UNIVERSITY OF CALGARY

Modelling Agent Conversations for Action

by

Roberto Augusto Flores-Méndez

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

JUNE, 2002

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled “Modelling Agent Conversations for Action” submitted by Roberto Augusto Flores-Méndez in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Supervisor, Robert C. Kremer, Department of Computer Science

Brian R. Gaines, Department of Computer Science

Saul Greenberg, Department of Computer Science

Robert W. Brennan, Department of Mechanical and Manufacturing Engineering

External Reader, Hector J. Levesque, University of Toronto

Date

ABSTRACT

Conversations are sequences of messages exchanged between interacting software agents. For conversations to be meaningful, agents ought to follow conversational principles governing the exchange of messages at any point in a conversation. These principles must be defined in publicly verifiable terms (if they are to be used in open environments) and must allow the composition of flexible conversations (if they are to account for the context in which they occur). The main contribution of this thesis is to define a unified model for conversations for action that fulfills these requirements. The conversational principle in this model is the negotiation of shared social commitments, which entails the adoption and discard of obligations to act. This principle is encoded using conversation policies, which govern the form of conversations according to the observable state of interacting agents. The applicability of this model is illustrated through the modelling of two example conversations: one on the Contract Net Protocol, and the second on an electronic bookstore scenario.

ACKNOWLEDGEMENTS

I once read that writing a book is like travelling to an unfamiliar city with only a visitors' guide: with it one can tour the main landmarks and recommended sights, but the lasting memories are those of the people one meets and the relationships one forges. Pursuing a doctoral degree was the same for me: I fulfilled the academic requirements while, in the process, meeting many people who enriched both my personal and academic life—to all of them, my utmost gratitude.

First of all, I would like to thank my supervisor and friend Rob Kremer, whose constant support and encouragement was central to the successful completion of this thesis. I am also grateful to the members of my supervisory committee: Bob Brennan, Brian Gaines, Saul Greenberg and Hector Levesque, whose thoughtful suggestions greatly helped to improve the quality of the work presented in this thesis.

In addition, I am thankful to colleagues that at one moment or another during these years gave me their time and support, particularly Douglas Norrie (who was a source of inspiration and advice), Brad Jones, Michal Pechoucek, Martyn Fletcher, Frank Maurer, Jörg Denzinger, Douglas Walker, Niek Wijngaards, Fuhua Lin, and the researchers at the Intelligent Systems Group at the University of Calgary.

I am indebted to Andy and Karyn Kremer for their superb job proofreading an earlier version of this thesis (needless to say, any errors in this version are mine alone). Finally, I cannot thank enough my wife Cecilia and children Ricardo and Alejandro, who always had a way of reminding me that there is (good) life besides school.

This work would not have been possible without the support of the Natural Sciences and Engineering Council of Canada (NSERC), Smart Technologies, Inc., the Alberta Software Engineering Research Consortium (ASERC) and the Alberta Informatics Circle of Research Excellence (iCORE).

To Cecilia, Ricardo and Alejandro.

The happiness of life is made up of minute fractions—the little, soon-forgotten charities of a kiss or smile, a kind look or heartfelt compliment.

Samuel Taylor Coleridge, poet (1772-1834)

Table of Contents

List of Figures

Chapter 1 Introduction

1.1 Aim

The aim of this thesis is to define a formal model for the structured specification of software agent conversations for action.

This chapter introduces the motivations for developing such a model for conversations. This is followed by a brief introduction of the model, along with an example illustrating its dynamics and application. Lastly, this chapter ends with a list of research objectives and the outline of the thesis.

1.2 Motivation

Software agents are autonomous, collaborative problem solving entities that are increasingly being applied as a key abstraction for developing software applications (Jennings & Wooldridge, 1998). One of the main characteristics of agent-based systems is that agents seek to interact among themselves to perform tasks that help them to meet their (individual or collective) objectives. For example, agents request and offer services, schedule the delivery of parts for manufacturing processes, and negotiate the best possible deal when shopping for goods.

1.2.1 Agent Communications

Although agents could interact through any action that affects their common environment, this thesis is particularly interested in agents that interact using an agent communication language (Genesereth & Ketchpel, 1994). In this view, software agents are conceptualized as purely communicational entities (Ferber, 1999) that only interact by exchanging messages through a communications channel.[1]

1.2.2 Conversations

Conversations are the meaningful exchange of messages between interacting agents. In agent-based systems, conversations are traditionally specified using conversation protocols (which are static structures specifying the sequences of messages making a conversation) and conversation policies (which are rules of inference specifying the principles governing the connectedness of message sequences during conversations).

Although policies and protocols are normally seen as competing techniques, this research subscribes to the view that a protocol is a particular conversation structure that should be constructed following the principles specified by conversation policies. However, it still remains a challenge for the agent communication language community to define the properties and principles that conversation policies should represent (Greaves, et al., 1999).

This thesis presents a model for conversations in which conversations are guided by policies based on the principle that agents requesting the performance of actions must negotiate the uptake of shared social commitments.

---

[1] Clark (1996) states that humans use both linguistic and non-linguistic signals (i.e., events that can be observed from the shared environment of interaction) when communicating. For example, that Bob has moved a chair as requested by Alice, and that this action is apparent to both of them (for details on mutual beliefs refer to (Fagin, et al., 1995)), then it may not be required that Bob tells Alice that he has done the action. In contrast, this thesis assumes that software agents do not have any sensors to perceive each other's actions and no other shared environment except for a communications channel through which to exchange messages. The main reason for this abstraction is that (unlike humans) software programs do not have a natural shared environment (c.f., our "real world") and any assumptions on the characteristics of any given environment may compromise the generality of this thesis.

1.3 A Model for Agent Conversations

In their influential book on cognition and social action, Winograd and Flores (1987) argue that the meaning of utterances is given by the patterns of commitment entered into by speaker and hearer by virtue of taking part in a conversation. In this view, every language act has consequences for conversing agents, leading to immediate actions and to commitments for future action.

Following that trend, this thesis explores a model for conversations whose main principle is the negotiation of shared social commitments, that is, directed obligations in which one agent has a responsibility relative to another agent for the performance of an action.

For agents to mutually know that a social commitment is being shared among them, it is necessary that they know that the other agents with whom this commitment is shared also know that this commitment is shared. To be faithful to the requirement that agents are autonomous entities, these agents cannot have any indication that a commitment is shared except for what they can observe through their communications; thus, messages must exist to establish shared social commitments. However, the uttering of these messages cannot result in the automatic imposition of social commitments; instead, speakers and hearers must engage in a negotiation towards the uptake of shared social commitments. This model specifies a small protocol, which is called the *Protocol for Proposals*, as a mechanism to negotiate shared social commitments.

Although the negotiation of shared social commitments is the central piece upon which the model for conversations is founded, it is nevertheless a vehicle to an end: that of giving rise to social obligations to action. That is, by mutually agreeing to uptake social commitments agents not only adopt the shared state of these commitments, but also uptake obligations to perform the negotiated actions (if and only if these agents are specified as the performers of the actions).[2] For example, that Alice and Bob have negotiated a shared social commitment specifying an action in which Bob is to tell Alice the time of the day, creates the obligations in Bob that he has to find out the time and that he has to tell the time to Alice.

Since the role of actions is important in this model, the next section gives a brief overview of this topic.

1.3.1 Actions

The model for conversations defines a basic structure of actions from which other actions are derived.

As shown in the UML class diagram in Figure 1, this model defines two basic types of actions: *individual actions* (which are atomic actions performed by one agent) and *composite actions* (actions encompassing various other actions). *Joint actions* are defined as composite actions in which there is more than one performer. The individual actions *ToOutput* and *ToInput* are defined as actions that generate an output and receive an input, respectively. These actions are part of the joint action *ToCommunicate*, which specifies that the output of the outputting action is the same as the input of the inputting action. The action *ToCommunicate* is specialized as a *ToSpeak* action in which the outputting and inputting actions are specialized to the actions *ToVoice* and *ToHear*, respectively, and where the communicated data is a set of illocutionary points. Lastly, the action *ToProcess* is defined as an individual action that gets an input and produces an output.

[2] This thesis refers to *social obligations* (or simply as obligations) as the obligations that are established by agents through their communications (in this case through the negotiation of commitments). These obligations are different than the ones that agents actually pursue, i.e., their intentions. Although it is expected that social obligations influence the intentions of an agent, this relationship is not specified in the model here presented.

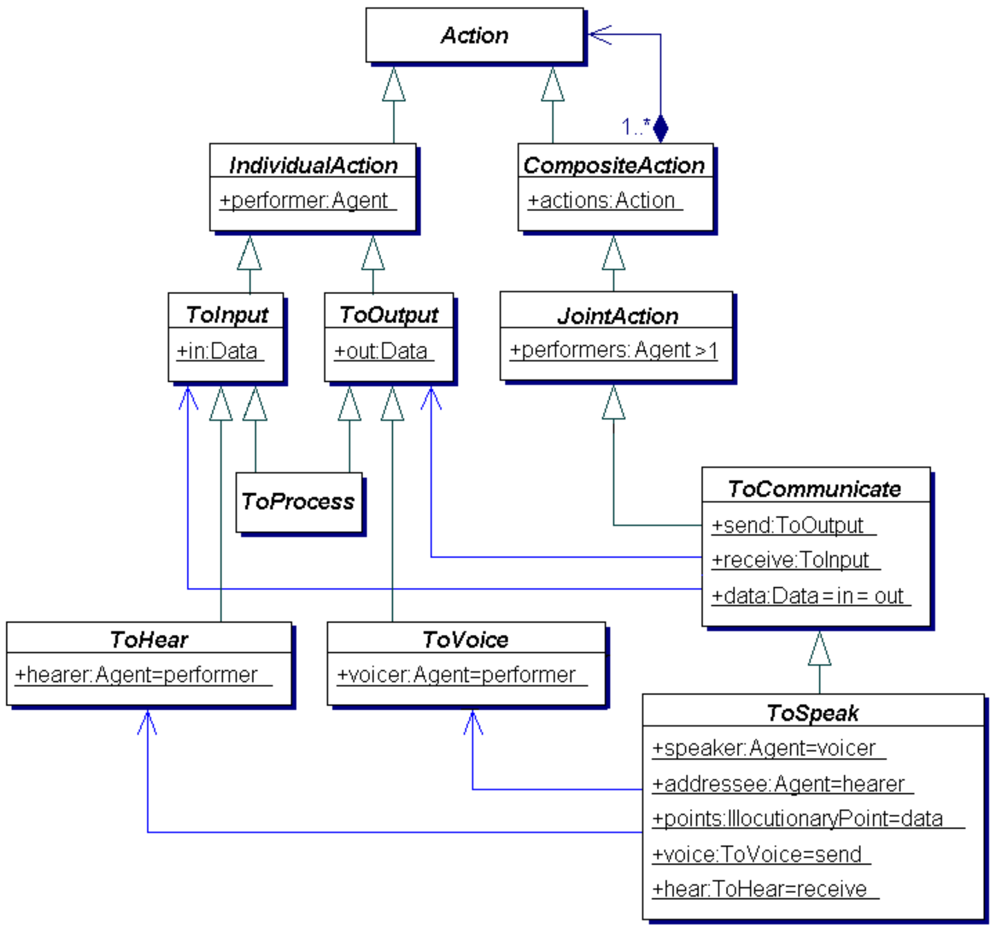


Figure 1. UML class diagram for basic actions in the model for conversations.

1.3.2 Social Commitments and Obligations

Social commitments are directed obligations in which one agent is responsible relative to another agent for the performance of an action. In this model, the adoption of shared commitments generates obligations to act upon the actions in the negotiated commitment.

In general, agents that have adopted a social commitment to perform an action have an obligation in which there is a social currency at stake, such as money or their own reputation. From this perspective, agents should be selective of the social commitments they adopt if they are to fulfill the obligations entailed by those commitments lest they risk the punishments associated with failure.

1.3.3 Speech Acts and Illocutionary Points

Agent communication languages (ACL) define the meaning of messages using notions from the speech-as-action tradition (Austin, 1962; Searle, 1975). In this tradition, utterances are actions (thus the term *speech acts*) that are performed by virtue of being uttered.

Speech acts are composed of an *illocutionary point* (signalling the intention of the act), an *illocutionary force* (signalling its imperative) and a *propositional content* (conveying the information communicated).

In the model for conversations, messages are defined as speech acts conveying a collection of illocutionary points (the illocutionary force is considered a secondary source of meaning and is not modelled). Four illocutionary points are defined to support the negotiation of shared social commitments:

- *Propose*: to put forth the adoption or discard of a shared social commitment,
- *Accept*: to accept adopting or discharging a shared social commitment,
- *Reject*: to reject adopting or discharging a social commitment, and
- *Counter*: to reject a previous proposal while putting forth another proposal to be considered instead.

A fifth illocutionary point (*Inform*) is used to communicate data (i.e., propositional content).

1.3.4 Conversation Policies

It is one thing to define illocutionary points and quite another to describe how they are used and what they can accomplish when used in conversations. The model for conversations defines the *Protocol for Proposals* (which is implemented using *conversation policies*) to guide the evolution of conversations and their potential accomplishments.

