] Parsons, S. (2002). *Game Theory and Decision Theory in Agent-based Systems*, volume 5 of *"Multi-agent Systems" International Book Series*. Kluwer Academic Publishers.

[Parsons et al., 2003a] Parsons, S., McBurney, P., and Wooldridge, M. (2003a). The mechanics of some formal inter-agent dialogues. In *Workshop on Agent Communication Languages*, pages 329–348.

[Parsons et al., 2004] Parsons, S., McBurney, P., and Wooldridge, M. (2004). Some preliminary steps towards a meta-theory for formal inter-agent dialogues. In *Proceedings of the First International Workshop on Argumentation in Multi-Agent Systems*.

[Parsons et al., 2002] Parsons, S., Wooldridge, M., and Amgoud, L. (2002). An analysis of formal inter-agent dialogues. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 394–401. ACM Press.

[Parsons et al., 2003b] Parsons, S., Wooldridge, M., and Amgoud, L. (2003b). On the outcomes of formal inter-agent dialogues. In *2nd International Conference on Autonomous Agents and Multiagent Systems*, Melbourne, Australia.

[Paun and Salomaa, 1999] Paun, G. and Salomaa, A., editors (1999). *Grammatical Models of Multi-Agent Systems*, volume 8 of *Topics in Computer Mathematics*. Gordon and Breach Science Publishers.

[Paurobally, 2002] Paurobally, S. (2002). *Rational Agents and the processes and states of negotiation*. Phd thesis, Imperial College, London.

[Paurobally et al., 2004] Paurobally, S., Cunningham, J., and Jennings, N. R. (2004). Verifying the contract net protocol: A case study in interaction protocol and agent communication language semantics.

[Pitt and Mamdani, 1999] Pitt, J. and Mamdani, A. (1999). A protocol-based semantics for an agent communication language. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 486–491, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Ramchurn et al., 2004] Ramchurn, S. D., Hunyh, D., and Jennings, N. R. (2004). Trust in multi-agent systems. *Knowledge Engineering Review*.

[Rao, 1996] Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In van Hoe, R., editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands.

[Reed, 1998] Reed, C. (1998). Dialogue frames in agent communication. In Demazeau, Y., editor, *Proceedings of the Third International Conference on Multi-Agent Systems(ICMAS-98)*, pages 246–253. IEEE Press.

[Rickard, 1989] Rickard, P. (1989). *A History of the French Language*. Routledge (UK).

[Robertson, 2004a] Robertson, D. (2004a). A lightweight coordination calculus for agent social norms. In *Declarative Agent Languages and Technologies*, New York, USA. a full day workshop occuring as part of AAMAS'04.

[Robertson, 2004b] Robertson, D. (2004b). A lightweight method for coordination of agent oriented web services. In *Proceedings of AAAI Spring Symposium on Semantic Web Services*, California, USA.

[Robertson, 2004c] Robertson, D. (2004c). Multi-agent coordination as distributed logic programming. In *Proceedings for International Conference on Logic Programming*.

[Rovatsos, 2004] Rovatsos, M. (2004). *Computational Interaction Frames*. PhD thesis, Department of Informatics, Technical University of Munich.

[Sadek, 1991] Sadek, M. D. (1991). *Attitudes Mentales et Interaction Rationnelle: Vers une Thorie Formelle de la Communication*. PhD thesis, Universit de Rennes I.

[Searle, 1969a] Searle, J. (1969a). *(on) Searle on Communication*. Cambridge University Press.

[Searle, 1969b] Searle, J. (1969b). *Speech Acts*. Cambridge University Press.

[Singh, 2000] Singh, M. P. (2000). A social semantics for agent communication languages. In Dignum, F. and Greaves, M., editors, *Issues in Agent Communication*, pages 31–45. Springer-Verlag: Heidelberg, Germany.

[the Intelligent Systems Laboratory, 1987] the Intelligent Systems Laboratory (1987). *Sicstus Prolog Users Manual*. Swedish Institute of Computer Science, Sweden.

[Vasconcelos, 2002] Vasconcelos, W. (2002). Skeleton-based agent development for electronic institutions. In *First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy. AAMAS, ACM Press.

[Vázquez-Salceda, 2003] Vázquez-Salceda, J. (2003). *The role of norms and electronic institutions in multi-agent systems applied to complex domains the HARMONIA framework*. Phd thesis, Universitat Politcnica de Catalunya.

[Verdicchio and Colombetti, 2003] Verdicchio, M. and Colombetti, M. (2003). A logical model of social commitment for agent communication. In Sandholm, T. and Yokoo, M., editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 03)*, pages 528–535, Melbourne, Australia. ACM Press.

[Vreeswijk and Prakken, 2000] Vreeswijk, G. A. W. and Prakken, H. (2000). Credulous and sceptical argument games for preferred semantics. *Lecture Notes in Computer Science*, 1919:239–??

[Walton, 2004a] Walton, C. (2004a). Model Checking Multi-Agent Web Services. In *Proceedings of the AAAI Spring Symposium on Semantic Web Services*, Stanford, CA. AAAI.

[Walton and Barker, 2004] Walton, C. and Barker, A. (2004). An Agent-based e-Science Experiment Builder. In *Proceedings of the 1st International Workshop on Semantic Intelligent Middleware for the Web and the Grid*, Valencia, Spain.

[Walton and Robertson, 2002] Walton, C. and Robertson, D. (2002). Flexible multi-agent protocols. Technical Report EDI-INF-RR-0164, University of Edinburgh.

[Walton, 2004b] Walton, C. D. (2004b). Multi-Agent Dialogue Protocols. In *Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida.

[Walton and Krabbe, 1995] Walton, D. and Krabbe, E. C. W. (1995). *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY press, Albany, NY, USA.

[Wooldridge, 2000] Wooldridge, M. (2000). Semantic issues in the verification of agent communication languages. *Autonomous Agents and Multi-Agent Systems*, 3(1):9–31.