Negotiating Shared Social Commitments

The first policy indicates that the uttering of a proposal (i.e., a speech act containing a *Propose* illocutionary point) commits the hearer to reply to the proposal; that is, the hearer is committed to utter a speech act containing an *Accept*, *Reject* or *Counter* illocutionary point with identical characteristics as that of the uttered *Propose*.

The second policy indicates that the uttering of a reply (i.e., a speech act containing an *Accept*, *Reject* or *Counter* illocutionary point) discards an obligation in which the speaker is committed to utter a reply.

The third policy defines that the uttering of a *Propose* followed by an *Accept* illocutionary point results in the adoption or discard (whatever operation it is negotiated) of a shared social commitment and corresponding obligations.

Proposing to Discharge Shared Social Commitments

Once a social commitment has been accepted as shared, it is usually the case that it will be discharged after the performance of its actions (thus freeing the interacting agents of any obligations acquired by adopting the commitment).[3]

Policy 4 specifies that the adoption to commit to certain actions also commits agents to propose its discharge. To enable this behaviour, involved actions define a *discharger* agent (that proposes the discharge) and a *discharged* agent (that is proposed the discharge).

1.3.5 Example: Giving the Time

Let us imagine now a simple conversation in which Alice asks Bob for the time (e.g., "Can you tell me the time?"[4]). Assuming that Bob is willing to provide this information (e.g., he

[3] Most conversations entail obligations that are transient in nature, i.e., they bind participants only during the time of interaction. That would be the case, for example, of a dialogue to pass the salt at dinnertime ("Could you pass the salt?"—"Yes…Here it is"—"Thanks"), in which an obligation is created—that of passing the salt—and is discharged once the salt is handed out. In contrast, there are other actions that entail obligations that remain after the interaction has ended (e.g., getting married, in which certain obligations, such as to love and respect one another, remain after the wedding ceremony).

utters “Sure…”), one can expect that he will check his watch and tell Alice the time he observed (e.g., “It is 10:05 a.m.”), and finally that Alice will kindly acknowledge the effort (e.g., saying “Thanks!”).

As simple as it seems, it is not straightforward to explain what makes Alice and Bob believe they are engaged in a structured conversation. The contention in this thesis is that this conversation (and perhaps all conversations for action) can be explained as an orderly sequence of speech acts negotiating the adoption and discard of shared social commitments.

To support this claim, this section illustrates how the model for conversations supports Alice and Bob’s conversation by showing the exchange of speech acts and the accumulation and discard of shared social commitments and obligations that these speech acts bring about.

As shown in the UML class diagram in Figure 2, the action of “giving the time” is defined as a class *ToGiveTheTime* that inherits from *ToGenerate* and *ToProposeToDischarge*. In this case, Bob is the *generator* and *discharger* agent in the action (i.e., he is the performer of an action that outputs the time of the day, the speaker in a communication where the time is informed, and the speaker of a proposal to discharge the commitment of giving the time), and Alice is the *receiver* and *discharged* agent (i.e., the agent who is informed the time, and the agent who is proposed the discharge of the commitment of giving the time).

[4] According to linguistic pragmatics, the simplest way to communicate meaning is to make use of utterances that mean literally what they say (e.g., “You tell me the time.”) However, it has been argued that this type of expression is not often used in human discourse because it is more socially acceptable to seek agreement than to impose actions onto others (Tannen, 1986). As a result, humans regularly make use of *indirect speech acts* (Searle, 1975), that is, utterances whose meaning is to be inferred. That is the case with the utterance “Do you know what time it is?”, where the intention is not to find out whether or not the hearer knows about the time (a yes/no question) but to request the hearer to do something else, like informing the time to the speaker, or remind the hearer that she is running late for an appointment. In this thesis, indirect speech is regarded as the outcome of cognitive ability to recognize meaning by virtue of the context shared between speaker and hearer, and it is not part of the model for conversations. Nevertheless, this type of speech is used in some examples to illustrate conversations in terms of colloquial human discourse.

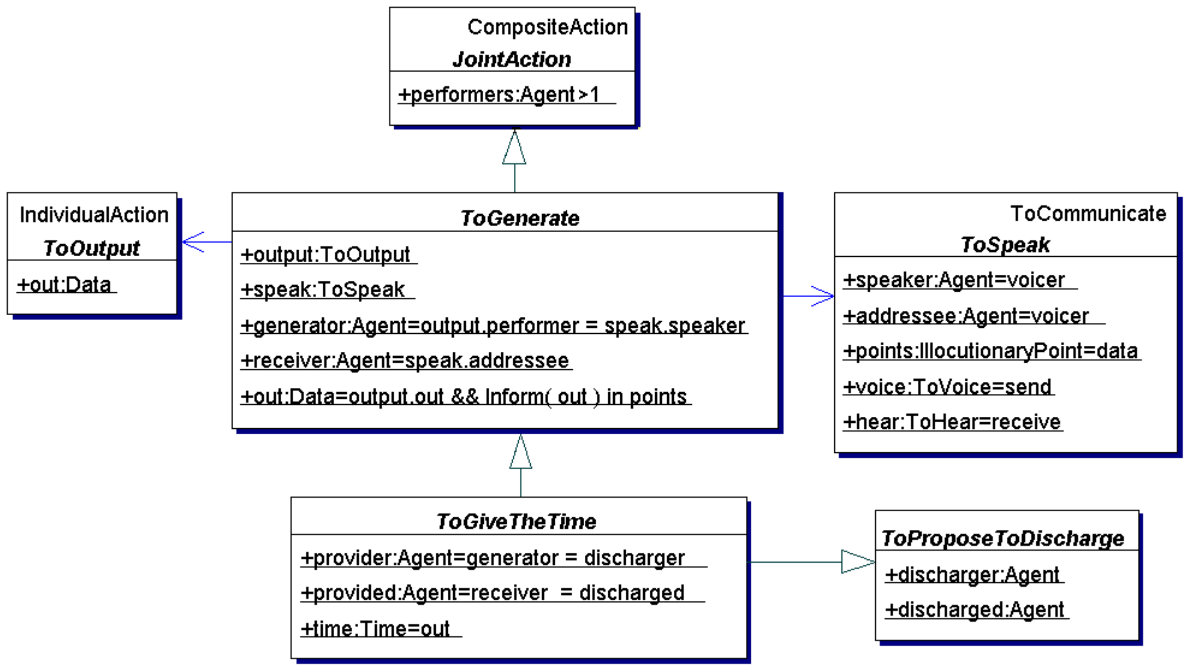


Figure 2. UML class diagram for the joint action *ToGiveTheTime*.

Figure 3 illustrates the conversation between Alice and Bob. This conversation begins with Alice's request that Bob give her the time.[5] This request takes the form of a speech act containing a *Propose* illocutionary point proposing to adopt as shared the social commitment that Bob commits to Alice to inform her the time. By virtue of its illocution (and according to Policy 1), this proposal creates the expectation that Bob will reply to Alice, that is, it creates obligations in which Bob is to reply to Alice's proposal to adopt the commitment. This is shown in Bob's obligations to participate in a *Speaking* joint activity in which he is the *speaker*, and in Alice's obligations in which she participates as the *hearer*.

This utterance is followed by Bob's acceptance of Alice's proposal ("Sure…"), which leads to the discard of the obligation to reply (Policy 2) and the uptake of the shared commitment (along with the obligations it entails), in which Bob informs Alice the time (Policy 3). In

[5] The representation of utterances, commitments and obligations in the figure was simplified for clarity.

addition, this acceptance makes Alice and Bob acquire the obligation that Bob is responsible to propose the discard of the commitment to give the time (Policy 4).

When Bob later informs the time to Alice ("It is 10:05 a.m.") he is not simply communicating the time but instead he is proposing to discard the shared commitment (and its corresponding obligations) that he informs the time by providing this information along with the proposal. Given that this utterance is a proposal it follows that there is now the expectation that Alice will reply to it (Policy 1).

Lastly, when Alice accepts to discard the commitment (by uttering "Thanks!"), she causes the discharge of the obligation that she will reply to Bob's proposal to discard the commitment to inform the time (Policy 2), as well as the shared commitment and obligations in which Bob is to inform her the time (Policy 3). This acceptance also results in the discard of the obligation that Bob is to propose to discard the agreed social commitment (Policy 4). This last utterance leaves no shared commitments or obligations.

As shown in this example, the model for conversations uses the negotiation of social commitments as a semantics to structure agent conversations for action. The remainder of the thesis will show in detail how this model is applied to structure conversations.

1.4 Research Objectives

The objectives for this research are the following:

1. To survey the state of the art on agent communication languages to determine their adequacy to support agent conversations.
2. To define the requirements that a model for conversations should support.
3. To propose a model for conversations that supports these requirements.
4. To evaluate the model for conversations in a range of practical domains.
5. To propose further research based on the experiences obtained.

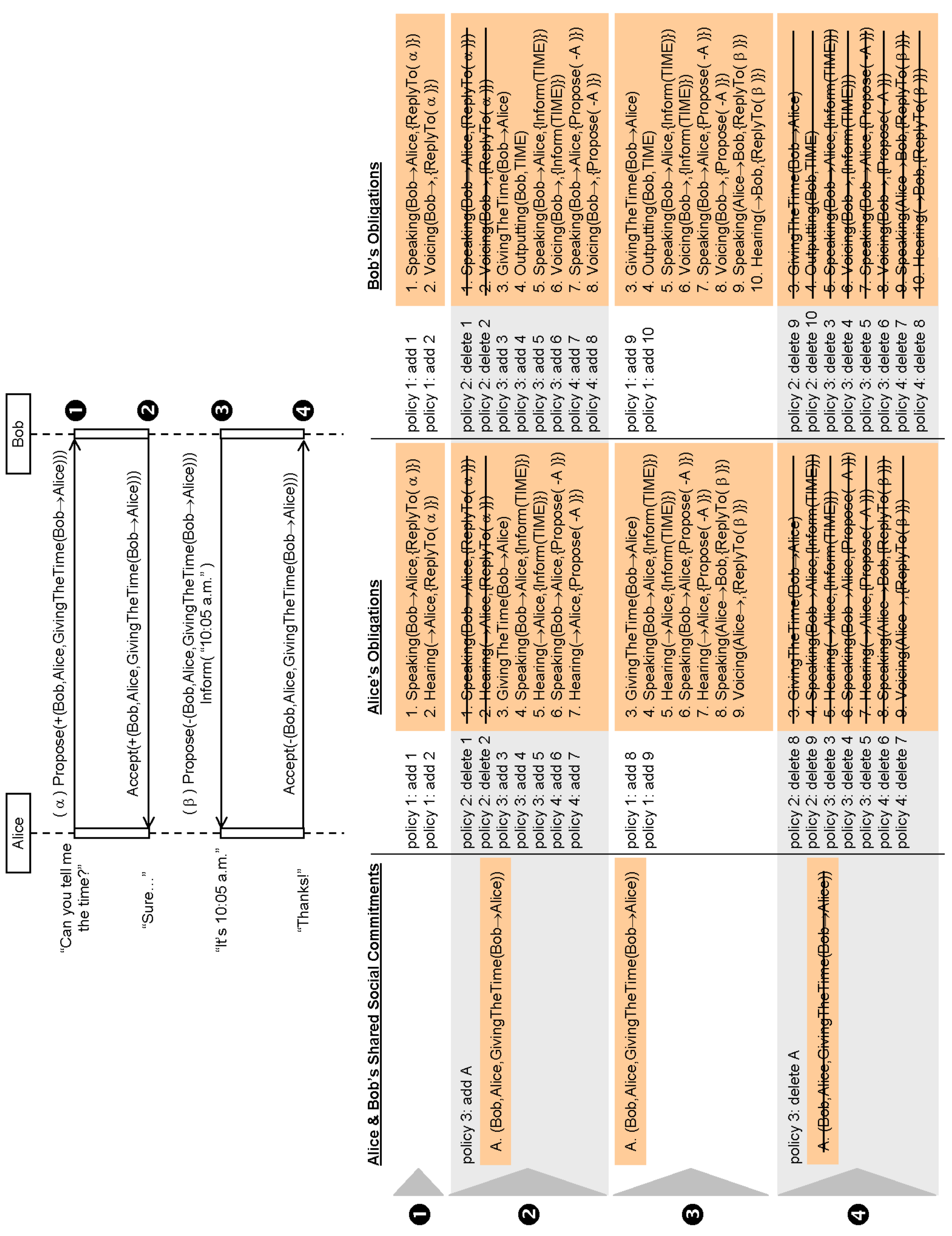

Alice
Bob
"Can you tell me the time?"
"Sure…"
"It's 10:05 a.m."
"Thanks!"
(α) Propose(+(Bob,Alice,GivingTheTime(Bob→Alice)))
Accept(+(Bob,Alice,GivingTheTime(Bob→Alice)))
(β) Propose(-(Bob,Alice,GivingTheTime(Bob→Alice)) Inform("10:05 a.m."))
Accept(-(Bob,Alice,GivingTheTime(Bob→Alice)))
1
2
3
4
Alice & Bob's Shared Social Commitments
Alice's Obligations
Bob's Obligations
policy 1: add 1
policy 1: add 2
1. Speaking(Bob→Alice,{ReplyTo(α)})
2. Hearing(→Alice,{ReplyTo(α)})
policy 1: add 1
policy 1: add 2
1. Speaking(Bob→Alice,{ReplyTo(α)})
2. Voicing(Bob→,{ReplyTo(α)})
policy 3: add A
A. (Bob,Alice,GivingTheTime(Bob→Alice))
policy 2: delete 1
policy 2: delete 2
policy 3: add 3
policy 3: add 4
policy 3: add 5
policy 3: add 6
policy 4: add 7
~~1. Speaking(Bob→Alice,{ReplyTo(α)})~~
~~2. Hearing(→Alice,{ReplyTo(α)})~~
3. GivingTheTime(Bob→Alice)
4. Speaking(Bob→Alice,{Inform(TIME)})
5. Hearing(→Alice,{Inform(TIME)})
6. Speaking(Bob→Alice,{Propose(-A)})
7. Hearing(→Alice,{Propose(-A)})
policy 2: delete 1
policy 2: delete 2
policy 3: add 3
policy 3: add 4
policy 3: add 5
policy 3: add 6
policy 4: add 7
policy 4: add 8
~~1. Speaking(Bob→Alice,{ReplyTo(α)})~~
~~2. Voicing(Bob→,{ReplyTo(α)})~~
3. GivingTheTime(Bob→Alice)
4. Outputting(Bob,TIME)
5. Speaking(Bob→Alice,{Inform(TIME)})
6. Voicing(Bob→,{Inform(TIME)})
7. Speaking(Bob→Alice,{Propose(-A)})
8. Voicing(Bob→,{Propose(-A)})
A. (Bob,Alice,GivingTheTime(Bob→Alice))
policy 1: add 8
policy 1: add 9
3. GivingTheTime(Bob→Alice)
4. Speaking(Bob→Alice,{Inform(TIME)})
5. Hearing(→Alice,{Inform(TIME)})
6. Speaking(Bob→Alice,{Propose(-A)})
7. Hearing(→Alice,{Propose(-A)})
8. Speaking(Alice→Bob,{ReplyTo(β)})
9. Voicing(Alice→,{ReplyTo(β)})
policy 1: add 9
policy 1: add 10
3. GivingTheTime(Bob→Alice)
4. Outputting(Bob,TIME)
5. Speaking(Bob→Alice,{Inform(TIME)})
6. Voicing(Bob→,{Inform(TIME)})
7. Speaking(Bob→Alice,{Propose(-A)})
8. Voicing(Bob→,{Propose(-A)})
9. Speaking(Alice→Bob,{ReplyTo(β)})
10. Hearing(→Bob,{ReplyTo(β)})
policy 3: delete A
~~A. (Bob,Alice,GivingTheTime(Bob→Alice))~~
policy 2: delete 8
policy 2: delete 9
policy 3: delete 3
policy 3: delete 4
policy 3: delete 5
policy 4: delete 6
policy 4: delete 7
~~3. GivingTheTime(Bob→Alice)~~
~~4. Speaking(Bob→Alice,{Inform(TIME)})~~
~~5. Hearing(→Alice,{Inform(TIME)})~~
~~6. Speaking(Bob→Alice,{Propose(-A)})~~
~~7. Hearing(→Alice,{Propose(-A)})~~
~~8. Speaking(Alice→Bob,{ReplyTo(β)})~~
~~9. Voicing(Alice→,{ReplyTo(β)})~~
policy 2: delete 9
policy 2: delete 10
policy 3: delete 3
policy 3: delete 4
policy 3: delete 5
policy 3: delete 6
policy 4: delete 7
policy 4: delete 8
~~3. GivingTheTime(Bob→Alice)~~
~~4. Outputting(Bob,TIME)~~
~~5. Speaking(Bob→Alice,{Inform(TIME)})~~
~~6. Voicing(Bob→,{Inform(TIME)})~~
~~7. Speaking(Bob→Alice,{Propose(-A)})~~
~~8. Voicing(Bob→,{Propose(-A)})~~
~~9. Speaking(Alice→Bob,{ReplyTo(β)})~~
~~10. Hearing(→Bob,{ReplyTo(β)})~~

Figure 3. UML interaction diagram for Alice and Bob's conversation to get the time.

As will be shown in the remaining chapters of this thesis, these objectives support the aim of defining a semantic model for the structured specification of agent conversations for action.

1.5 Thesis Overview

This first chapter succinctly describes the motivations for defining a model for conversations, mainly as the means to support software agent interoperability. This model uses speech acts and conversation policies as the core notions to guide the negotiation of shared social commitments. The dynamics of this model were shown through a brief example about requesting the time of the day.

The remaining chapters of the thesis unfold as follows. Chapter 2 describes related work in the area of agent communication languages; this chapter highlights the main drawbacks of current ACL methodologies in their approach to conversations, and lists the desiderata for a model for conversations. Chapter 3 describes in detail the presented model for conversations, which claims to support structured conversations for action. Chapters 4 and 5 present thorough examples showing how this model supports conversations in different domains of application. Chapter 6 presents an evaluation of this model, a brief description of related and future work, and concludes with a review of the research objectives set in the first chapter.

Chapter 2 Related Work

2.1 Overview

This chapter describes current approaches to agent communication languages. These approaches conceptualize conversations in terms of speech acts and conversation protocols.

This chapter reviews these approaches, highlighting their weaknesses in supporting agent interoperability, and finally describes desired requirements for a model for conversations for open environments.

2.2 The Study of Natural Language

In linguistics, natural language is studied under three broad domains: *syntax* (which dictates the rules for combining words into well-formed sentences), *semantics* (which specifies the literal meaning of sentences regardless of context) and *pragmatics* (which specifies the meaning that arises from specific contexts of use).

The defining element between semantics and pragmatics is their appeal to context, that is, to the set of circumstances in which utterances occur. As explained by Clark in his description on language use (Clark, 1996), context is the current common ground among conversing agents, including the physical and social settings of interaction and the knowledge that these agents share.

2.3 Speech Act Theory

A sub-field of linguistic pragmatics is that of *speech act theory*, proposed by Austin (1962). In this theory, Austin argues that there are utterances that cannot be catalogued according to their truth or falsity (which was the prevalent view at that time), but rather according to their *felicity*, that is, their adequacy and appropriateness to the context in which they are uttered.

Austin advocated that utterances are acts that people perform in speaking: specifically that utterances are intended to get hearers to do things based on their understanding of what the speaker meant. According to Austin's terminology, an utterance by a speaker is a *perlocutionary act* and the consequent action by the hearer is the *perlocutionary effect*. Perlocutionary effects, however, are the result of the hearer's interpretation of an utterance and do not necessarily reflect the speaker's intention. Therefore, Austin defined the act of getting the hearer to recognize the speaker's meaning as an *illocutionary act* and the recognition itself is called the *illocutionary effect*.

Searle (1975) complemented Austin's work by proposing a classification of speech acts. Searle argued that speech acts have three identifying elements: an *illocutionary point* (i.e., the publicly intended perlocutionary effect), an *illocutionary force* (i.e., the manner and degree), and a *propositional content* (i.e., the expression indicating subjective information).

Searle classified speech acts based on their illocutionary point as belonging to one of the following five types: [6]

- *assertives*: in which the speaker expresses a proposition (e.g., to inform, to notify),
- *directives*: in which the speaker attempts to get the hearer to do something (e.g., to request, to command),

[6] Clark (1996) acknowledges that this classification has worked for researchers as a point of reference for the study of speech acts. Nevertheless, he argues that there are at least two problems with this categorization: a) that it is not absolute, i.e., it does not account for all potential illocutionary acts, and b) that every illocutionary act is assumed to belong to one and only one category.

- *commissives*: in which the speaker commits to a future course of action (e.g. to promise, to offer),
- *expressives*: in which the speaker expresses a psychological condition about a state of affairs (e.g. to praise, to apologize), and
- *declarations*: in which the speaker—given his invested authority—states a proposition as a fact in the world (e.g. to marry (by a minister), to sentence (by a judge)).

2.3.1 *Speech Acts and ACL*

Software agents are problem solver computer programs that communicate with one another to elicit collaboration toward pursuing actions that they cannot accomplish individually. In this view, agents are autonomous goal-oriented entities with intentional communications, that is, they exchange messages that are designed to achieve certain ends, just as any other action that they may pursue to achieve their goals.

Since speech act theory was proposed as a model to analyze intentional components of individual natural language utterances, it was not unexpected that agent researchers would adopt it as a model for their ACL message semantics.

2.4 Speech Act Semantics in ACL

Currently there are five dominating approaches in the landscape of semantics for ACL. These are: Cohen and Levesque's Joint Intention Theory, FIPA ACL, KQML, Singh's social semantics for ACL, and Colombetti's Albatross. Based on their semantic principles, these ACL can be classified as based on either mental or social attributes.

2.4.1 *Mental Semantics*

Prevailing ACL specify their speech act semantics in terms of mental attributes, such as beliefs, intentions and goals. These include Cohen and Levesque's Joint Intention Theory, FIPA ACL and KQML. These are briefly described below.

Cohen And Levesque's Joint Intention Theory

Cohen and Levesque's Joint Intention Theory (JIT) (Cohen & Levesque, 1990a; Cohen & Levesque, 1990b; Kumar, et al., 2000) is a theory of rational action in which communicational acts are specified as attempts.

The communicational act *ATTEMPT* is defined as an action that an agent performs to achieve a goal *p* (which is currently believed to be false) while at least trying to achieve an intended effect *q* in case *p* cannot occur. That is, *p* represents a goal that may or may not be achieved by the attempt, and *q* represents what it takes to make an honest effort.

ATTEMPT is the basic component from which other communicative acts, such as *INFORM*, are derived. Therefore, *INFORM* is specified as an *ATTEMPT* where *p* is a goal in which a hearer is to believe that the speaker and herself (the hearer) mutually believe in *p*; and where the intended effect *q* is to create in the hearer the belief that both the speaker and hearer mutually believe that the speaker believes in *p*.

FIPA ACL

The Foundation for Intelligent Physical Agents (FIPA, 1997) proposes an ACL (FIPA-ACL) in which speech acts are defined based on the rational conditions preceding and succeeding their utterance. For example, the communicative act *inform* is defined as an illocution whose feasibility precondition is that the speaker both believes in a proposition *p*, and does not believe that the hearer has any belief or uncertainty about this proposition, and whose rational effect (the post-condition) is that the hearer comes to believe *p*.

KQML

Similar to FIPA-ACL, communicative acts in the Knowledge Query and Manipulation Language (KQML) (Finin, et al., 1997; Labrou, 1997; Labrou & Finin, 1999) also specifies the pre- and post-conditions of a speech act utterance. These pre- and post-conditions can be illustrated by the communicative act *tell*, which is defined as an illocution whose

- pre-conditions are that the speaker believes in a proposition *p*, that he knows that the hearer wants to either believe or not believe *p*, and that the hearer intends (i.e., is committed) to know this proposition; and
- post-conditions are that the speaker knows that the hearer knows that the speaker believes in *p*; and that the hearer knows that the speaker believes in *p*.

2.4.2 Social Semantics

An alternative approach to speech act semantics is to define the meaning of messages using social commitments. Two approaches to social semantics are described below; these are the social semantic models proposed by Singh and Colombetti.

Singh's Social Semantics

Singh (1999) proposes a semantic model to formalize speech acts in terms of their objective, subjective and practical validity claims (Habermas, 1984).

In this model, *objective validity claims* is the meaning that is given as the intrinsic value of an illocution. For example, that agent *x* is informing agent *y* of a proposition *p* indicates a commitment between *x* and *y* on the validity of *p*. *Subjective validity claims* signal the meaning of an utterance in cases when the speaker is deemed sincere. For example, that *x* is informing *y* of a proposition *p*, and *x* is considered to be truthful, indicates a commitment between *x* and *y* in which *x* beliefs *p* to be true. Lastly, *practical validity claims* are communicational meta-commitments that justify speakers (upon a context *G*) to utter a communicative act. Thus, that *x* is informing *y* is justified by the commitment of *x* to *G* that he has reasons to know the value of *p*.

Colombetti's Albatross

Colombetti (2000) proposes an agent language (called *Albatross*) in which speech acts eliciting actions bind agents to *pre-commitments*, that is, weaker states of commitment that do not obligate to action. Once pre-commitments are established they can be confirmed as commitments through the uttering of additional speech acts. This uptake of commitments is achieved through the speech acts *request* (a directive) and *accept* (an assertive).

A *request* is a speech act that is uttered as an act *e* by agent *x* in which he communicates to agent *y* the proposition φ that she does action α before time *d* has elapsed (where *d* is in the future). By uttering this request, *x* is not imposing a commitment on *y* but rather is negotiating this commitment through a pre-commitment. This pre-commitment does not yet bind *y* to do action α since it has to be changed to a commitment in order to create an obligation. An *accept* is a speech act that does exactly that, that is, it creates a commitment based on the existence of a pre-commitment.

2.5 Conversational Sequencing in ACL

Agents can be analysed as entities that use some sort of utility function to optimize their behaviour. In this view, agents select the best action to pursue next by taking into account the variables that affect this function (e.g., current goals, available resources). This also applies to conversations, and agents are expected to choose the utterances they believe constitute an optimal participation at any given time.

The problem with conversations, as with any other joint action, is that participants need to coordinate their intentions in order to advance them. To support agents' autonomy, such coordination cannot be achieved by having agents directly accessing each other's internal states. Instead they must infer other agents' intentions from their public communicative behaviour. There are two approaches that agents can apply to coordinate their intentions in conversations. These are *conversation protocols* and *conversation policies*.

2.5.1 Conversation Protocols

Conversation protocols are static structures that deterministically specify which messages can follow any given message in a conversation.

Protocols are usually represented with state-transition diagrams (e.g., Winograd & Flores, 1987; Bradshaw, et al., 1997) or Petri Nets (e.g., Ferber, 1999; Cost, et al., 1999).

Although they are simple to implement, protocols lack any compositional rules to define how they can be extended or merged (Greaves, et al., 1999). That is, once a protocol is specified, any modification to the types or sequencing of messages results in a completely new protocol that needs to be provided to all interacting agents.

2.5.2 Conversation Policies

On the other hand, conversation policies are defined as declarative rules that constrain the nature and exchange of speech acts between agents. These rules correspond to assumptions limiting the scope of utterances that agents should consider when selecting their participations in a conversation.

Greaves, et al., (1999) has proposed several requirements for conversation policies. These are: a) they must be independent from specific implementation techniques; b) they must be flexible enough to allow dynamic context-dependent composition; and finally, c) they must support conversations between agents of different levels of sophistication.

2.5.3 Conversations in Current ACL

This section briefly describes the approaches taken by surveyed ACL with respect to the ordering of utterances in a conversation.

Smith et al. (1998) proposed an extension to Cohen and Levesque's JIT in which speech acts compositionally specify the speech acts preceding and following their utterance. In this view, speech act sequencing is pre-defined to those speech acts (and their subtypes) in a specification. That is the case of the communicative act *ACKNOWLEDGE*, for example, which is defined as an *INFORM* in which the speaker expresses that he believes that the hearer believes a proposition *p* given that the hearer previously uttered an *INFORM* informing his belief in *p* (in other words, the utterance of an *INFORM* informing *p* can be followed by an *ACKNOWLEDGE* informing that the informed agent believes that the informer agent believes in *p*).

Similarly, KQML supports a limited approach to conversational sequencing based on the causal relation between the mental states in the pre- and post-conditions of speech acts.

Although this approach can be viable for simple conversations, it has been complemented by protocols composed in a definite clause grammar. In contrast, FIPA ACL and Albatross do not make any attempt to define rules of conversational sequencing, and rely on protocols to govern conversations.

Lastly, Singh's social semantics model is currently being enhanced with a framework for conversations called *commitment machines* (Yolum & Singh, 2001). This framework specifies conversations as sequences of states and transitions that affect the commitments of interacting agents. The operational semantics of interactions is captured by reasoning rules indicating the manipulation of commitments.

2.5.4 Requirements for a Model for Conversations

Prior to embracing new endeavours (such as developing a new model for conversations), one needs to analyse existing techniques in order to avoid reinventing what already exists. To that end, this section presents the requirements for a model for conversations and evaluates whether or not these requirements are satisfied by existing ACL.

Speech Act Semantics

One of the premises of speech act theory is that utterances are events that change the mental states of interacting agents based on their ability to recognize one another's intentions (Cohen, Morgan and Pollack, 1990; Perrault, 1990).[7] This implies that hearers are bound to infer speaker's intent solely by the reasons they may think of as to why the speaker selected a specific utterance at that particular occasion (and where the hearers also know that the speaker selected this utterance knowing that hearers will engage in such a deliberation) (Sadock, 1990). What it is important here is the notion that utterances do not readily

[7] As asserted by Bratman (1990), the main reason for agents to communicate their intentions is to coordinate their actions. Bratman eloquently argues that intentions are conduct controllers whose communication creates social expectations of behaviour. Particularly, he contends that intentions are a stable mental notion (in contrast to desires, which he sees as mere potential influencers of conduct) that agents could rely on to persist until the time of action, a characteristic that makes them adequate for agents to rely on to coordinate their actions.

embody speaker's intent, but that utterances are formed in such a way as to allow hearers to find reasons to derive such intent. That is, utterances are actions that have a meaning that is independent of (but not unrelated to) the speaker's mental states. The challenge faced by speakers and hearers is to coordinate their expectations as to convey in an utterance (in the case of speakers) and to recognize in an utterance (in the case of hearers) the same mental states.

From this perspective, current ACL based on mental semantics (i.e., JIT, FIPA ACL, KQML, and (to a certain extent) Singh's model) attempt to bridge the communication of intent by directly defining utterances in terms of the mental states of agents. Although these definitions could result in the simplification of the inference process bore by hearers, it makes it so by imposing that speakers use an utterance only when their mental states match its definition, which demand that agents are always *sincere*. As has been contended in the past by various researchers (e.g., Singh (1998), Wooldridge (1998), FIPA (1997)), sincerity in communications is an unrealistic requirement for autonomous agents: confirming that an utterance complies with its definition (or proving otherwise) requires agents to inspect each other's mental states, which could only be done if agents are implemented in pre-established ways (thus restricting the autonomy on their design and construction, i.e., their heterogeneity). Because of their use of mental states to define utterances, ACL based on mental semantics are said to be not publicly verifiable (i.e., the occurrence of an utterance cannot be confirmed to match its definition). In contrast, this thesis contends that utterances *do* have a public meaning that is used by speakers to hint at their intentions, and by hearers to infer the intentions of speakers.

This leads to the *first requirement* for a model for conversations:

To support agent conversations in open environments, a model for conversations must specify its message semantics using publicly verifiable principles.

Rather than directly linking utterances to the mental states of interacting agents, this thesis adopts the stance that inferring speakers' intents is a gradual process involving contractual social aspects (Winograd & Flores, 1987). In particular, that the notion of social commitments can provide for a principled way to connect the external (public) world of interactions with the internal (private) world of individual rational action. As such, the notion of social commitments will be used in this thesis to define a model for conversations.

Conversational Sequencing

The question of what it is that makes a conversation meaningful has puzzled researchers for some time now. As defined by Craig and Tracy (1983), a coherent conversation is a sequence of utterances exchanged by competent agents that seem to be connected in orderly and meaningful ways. They argue that conversations could be analysed from the points of view of *form* and *strategy*, where the former refers to the conversational rules and patterns for structuring conversations (e.g., sequencing and turn-taking), and the latter refers to the agents' deliberate selection of utterances that help them accomplish their individual goals.

On the one hand, form and strategy are complementary, that is, rational strategies can use structural rules as a resource for accomplishing an agent's conversational goals. On the other hand, their application is not symmetric: agents may have contrasting strategies (since each agent individually selects the strategy that they believe will lead to their goals) but they must have the same formation rules to structure their conversational exchanges.

Even though these two views are worth of investigation, this thesis is particularly interested in the public aspects of conversational structuring (rather than the private aspects of rational strategies). This public aspect of conversations is what has normally been addressed by conversation protocols (patterns) and conversation policies (rules). This thesis explores the same vein, but emphasises the use of conversational policies over protocols, since protocols can be seen as static subsets of the universe of conversations allowed by policies (which could potentially provide for more flexible conversations).

As such, the *second requirement* for a model for conversations is:

To support the form of conversations, a model for conversations must define policies governing conversational composition.

One of the characteristics noted in most surveyed ACL is their dependency on conversation protocols, which limit their support for flexible, context-dependent interoperability. It is worth noting that at least one of these approaches (Singh's *commitment machines*) investigates the principles governing conversational composition. The main similarities and differences between this approach and the model for conversations described in this thesis are later considered in Chapter 6.

2.6 Summary

This chapter briefly analysed current ACL approaches and their theoretical foundations. The ACL surveyed were Cohen and Levesque's Joint Intention Theory, KQML, FIPA-ACL, Singh's social semantics and Colombetti's Albatross.

The main motivation for investigating these ACL was to identify their adequacy to support agent conversations in open environments. This led to two requirements: first, that message semantics must be based on publicly verifiable principles (so that any utterance could be verified to comply with the language definitions); and second, that the sequencing of messages must be governed by flexible compositional policies (so that conversations could be dynamically composed to account for the context-dependent circumstances on which they occur). These requirements were shown not to be satisfactorily supported by the surveyed ACL. The next chapter will describe a model for agent conversations that does satisfy these requirements.

Chapter 3
Modelling Agent Conversations for Action

3.1 Overview

This chapter describes the basics notions in the model for conversations for action, whose fundamental principle is the negotiation of shared social commitments and the obligations these commitments entail.

The first section in this chapter presents the specification of actions and events (which are the occurrence of actions), speech acts (which are specified as actions) and utterances (which are modelled as occurrences of speech acts). Subsequent sections define social commitments, shared social commitments, and the operations that can be proposed to affect the state of shared social commitments of agents, namely their addition or discard as shared. This is followed by the definition of agents, which are modelled as entities with a record of utterances, shared social commitments, and obligations resulting from the adoption of these social commitments. The negotiation of shared social commitments is supported by a simple protocol implemented using illocutionary points and conversation policies (which together comprise the *Protocol for Proposals*). Also, this chapter presents a definition of normative societies, that is, societies where norms define the expected behaviour of agents when engaged in joint activities.

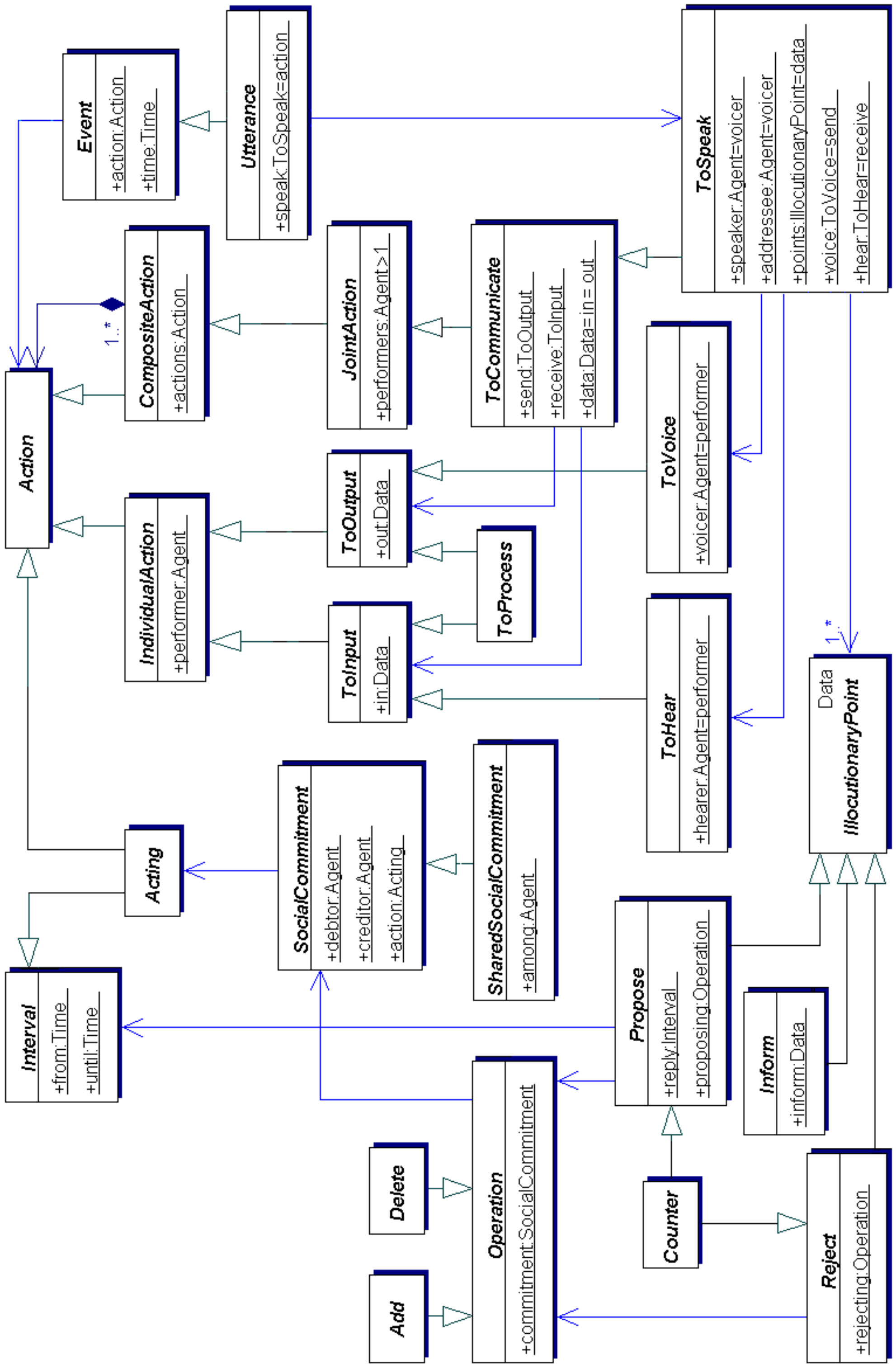


Figure 4. UML diagram of main classes in the model for conversations.

Figure 4 shows a simplified UML class diagram with most of these concepts. This diagram can be used as a road map when reading subsequent sections.

Lastly, this model is formalized using the Object-Z formal language notation.[8] Some of the advantages of this language are that it allows type checking of specifications, formal proof of certain properties of modelled systems, and a reasonably straightforward translation to computer implementations.[9]

3.2 Actions

Multi-agent systems are systems composed of loosely coupled agents that coordinate their actions to achieve their individual and collective objectives (Ferber, 1999). In these systems, the purpose of interacting (or conversing, in the case of purely communicational agents) is to bring about the execution of actions.

3.2.1 Individual and Composite Actions

Actions are defined as either individual or composite, where an *individual action* is an atomic action performed by one agent, and a *composite action* is a collection of actions performed by one or more agents.

This composition of actions is modelled following the guidelines of the *Composite* design pattern (Gamma, et al., 1995). As a result, the following three classes are modelled: the

[8] For readers not familiar with Z and Object-Z, Appendix A presents a brief tutorial on these specification languages. Further reading on these languages can be found in (Spivey, 1992) and (Diller, 1990), and (Smith, 2000), respectively. The type consistency of specifications in this thesis was verified using the Wizard type checker (SVRC, 2002).

[9] The main advantage of a formal specification is that it helps to unambiguously describe a state of affairs. Propositional, first-order and modal logics are examples of formal notations that can be used to this end. Object-Z is a specification language whose formal notation is derived from first-order logic, set theory and object-oriented principles. In the realms of Computer Science, it is not only desirable to have unequivocal specifications but it is usually required to also have a computational implementation. Under these circumstances Object-Z is a more appealing candidate than other formal notations to specify computational systems due to its use of object-oriented principles, which facilitate the implementation of systems in common object-oriented programming languages.

class *Action* (as the superclass component node), the class *IndividualAction* (as the leaf node, a subclass of *Action*), and the class *CompositeAction* (as the composite node, another subclass of *Action*).

The class *Action* contains the state variable *performers* (to reference a set of one or more objects of type—or subtype of—Agent), and the empty operation *Perform* (which is overridden by subclasses to specify what it means to do the action.

$Action$

$$performers : \mathbb{P}_1 \downarrow Agent$$

$$Perform \mathrel{\widehat{=}} [\,]$$

The class *IndividualAction* inherits from *Action* and defines the variable *performer* (to reference an instance of type—or subtype of—*Agent*), which is specified as the only agent in the set of performers (which effectively makes it the only performer of the action).

$IndividualAction$

$Action$

$$performer : \downarrow Agent$$

$$performers = \{performer\}$$

Lastly, the class *CompositeAction* is a class inheriting from *Action* that defines the variable *actions* (to reference a set of one or more instances of type—or subtype of—*Action*), and the operation *Perform*, which specifies that all actions in the *actions* set are performed concurrently. This class also specifies that the set of performers of this action is equal to the set of all performers of actions in the set *actions*.

$$\begin{array}{l}
\textbf{\textit{CompositeAction}} \\
\quad Action \\
\quad \begin{array}{|l}
actions : \mathbb{P}_1 \downarrow Action \\
\hline
performers = \{performer : \downarrow Agent \mid \\
\qquad \forall\, action : actions \\
\qquad \bullet \forall\, agent : action.performers \\
\qquad\quad \bullet\, agent = performer\}
\end{array} \\
\quad Perform \mathrel{\widehat{=}} \bigwedge action : actions \bullet action.Perform
\end{array}$$

Basic Individual Actions

There are three basic individual actions from which we derive more meaningful actions:

- *ToInput*: actions that receive an input,
- *ToOutput*: actions that generate an output, and
- *ToProcess*: actions that receive an input and generate an output.

As shown below, all inputs and outputs in these classes are sets of objects of type—or subtype of—*Data* (which is an empty class that acts as the superclass for all data classes).

$$\begin{array}{l}
\textbf{\textit{ToInput}} \\
\quad IndividualAction \\
\quad \begin{array}{|l}
data? : \mathbb{P} \downarrow Data
\end{array}
\end{array}$$

$$\begin{array}{l}
\textbf{\textit{ToOutput}} \\
\quad IndividualAction \\
\quad \begin{array}{|l}
data! : \mathbb{P} \downarrow Data
\end{array}
\end{array}$$

$$\begin{array}{l}
\textbf{\textit{ToProcess}} \\
\quad ToInput \\
\quad ToOutput
\end{array}$$

Joint Actions

Lastly, a *joint action* is a type of composite action in which there are two or more performers.[10] All joint actions in the model are derived from this class.

$JointAction$

$CompositeAction$

$\#performers \geq 2$

3.3 Communicational Actions

A *communicational action* is a joint action that involves communication between two agents. In this model the basic communicational action (from which other communicational actions are derived) is *ToCommunicate*. As shown below, this action specifies the state variables *sender* and *receiver* (both of type *Agent*), a variable *data* (to reference a non-empty set of objects of type *Data*), and two individual actions named *send* and *receive* (of type *ToOutput* and *ToInput*, respectively). This class also specifies that the *sender* is the performer of the *send* action; that the *receiver* is the performer of the *receive* action; that the value of the variable *data* equals both the output of the *send* action and the input of the *receive* action; and that the actions *send* and *receive* are the only actions in this joint action. Lastly, the operation *Perform* indicates that the performance of the *send* action precedes the performance of the *receive* action (which effectively makes the output of one action the input for the next).

[10] This definition is an oversimplification of joint actions as described by Clark (1996). Clark specifies that there exists two types of individual actions: *autonomous actions* (actions performed by one agent in isolation) and *participatory actions* (actions performed by one agent in coordination with the performances of other agents in a joint action). As such, joint actions have participatory actions as their base actions. In the case of this model, however, individual actions are treated as atomic and no differentiation is made between the two.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{ToCommunicate}} \\
\hline
\textit{JointAction} \\
\begin{array}{|l}
\hline
\textit{sender}, \textit{receiver} : \downarrow \textit{Agent} \\
\textit{data} : \mathbb{P}_1 \downarrow \textit{Data} \\
\textit{send} : \downarrow \textit{ToOutput} \\
\textit{receive} : \downarrow \textit{ToInput} \\
\hline
\textit{sender} = \textit{send}.\textit{performer} \land \\
\textit{receiver} = \textit{receive}.\textit{performer} \land \\
\textit{data} = \textit{send}.\textit{data}! = \textit{receive}.\textit{data}? \\
\textit{actions} = \{\textit{send}, \textit{receive}\} \\
\hline
\end{array} \\
\textit{Perform} \mathrel{\widehat=} \textit{send}.\textit{Perform} \fatsemi \textit{receive}.\textit{Perform} \\
\hline
\end{array}$$

Based on this definition, the action *Communicating* is defined as the union of the classes *ToCommunicate* and *CompositeActing*.

3.3.1 Speech Acts

According to the speech-as-action tradition, speech acts are composed of an *illocutionary point* (conveying the intent of a speech act), an *illocutionary force* (expressing the strength) and a *propositional content* (carrying the information or data).

In this model, the illocutionary point is the main carrier of meaning (rather than the illocutionary force, which is not modelled) [11]. As such, speaking is a joint action in which a speaker communicates illocutionary points to an addressee (where an illocutionary point is a subtype of *Data*). As shown below, the class of *ToSpeak* is a subclass of *ToCommunicate* that defines the state variables *speaker* and *addressee* (to indicate the sender and receiver agents), *points* (to indicate a non-empty set of illocutionary points that is communicated),

[11] To illustrate the difference between the illocutionary point and illocutionary force in a speech act, imagine the utterances “Could you please do your homework now?” and “You must do your home work right now!” These utterances have the same illocutionary point (that of the addressee doing his/her homework now) but different illocutionary force (the former being a polite request and the latter a forceful order). Nevertheless, that the illocutionary force is not currently part of the model for conversations does not mean that its use may not be desirable or advantageous (e.g., when requests are to be communicated with a sense of urgency). Although the model could well accommodate such extensions, their realization is left as an exercise for future research.

and the variables *voice* and *hear* (to indicate the actions for sending and receiving the communicated illocutionary points, respectively). The types *ToVoice* and *ToHear* (from which the variables *voice* and *hear* are instantiated) are defined as subclasses of *ToSend* and *ToInput*, respectively.

$$
\begin{array}{|l}
\hline
\textit{ToSpeak} \\
\textit{ToCommunicate} \\
\begin{array}{|l}
\hline
speaker, addressee : \downarrow Agent \\
points : \mathbb{P}_1 \downarrow IllocutionaryPoint \\
voice : \downarrow ToVoice \\
hear : \downarrow ToHear \\
\hline
speaker = sender \land \\
addressee = receiver \land \\
points = data \\
voice = send \land \\
hear = receive \\
\hline
\end{array} \\
\hline
\end{array}
$$

Lastly, we define an illocutionary point that conveys information (i.e., propositional content) between a speaker and addressee. As shown below, the *Inform* illocutionary point defines a variable *informing* that references a set of instances of type—or subtype of—*Data*. In section *3.8*:*Illocutionary Points* we define other illocutionary points that are used for the negotiation of shared social commitments.

$$
\begin{array}{|l}
\hline
\textit{Inform} \\
\textit{IllocutionaryPoint} \\
\begin{array}{|l}
\hline
informing : \mathbb{P} \downarrow Data \\
\hline
\end{array} \\
\hline
\end{array}
$$

Based on this definition, the action *Speaking* is defined as the union of the classes *ToSpeak* and *CompositeActing*.

3.3.2 Other Basic Joint Actions

We define two basic joint action classes from which other application-dependent actions are derived. These joint actions are *ToGenerate* and *ToProduce*. As shown below, the class *ToGenerate* specifies that a *generator* agent first performs a *ToOutput* action that produces some output data (*out*). Then it performs a *ToSpeak* action to inform a *receiver* agent of the output data.

$$
\begin{array}{l}
\textit{ToGenerate} \\
\hline
\textit{JointAction} \\
\begin{array}{|l}
\hline
\textit{generator}, \textit{receiver} : \downarrow \textit{Agent} \\
\textit{out} : \mathbb{P} \downarrow \textit{Data} \\
\textit{output} : \downarrow \textit{ToOutput} \\
\textit{speak} : \downarrow \textit{ToSpeak} \\
\hline
\textit{generator} = \textit{output}.\textit{performer} = \textit{speak}.\textit{speaker} \land \\
\textit{receiver} = \textit{speak}.\textit{addressee} \land \\
\textit{out} = \textit{output}.\textit{data}! \land \\
\textit{actions} = \{\textit{output}, \textit{speak}\} \\
\exists\, i : \textit{Inform} \mid \\
\quad i \in \textit{speak}.\textit{points} \\
\bullet\ i.\textit{informing} = \textit{out} \\
\hline
\end{array} \\
\textit{Perform} \mathrel{\widehat{=}} \textit{output}.\textit{Perform} \mathbin{;} \textit{speak}.\textit{Perform} \\
\hline
\end{array}
$$

The class *ToProduce* is a subclass of *ToGenerate* in which the *output* action is an instance of *ToProcess*, that is, an individual action that generates an output given an input.[12] Based on the behaviour inherited from *ToGenerate*, the output of the action is communicated to the receiver through an *Inform* illocutionary point.

[12] This class also defines the variable *producer* to reference the inherited *generator* agent. The only reason for this redundancy is to facilitate readability, that is, a *producer* performs a *producing* action (c.f., *generator*).

ToProduce

ToGenerate

$producer : \downarrow Agent$
$in : \mathbb{P} \downarrow Data$
$process : \downarrow ToProcess$

$producer = generator \wedge$
$process = output \wedge$
$in = process.data?$

3.3.3 Events and Utterances

Actions in this model are abstract concepts that have not occurred in the environment. When they do happen they are considered events. Therefore, an event is the occurrence of an action at a certain moment in time. As shown below, *Event* is a class specifying the state variables *time* (to indicate the time of occurrence of the event) and *action* (to indicate the action that occurred).[13]

Event

$time : Time$
$action : \downarrow Action$

Just as an event is defined as the record of the occurrence of any type of action, an utterance is an event specifically involving the illocution of a speech act. Therefore, *Utterance* is a class inheriting from *Event* whose variable *action* is restricted to an instance of type *ToSpeak*.

Utterance

Event

$speechAct : ToSpeak$

$speechAct = action$

[13] In our model, the type *Time* is simply defined as a natural number (i.e., *Time* = $\mathbb{N}$).

3.3.4 Scheduled Actions

When actions are negotiated they ought to refer to a time interval indicating when their performance is expected to occur. We define the class *Interval* to indicate a time period. This class defines the state variables *from* and *until* to denote the lower and upper bound of the interval.

$Interval$

$from, until : Time$

$from \leq until$

Using this definition, the class *Acting* is specified as a subclass of *Action* that defines a variable *time* of type *Interval*. Interval denotes the time period within which the action is to be performed.[14] This class specifies an empty operation *ProcessNorm* that is overridden by subclasses and which is used at the time that the action is evaluated by norms in a society (as explained in section *3.9: Conversation Policies*).

$Acting$

$Action$

$time : Interval$

$ProcessNorm \mathrel{\widehat{=}} [\,]$

The class *IndividualActing* is a subclass of *IndividualAction* and *Acting* that overrides the operation *ProcessNorm* to invoke the operation *ProcessAction* of a given norm, which is an instance of type *ActionNorm* (this class is also explained is section *3.9: Conversation Policies*).

[14] This thesis follows the convention of naming actions either in their infinitive form (e.g., *ToSpeak*) in the case of abstract actions, or in their present participle form (e.g., *Speaking*) in the case of scheduled actions (i.e., actions that have an expected interval time of occurrence).

$$\begin{array}{|l}
\hline
\textit{IndividualActing} \\
\textit{IndividualAction} \\
\textit{Acting} \\
\textit{ProcessNorm} \mathrel{\widehat=} [\, \textit{norm?} : \downarrow \textit{ActionNorm} \,] \bullet \textit{norm?.ProcessAction} \\
\hline
\end{array}$$

Likewise, the class *CompositeActing* is a subclass of *IndividualAction* and *Acting* that overrides the operation *ProcessNorm* not only to invoke the operation *ProcessAction* but also to invoke the operation *ProcessNorm* for each enclosed action. Also, this class constraints all enclosed actions to be of type—or subtype of—*Acting*, and that the time period in which these actions are expected to occur is within the time of occurrence specified for the composite action.

$$\begin{array}{|l}
\hline
\textit{CompositeActing} \\
\textit{CompositeAction} \\
\textit{Acting} \\
\begin{array}{|l}
\hline
\forall\, \textit{action} : \textit{actions} \\
\bullet\, \exists\, a : \downarrow \textit{Acting} \mid \\
\qquad a = \textit{action} \\
\quad \bullet\, a.\textit{time}.\textit{from} \geq \textit{time}.\textit{from} \land \\
\qquad a.\textit{time}.\textit{until} \leq \textit{time}.\textit{until} \\
\hline
\end{array} \\
\textit{ProcessNorm} \mathrel{\widehat=} [\, \textit{norm?} : \downarrow \textit{ActionNorm} \,] \bullet \\
\qquad\qquad \bigwedge \textit{action} : \textit{actions} \bullet \textit{norm?.ProcessAction} \land \textit{action.ProcessNorm} \\
\hline
\end{array}$$

3.4 Social Commitments

Social commitments are directed obligations in which one agent (called the *debtor* of the commitment) has a responsibility relative to another agent (the *creditor* of the commitment) for the performance of an action (Singh, et al., 1999).[15] Accordingly, the class *SocialCommitment* specifies the state variables *debtor*, *creditor* and *action*.[16]

---

[15] This type of commitment is also known as *relational commitment* (Ferber, 1999).

[16] Strictly speaking, the agent responsible for the action does not need to be the one performing this action. It can be inferred that agents that become responsible for an action that is performed by another agent should

SocialCommitment

$debtor, creditor : \downarrow Agent$
$action : \downarrow Acting$

3.4.1 Shared Social Commitments

Once a social commitment has been proposed and agreed upon by negotiating agents it acquires the status of being shared. The class *SharedSocialCommitment* represents this type of social commitment. This class inherits from *SocialCommitment* and defines the variable *among* to indicate the set of agents among which this commitment is shared.[17]

SharedSocialCommitment

SocialCommitment

$among : \mathbb{P} \downarrow Agent$

$\# among \geq 2$

3.4.2 Operations

The proposal and later acceptance of a social commitment can have two possible outcomes: either the commitment is adopted as shared, or the commitment is discarded as not shared any longer. Therefore, two operations are defined which agents can use when putting forth commitments for negotiation: adding and deleting. First, we define the class *Operation* as the superclass for these two operations. This class defines a single state variable *commitment* to reference an instance of type *SocialCommitment*.

Operation

$commitment : SocialCommitment$

have a degree of authority over this agent so that the action is accomplished. Nevertheless, this dimension of social commitments involving social organizations is not addressed in this thesis.

[17] Although strictly speaking a social commitment could be shared by any number of agents (as long as it is more than one) our model currently deals only with the commitments shared between a speaker and an addressee.

The classes *Add* and *Delete* are then defined as inheriting from the class *Operation* (the usage of these operations is shown later in section *3.7: Negotiating Shared Social Commitments*).

$$\begin{array}{|l}\hline \mathit{Add} \\ \mathit{Operation} \\ \hline \end{array}$$

$$\begin{array}{|l}\hline \mathit{Delete} \\ \mathit{Operation} \\ \hline \end{array}$$

3.5 Obligations

Obligations are engagements that bind an agent to a course of action.[18] In this model, obligations are created (and discarded) by the adoption (and discharge) of shared social commitments. For example, that Bob has adopted a commitment in which he is to tell Alice the time creates the obligations in Bob that he has to find out the time and that he has to tell the time to Alice. Obligations are defined as actions that are expected to occur (i.e., as any action of type *Acting*).

$$\mathit{Obligation} == \downarrow \mathit{Acting}$$

3.6 Agents

Agents are conceptualized as entities that keep a collection of shared social commitments and obligations, and a history of the utterances in which they have participated.

As shown in Figure 5, the class *Agent* defines the variables *commitments* (which holds the shared social commitments of the agent), *obligations* (which holds the social obligations of

[18] This definition of obligation is akin to the notion of *simple action-commitment statements* specified by D.N. Walton and E.C.W. Krabbe (1995).

the agent),[19] *inbox* and *history* (which are the sequence of utterances that the agent has just received but has not processed yet, and the sequence of utterances that have already been processed, respectively), and the variable *current* (which specifies the utterance that is currently being processed).[20]

The remaining variables *com_add*, *com_delete*, *obl_add* and *obl_delete* are auxiliary variables that function as temporary containers to hold the commitments and obligations that are accumulated by the various norms when processing the utterance in *current*. As will be seen shortly, norms do not modify the state of commitments and obligations of an agent while each of the norms are being evaluated but rather all modifications are pooled in the above auxiliary variables for their later application once the utterance has been processed by all norms. The benefit of this technique is that it maintains the consistency of an agent state regardless of the order in which norms process an utterance, and the order in which the illocutionary points within the utterance are processed by a norm.

[19] There is a compelling reason to define the collections of shared commitments and obligation as bags (which allow duplicate elements) rather than sets (which do not allow duplicates). In our model, that an agent holds identical obligations only means that she has recorded those entries given independent interactions, not necessarily that the agent will perform the involved actions as many times as recorded. That is, the possible optimization of performances (i.e., whether one performance satisfies all obligations or if independent performances are required) is bound by the nature of the involved actions. For example, that Alice has committed to Bob to brush her teeth after dinner and that she has also committed to Charles to brush her teeth after dinner would generate two identical entries in Alice's record of obligations. No other details withstanding, Alice should be able to perform the action once to satisfy both Bob and Charles. On the other hand, there are actions that would not allow such optimizations. That would be the case of Alice and Bob adopting a shared commitment in which Bob bakes a cake for Alice tomorrow. If later that day Alice and Bob adopt another commitment in which Bob bakes a cake (one with identical characteristics as that of the previously requested cake) for Alice by the same time tomorrow, then Alice and Bob will hold a pair of identical commitments and obligations that should lead Bob to prepare two identical cakes for Alice.

[20] The meaning of "processing an utterance" will become clearer in later sections on norms and societies. In brief, when an agent receives an utterance this utterance is stored in the *inbox* sequence until its turn come to be processed by the norms that govern the behaviour of the agent (at which point the utterance becomes *current*, that is, the utterance currently being processed). Once this utterance has been processed, it becomes part of the history of utterances of the agent (and thus it is appended to the *history* sequence). It is worth noting that the model for conversations allows one agent to hold simultaneous conversations. In such cases, the utterances from these conversations accumulate in the sequence *inbox* in the order they arrive. In cases where some conversations may have higher priority than others, it is possible (and sometimes desirable) to extend the functionality of the agent specification to allow selecting the next utterance to process according to their perceived importance (rather than FIFO, as it is currently specified). Although the model for conversations has room for such extensions, they are not part of the specification presented in this thesis.

Agent

$$\begin{array}{l}
commitments, com\_add, com\_delete : \mathrm{bag}\ SharedSocialCommitment \\
obligations, obl\_add, obl\_delete : \mathrm{bag}\ Obligation \\
inbox, history : \mathrm{seq}\ Utterance \\
current : Utterance \\
\hline
\forall\, c : \mathrm{dom}\ commitments \\
\bullet\ self \in c.among \\
\forall\, o : \mathrm{dom}\ obligations \\
\bullet\ self \in o.performers \\
\forall\, u : \mathrm{ran}\,(inbox \frown \langle current \rangle \frown history) \\
\bullet\ u.speechAct.speaker = self \lor u.speechAct.addressee = self \\
\forall\, u_1, u_2 : Utterance \mid \langle u_1, u_2 \rangle\ \mathrm{in}\ inbox \\
\bullet\ u_1.time > u_2.time
\end{array}$$

INIT

$$\begin{array}{l}
commitments = \varnothing \\
obligations = \varnothing \\
inbox = \langle\,\rangle \\
history = \langle\,\rangle
\end{array}$$

AddUtterance

$$\begin{array}{l}
\Delta(inbox) \\
u? : Utterance \\
\hline
inbox' = inbox \frown \langle u? \rangle
\end{array}$$

Figure 5. Definition of the class *Agent* (part 1 of 3).

In addition, several restrictions are set on the following variables:

- *commitments* only holds shared social commitments in which the agent is one of the agents among which this commitment is shared,
- *obligations* only holds obligations in which the agent is one of the performers of the action,

$$\begin{array}{|l} \textit{SetCurrent} \\ \hline \Delta(inbox, current, com\_add, com\_delete, obl\_add, obl\_delete) \\ \hline current' = head\ inbox \\ inbox' = tail\ inbox \\ com\_add' = \varnothing \\ com\_delete' = \varnothing \\ obl\_add' = \varnothing \\ obl\_delete' = \varnothing \\ \hline \end{array}$$

$$ProcessCurrent \mathrel{\widehat=} [\, norms? : \mathbb{P} \downarrow Norm \,] \bullet \bigwedge n : norms? \bullet n.ProcessUtterance$$

$$\begin{array}{|l} \textit{AdvanceCurrent} \\ \hline \Delta(current, history) \\ \hline history' = \langle current \rangle \frown history \\ commitments' = (commitments \setminus com\_delete) \uplus com\_add \\ obligations' = (obligations \setminus obl\_delete) \uplus obl\_add \\ \hline \end{array}$$

$$Process \mathrel{\widehat=} SetCurrent \fatsemi ProcessCurrent \fatsemi AdvanceCurrent$$

Figure 6. Definition of the class *Agent* (part 2 of 3).

- *inbox* (whose utterances are ordered by time of occurrence), *current* and *history* only hold utterances in which the agent is either the speaker or the addressee.[21]

The interface of this class is defined by the following operations.

Figure 5 shows the operation schemas *INIT*, which initializes the variables *commitments*, *obligations*, *inbox* and *history* to empty; and *AddUtterance*, which appends an utterance to the *inbox* sequence.

[21] This restriction reflects the fact that the *Agent* class is designed to model the behavioural representation that an observer agent holds of other agents in the environment. In practice this means that only when an agent (Alice) has perceived that an utterance has occurred between two agents (Bob and Charles), she is justified in updating the representation she maintains of these two agents. Now imagine that there was another agent (Dave) in the proximity, who may have also perceived the utterance between Bob and Charles. Would Alice be justified in updating her representation of Dave to reflect that he has witnessed the communication? From a behavioural perspective, the answer is no. Basically, Alice can only know that Dave has also witnessed the utterance if Dave makes his awareness public (e.g. by uttering that he knows about it), in which case Alice would be justified in updating her and Dave's representations because there was an utterance involved. Otherwise Alice would need to reason whether or not there are reasons that justify concluding that Dave knows about the utterance—which clearly is a rational process that lies outside of the scope of the model.

$$
\begin{array}{|l}
\textit{AddCommitments} \\
\hline
\Delta(com\_add) \\
c? : \mathrm{bag}\ SharedSocialCommitment \\
\hline
com\_add' = com\_add \uplus c? \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{DeleteCommitments} \\
\hline
\Delta(com\_delete) \\
c? : \mathrm{bag}\ SharedSocialCommitment \\
\hline
com\_delete' = com\_delete \uplus c? \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{AddObligations} \\
\hline
\Delta(obl\_add) \\
o? : \mathrm{bag}\ Obligation \\
\hline
obl\_add' = obl\_add \uplus o? \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{DeleteObligations} \\
\hline
\Delta(obl\_delete) \\
o? : \mathrm{bag}\ Obligation \\
\hline
obl\_delete' = obl\_delete \uplus o? \\
\hline
\end{array}
$$

$$
\begin{array}{l}
SendUtterance \mathrel{\widehat=} [\, speechAct? : ToSpeak;\ u : Utterance \mid \\
\qquad speechAct?.speaker = self \land \\
\qquad u = (\mu\, new : Utterance \mid \\
\qquad\qquad new.time = now \land \\
\qquad\qquad new.speechAct = speechAct?) \,] \bullet \\
\quad \bigwedge a : \{speechAct?.speaker, speechAct?.addressee\} \bullet a.AddUtterance
\end{array}
$$

Figure 7. Definition of the class *Agent* (part 3 of 3).

Figure 6 shows the operations *SetCurrent*, which initializes as empty all auxiliary variables, and assigns the next utterance out of *inbox* as the next *current*; *ProcessCurrent*, which processes the utterance in *current* through all supplied norms; *AdvanceCurrent*, which adds *current* to the history of utterances, and updates the commitments and obligations of the agent with those commitments and obligations from the auxiliary variables; and *Process*, which sequentially invokes *SetCurrent*, *ProcessCurrent* and *AdvanceCurrent*.

Figure 7 shows the operation schemas *AddCommitments* and *DeleteCommitments*, which add shared social commitments to the *com_add* and *com_delete* variables, respectively; and *AddObligations* and *DeleteObligations*, which add obligations to the variables *obl_add* and *obl_delete*. Lastly, this class defines the operation *SendUtterance*, which specifies that the agent utters a given speech act to an addressee if and only if the agent is set as the speaker of the speech act.

3.7 Negotiating Shared Social Commitments

One way to support the autonomy of agents is to allow them to decide whether or not other agents can commit them to execute actions. In the case of this model, this means that shared social commitments are not imposed but rather are negotiated between interacting agents. To that end, we define a negotiation protocol that we call the *Protocol for Proposals* (PFP), which provides a flexible and unambiguous pattern of conversational turn-taking supporting the mutual adoption and discharge of social commitments to action. Briefly explained, an instance of the PFP starts with a proposal from a speaker to a hearer to adopt or discard a shared social commitment. Either the hearer replies with an acceptance, rejection or counterproposal, or the speaker issues a withdrawal (i.e., a rejection to one's own proposal) or a counterproposal.[22] All replies except a counterproposal terminate an instance of the protocol. A counterproposal is deemed a proposal in the sense that it can be followed by the same messages that can reply to a proposal (but with speaker-hearer roles inverted if the previously proposed agent is the speaker of the counterproposal).[23] Lastly, when an acceptance is issued, both speaker and

[22] It is also possible that the hearer goes silent. In such cases, the elapsing of the expected reply time indicates to the speaker (or any observer) that the hearer either intentionally forfeited his obligation to reply or was unable to communicate as expected. In such matters, we are assuming that communication between agents is reliable, that is, the transmission of utterances is always achieved (c.f. the *coordinated attack* problem (Fagin, et al., 1995)).

[23] In theory, a counterproposal can follow another counterproposal *ad infinitum*; in practice, however, successive counterproposals are limited by the reasoning, competence or endurance of interacting agents.

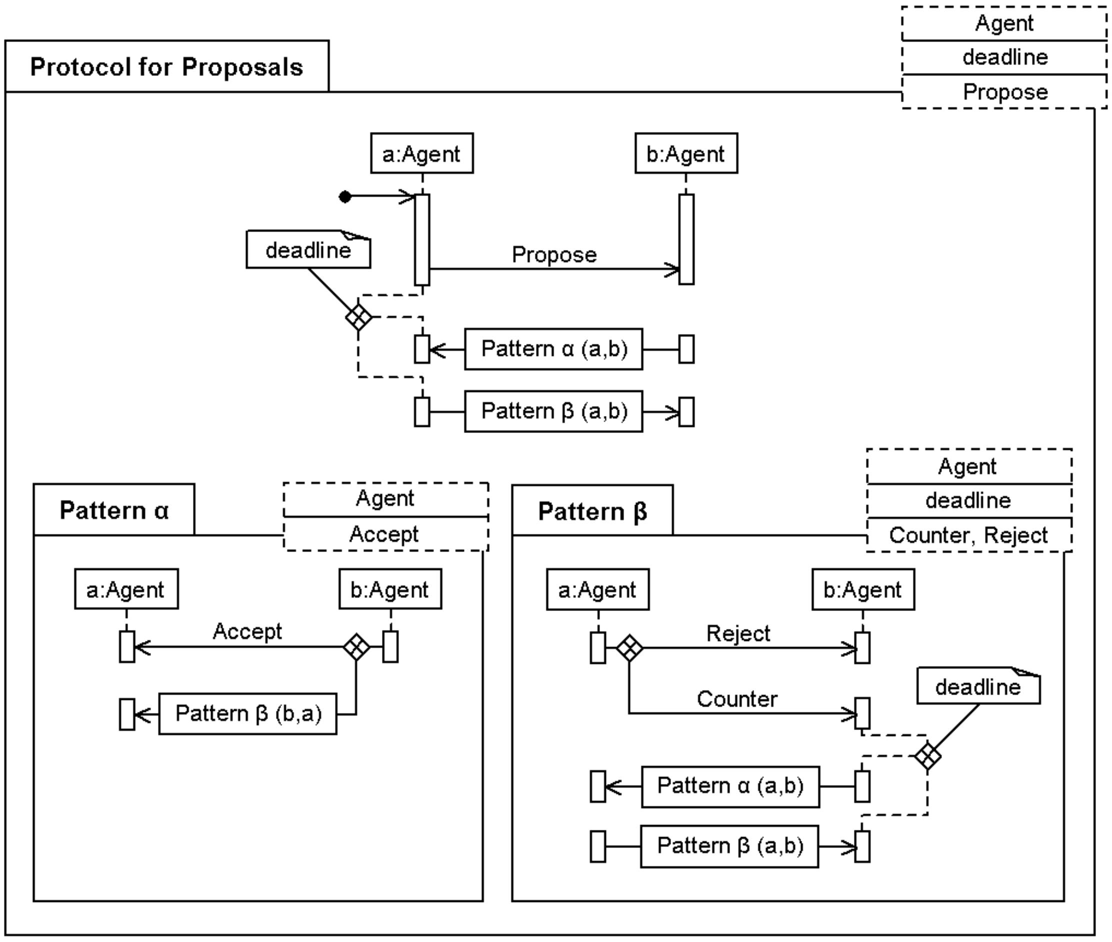


Figure 8. AUML interaction template for the *Protocol for Proposals*.

hearer simultaneously apply the proposed (and now accepted) operation on commitments to their record of shared social commitments and obligations.

Figure 8 shows the AUML (Odell, et al., 2001) interaction diagram for the PFP. As shown in the figure, the protocol starts with a proposal from agent *a* to agent *b*. This message can be followed (before the expiration of a deadline) by the interaction patterns α or β. The interaction pattern α indicates that either agent *b* sends an accepting message to agent *a*, or that the interaction follows pattern β (but with agents *a* and *b*'s participatory roles inverted, that is, the identity of the agent that in pattern α was agent *a* in pattern β will be agent *b*, and likewise for agent *b*). The interaction pattern β indicates that agent *a* sends a rejection or counterproposal message to agent *b*, in which case the interaction follows (before the expiration of a deadline) by either pattern α or pattern β.

As described next, the model for conversations implements the PFP using illocutionary points and conversation policies.

3.8 Illocutionary Points

This section defines the illocutionary points that are used to implement the PFP, namely *Propose*, *Accept*, *Reject* and *Counter* (for counter-proposals). These illocutionary points are defined as inheriting from the class *IllocutionaryPoint*).

3.8.1 Propose

The class *Propose* defines the state variables *proposing* (which indicates the operation on a social commitment being proposed) and *reply* (which indicates the time interval when a reply is expected).

Propose

$IllocutionaryPoint$

$proposing : \downarrow Operation$
$reply : Interval$

$proposing \in Add \land proposing \in Delete$

3.8.2 Accept

The class *Accept* defines the state variable *accepting* (which indicates the operation on a social commitment being accepted).

Accept

$IllocutionaryPoint$

$accepting : \downarrow Operation$

$accepting \in Add \land accepting \in Delete$

3.8.3 Reject

The class *Reject* defines the state variable *rejecting* (which indicates the operation on a social commitment being rejected).

$Reject$

$IllocutionaryPoint$

$rejecting : \downarrow Operation$

$rejecting \in Add \wedge rejecting \in Delete$

3.8.4 Counter

The class *Counter* is a class inheriting from *Reject* and *Propose*, where the former indicates a commitment previously proposed and now being rejected, and the latter indicates a newly proposed commitment (and corresponding expected time of reply).

$Counter$

$Reject$

$Propose$

To summarize, this section presented the illocutionary points that compose the speech acts that agents can use to negotiate shared social commitments. It is worth mentioning that up to this point only the *structures* that are communicated have been defined, and nothing has been said about how these structures are actually *used* or what they can *accomplish* when used in conversations. That is the topic of the next section on conversation policies.

3.9 Conversation Policies

The model defines conversation policies as norms that agents are expected to follow during their conversations. As shown below, the class *Norm* (which is the superclass of all norms in this model) defines a sole abstract operation *ProcessUtterance*. This operation, which is invoked when evaluating the current utterance of an agent, is overridden by subclasses defining concrete normative behaviour.

$$\begin{array}{|l}\hline \mathit{Norm} \\ \mathit{ProcessUtterance} \mathrel{\widehat{=}} [\,] \\ \hline\end{array}$$

The class *ActionNorm* is a subclass of *Norm* that defines an abstract operation *ProcessAction*. This operation is invoked by norms processing an utterance. It is overridden by policies that generate obligations based on the actions of negotiated social commitments.

$$\begin{array}{|l}\hline \mathit{ActionNorm} \\ \mathit{Norm} \\ \mathit{ProcessAction} \mathrel{\widehat{=}} [\,] \\ \hline\end{array}$$

3.9.1 Policy 1: Adopting Obligations to Reply

The first policy in the model specifies that agents proposed or counterproposed have to reply.[24]

As shown in Figure 9, this policy is specified as the class *PFPpolicy1* that inherits from *Norm* and defines the operations *ProcessUtterance* (which overrides the operation inherited from *Norm*) and *ProcessProposal*.

The operation *ProcessUtterance* specifies that for each proposal in a given utterance it successively invokes the operations *ProcessProposal* and *AddObligations*, which result in the generation and addition of obligations to a given agent instance.

The operation *ProcessProposal* is specified as receiving a proposal and returning a set of obligations to reply. In the case that the agent given as input is the speaker of the proposal, this operation returns obligations to a speaking action in which the agent is to hear the reply. In the case that the agent is the addressee, the operation returns obligations to the

[24] Although this is a strong assumption for autonomous agents, we see it as a rule of politeness: you answer if you are proposed. In any event, agents are still free to disregard this (or any) policy when they see it fit (for example if an insidious or defective agent sends inappropriate or hostile messages).

$$
\begin{array}{l}
\textit{PFPpolicy1} \\
\hline
\upharpoonleft(\textit{ProcessUtterance}) \\
\textit{Norm} \\
\quad \begin{array}{|l}
\textit{ProcessProposal} \\
\hline
\textit{agent?}, \textit{speaker?}, \textit{addressee?} : \downarrow\textit{Agent} \\
\textit{propose?} : \downarrow\textit{Propose} \\
\textit{result!} : \mathbb{P}\ \textit{Obligation} \\
\hline
\exists\, sa : \textit{Speaking} \mid \\
\quad sa.time = \textit{propose?}.reply \wedge \\
\quad sa.speaker = \textit{addressee?} \wedge \\
\quad sa.addressee = \textit{speaker?} \wedge \\
\quad sa.points = \textit{isReplyTo}(\textit{propose?}) \\
\bullet\ (\textit{agent?} = \textit{speaker?} \wedge \textit{result!} = \{sa, sa.hear\}) \vee \\
\quad (\textit{agent?} = \textit{addressee?} \wedge \textit{result!} = \{sa, sa.voice\})
\end{array} \\
\textit{ProcessUtterance} \mathrel{\widehat=} [\ \textit{agent?} : \downarrow\textit{Agent};\ \ u? : \textit{Utterance}\ ] \bullet \\
\qquad\qquad \bigwedge \textit{propose} : \downarrow\textit{Propose} \mid \\
\qquad\qquad\quad \textit{propose} \in u?.\textit{speechAct}.\textit{points} \\
\qquad\qquad \bullet\ \textit{ProcessProposal} \fatsemi \textit{agent?}.\textit{AddObligations}
\end{array}
$$

Figure 9. *Policy 1*: A proposal commits addressee agents to reply.

same speaking action but where the agent is to voice the reply. In addition, the reply's illocutionary points are specified by the axiom *isReplyTo* (which is defined below).

The axiom *isReplyTo* is a function that defines the set of illocutionary points that qualify as a reply to a given proposal. As shown below, this operation specifies that a proposal is replied to by a set of illocutionary points where there is either one illocutionary point accepting (and not one rejecting), or one illocutionary point rejecting (and not one accepting) the operation specified in the proposal.

$$\begin{array}{|l}
isReplyTo : \downarrow Propose \rightarrow \mathbb{P}_1 \downarrow IllocutionaryPoint \\
\hline
\forall\, propose : \downarrow Propose \\
\bullet\ \exists\, points : \mathbb{P} \downarrow IllocutionaryPoint \mid \\
\quad \exists\, a : Accept;\ r : \downarrow Reject \mid \\
\qquad a.accepting = propose.proposing \land \\
\qquad r.rejecting = propose.proposing \\
\quad \bullet\ (a \in points \land r \notin points) \lor \\
\qquad (r \in points \land a \notin points) \\
\quad \bullet\ points = isReplyTo(propose)
\end{array}$$

3.9.2 Policy 2: Discarding Obligations to Reply

Once an agent has adopted obligations to reply, she can expect these obligations to be discarded if any of the following two conditions occur:

1. The agent that was proposed to (or counterproposed to) utters a speech act containing an *Accept* illocutionary point with the same operation on commitment as that of the previously uttered *Propose* (or *Counter*).

2. The proposing or proposed to (or the counterproposing or counterproposed to) agent utters a speech act containing a *Reject* illocutionary point with the same operation on commitment as that of the previously uttered *Propose* (or *Counter*).

To support these conditions we define the class *PFPpolicy2* (shown in Figure 10 and Figure 11) which inherits from *Norm* and declares the operations *ProcessAcceptance*, *ProcessRejection* and *ProcessUtterance* (which overrides the abstract operation inherited from *Norm*).

The operation *ProcessUtterance* is specified as invoking the operations *ProcessAcceptance* and *ProcessRejection* for each acceptance and rejection (respectively) in a given utterance, and then invoking the operation *DeleteObligations*, which results in the generation and deletion of obligations from a given agent instance.

$$
\begin{array}{l}
\textit{PFPpolicy2} \\
\hline
\upharpoonleft(\textit{ProcessUtterance}) \\
\textit{Norm} \\
\quad \textit{ProcessAcceptance} \\
\quad \hline
\quad \textit{agent}?, \textit{speaker}?, \textit{addressee}? : \downarrow \textit{Agent} \\
\quad \textit{accept}? : \textit{Accept} \\
\quad \textit{time}? : \textit{Time} \\
\quad \textit{result}! : \mathbb{P}\ \textit{Obligation} \\
\quad \hline
\quad \exists\, \textit{setset} : \mathbb{P}(\mathbb{P}\ \textit{Obligation}) \mid \\
\quad\quad \textit{setset} = \{\textit{set} : \mathbb{P}\ \textit{Obligation} \mid \\
\quad\quad\quad \forall\, \textit{propose} : \textit{getProposeForAccept}(\textit{agent}?.\textit{history}, \\
\quad\quad\quad\quad\quad\quad\quad\quad \textit{speaker}?, \textit{addressee}?, \\
\quad\quad\quad\quad\quad\quad\quad\quad \textit{accept}?, \textit{time}?) \\
\quad\quad\quad \bullet \exists\, \textit{sa} : \textit{Speaking} \mid \\
\quad\quad\quad\quad \textit{sa}.\textit{time} = \textit{propose}.\textit{reply} \wedge \\
\quad\quad\quad\quad \textit{sa}.\textit{speaker} = \textit{speaker}? \wedge \\
\quad\quad\quad\quad \textit{sa}.\textit{addressee} = \textit{addressee}? \wedge \\
\quad\quad\quad\quad \textit{sa}.\textit{points} = \textit{isReplyTo}(\textit{propose}) \\
\quad\quad\quad\quad \bullet (\textit{agent}? = \textit{addressee}? \wedge \textit{set} = \{\textit{sa}, \textit{sa}.\textit{hear}\}) \vee \\
\quad\quad\quad\quad\quad (\textit{agent}? = \textit{speaker}? \wedge \textit{set} = \{\textit{sa}, \textit{sa}.\textit{voice}\})\} \\
\quad \bullet (\forall\, o : \textit{result}! \bullet \exists\, \textit{set} : \textit{setset} \bullet o \in \textit{set}) \wedge \\
\quad\quad (\forall\, o : \textit{Obligation} \mid \exists\, \textit{set} : \textit{setset} \bullet o \in \textit{set} \bullet o \in \textit{result}!)
\end{array}
$$

Figure 10. *Policy 2*: A reply releases agents of the obligation to reply (part 1 of 2).

The operation *ProcessAcceptance* is specified as receiving an acceptance and checking whether or not this acceptance has matching proposals in the history of past utterances of the agent. If such proposals exist then the history of utterances is analysed once more to find out whether or not these proposals have been replied to. If at least one of these proposals has not been replied to, then the acceptance is a legitimate reply, and thus the corresponding obligations are selected for discharge. The process of finding matching proposals to a given acceptance is specified by the axiom *getProposeForAccept*, which will be defined shortly.

$$
\begin{array}{l}
\textit{ProcessRejection} \\
\hline
agent?, speaker?, addressee? : \downarrow Agent \\
reject? : \downarrow Reject \\
time? : Time \\
result! : \mathbb{P}\ Obligation \\
\hline
\exists\, setset : \mathbb{P}(\mathbb{P}\ Obligation) \mid \\
\quad setset = \{ set : \mathbb{P}\ Obligation \mid \\
\qquad \forall\, propose : getProposeForReject(agent?.history, \\
\qquad\qquad\qquad\qquad\qquad\qquad speaker?, addressee?, \\
\qquad\qquad\qquad\qquad\qquad\qquad reject?, time?) \\
\qquad \bullet\ \exists\, speaking : \mathbb{P}\ Speaking; \\
\qquad\quad voicing : \mathbb{P}\ Voicing; \\
\qquad\quad hearing : \mathbb{P}\ Hearing \mid \\
\qquad\quad speaking = \{ sa : Speaking \mid \\
\qquad\qquad\qquad sa.time = propose.reply \land \\
\qquad\qquad\qquad \{sa.speaker, sa.addressee\} = \{speaker?, addressee?\} \land \\
\qquad\qquad\qquad sa.points = isReplyTo(propose)\} \land \\
\qquad\quad voicing = \{ v : Voicing \mid \forall\, sa : speaking \\
\qquad\qquad\qquad\qquad \bullet\ sa.voice = v\} \land \\
\qquad\quad hearing = \{ h : Hearing \mid \forall\, sa : speaking \\
\qquad\qquad\qquad\qquad \bullet\ sa.hear = h\} \\
\qquad\ \bullet\ \exists\, tmp : \mathbb{P} \downarrow Acting \mid tmp = speaking \\
\qquad\quad \bullet\ set = tmp \cup voicing \cup hearing\} \\
\bullet\ (\forall\, o : result! \bullet \exists\, set : setset \bullet o \in set) \land \\
\quad (\forall\, o : Obligation \mid \exists\, set : setset \bullet o \in set \bullet o \in result!)
\end{array}
$$

$$
\begin{array}{l}
ProcessUtterance \mathrel{\widehat{=}} [\, agent? : \downarrow Agent;\ u? : Utterance \,] \bullet \\
\quad (\bigwedge accept : Accept \mid accept \in u?.speechAct.points \\
\quad\ \bullet\ ProcessAcceptance \mathbin{;} agent?.DeleteObligations) \land \\
\quad (\bigwedge reject : \downarrow Reject \mid reject \in u?.speechAct.points \\
\quad\ \bullet\ ProcessRejection \mathbin{;} agent?.DeleteObligations)
\end{array}
$$

Figure 11. *Policy 2*: A reply releases agents of the obligation to reply (part 2 of 2).

Similar to the operation *ProcessAcceptance*, the operation *ProcessRejection* specifies that for each given valid rejection that matches one or more past, non-replied to proposals there exist obligations to reply that will be later discarded from the state of the agent. As before, this operation takes a rejection and checks whether or not there are matching proposals in the history of past utterances of the agent.[25] If such proposals exist then the history of utterances is analysed once more to detect whether or not each of these proposals has been replied to. If at least one proposal has not been replied to, then the rejection is a legitimate reply to the proposal, and thus the corresponding obligations to reply are selected for discharge. The process of finding matching proposals to a given rejection is specified by the axiom *getProposeForReject*, which is defined next.

Querying the History of Utterances

The PFP specifies that conversations negotiating shared social commitments are patterns of proposals followed by an acceptance, a rejection or a counterproposal. To keep track of these patterns, agents maintain a history of the utterances in which they have participated. When an agent receives an acceptance or a rejection, the history of utterances is analysed to determine whether or not there is a matching past proposal (i.e., a not-yet-replied proposal with an identical operation on commitment as that of the acceptance or rejection) to derive that this acceptance or rejection is part of an ongoing conversation.

These queries are supported by the axioms *getProposeForAccept* and *getProposeForReject*. The axiom *getProposeForAccept* (shown in Figure 12) specifies that if there exists a past proposal from addressee to speaker that

- is still answerable at the time that the acceptance occurred;
- proposes the same operation on commitment as that of the acceptance; and
- for which a later acceptance or rejection replying to this proposal does not exist;

then this proposal is a member of the set returned by the function.

The axiom *getProposeForReject* (shown in Figure 13) is specified similarly.

[25] In the case of acceptances, a matching past proposal is one that was uttered by the addressee to the speaker of the acceptance. However, in the case of a rejection the scenario is slightly more complex, since the rejection to a proposal can come from either the speaker or addressee of the proposal.

$$
\begin{array}{l}
getProposeForAccept : \mathrm{seq}\, Utterance \times Agent \times Agent \times \\
\qquad\qquad Accept \times Time \rightarrow \mathbb{P} \downarrow Propose \\
\hline
\forall\, utterances : \mathrm{seq}\, Utterance;\ speaker, addressee : Agent; \\
\qquad accept : Accept;\ time : Time \\
\bullet\ \exists\, result : \mathbb{P} \downarrow Propose \mid \\
\quad result = \{ p : \downarrow Propose \mid \\
\qquad\quad \forall\, u_1 : Utterance \mid \\
\qquad\qquad u_1 \in \mathrm{ran}\, utterances \wedge \\
\qquad\qquad u_1.time < time \wedge \\
\qquad\qquad u_1.speechAct.speaker = addressee \wedge \\
\qquad\qquad u_1.speechAct.addressee = speaker \\
\qquad\quad \bullet\ \exists\, p_1 : \downarrow Propose \mid \\
\qquad\qquad p_1 \in u_1.speechAct.points \wedge \\
\qquad\qquad p_1.reply.from \leq time \leq p_1.reply.until \wedge \\
\qquad\qquad p_1.proposing = accept.accepting \wedge \\
\qquad\qquad \neg\, (\exists\, u_2 : Utterance \mid \\
\qquad\qquad\quad u_2 \in \mathrm{ran}\, utterances \wedge \\
\qquad\qquad\quad u_1.time < u_2.time < time \wedge \\
\qquad\qquad\quad p_1.reply.from \leq u_2.time \leq p_1.reply.until \\
\qquad\qquad \bullet\ (\exists\, a_1 : Accept \mid \\
\qquad\qquad\quad a_1 \in u_2.speechAct.points \wedge \\
\qquad\qquad\quad a_1.accepting = p_1.proposing \wedge \\
\qquad\qquad\quad u_2.speechAct.speaker = u_1.speechAct.addressee \wedge \\
\qquad\qquad\quad u_2.speechAct.addressee = u_1.speechAct.speaker \\
\qquad\qquad\quad \bullet\ \neg\, (\exists\, r_1 : \downarrow Reject \mid \\
\qquad\qquad\qquad r_1 \in u_2.speechAct.points \\
\qquad\qquad\qquad \bullet\ r_1.rejecting = a_1.accepting)) \vee \\
\qquad\qquad (\exists\, r_2 : \downarrow Reject \mid \\
\qquad\qquad\quad r_2 \in u_2.speechAct.points \wedge \\
\qquad\qquad\quad r_2.rejecting = p_1.proposing \wedge \\
\qquad\qquad\quad \{u_1.speechAct.speaker, u_1.speechAct.addressee\} = \\
\qquad\qquad\quad \{u_2.speechAct.speaker, u_2.speechAct.addressee\} \\
\qquad\qquad\quad \bullet\ \neg\, (\exists\, a_2 : Accept \mid \\
\qquad\qquad\qquad a_2 \in u_2.speechAct.points \wedge \\
\qquad\qquad\qquad u_2.speechAct.speaker = u_1.speechAct.addressee \wedge \\
\qquad\qquad\qquad u_2.speechAct.addressee = u_1.speechAct.speaker \\
\qquad\qquad\qquad \bullet\ a_2.accepting = r_2.rejecting))) \\
\qquad\quad \bullet\ p_1 = p \} \\
\quad \bullet\ result = getProposeForAccept(utterances, speaker, addressee, accept, time)
\end{array}
$$

Figure 12. Axiom *getProposeForAccept*: To retrieve a past proposal given an acceptance.

$$
\begin{array}{l}
getProposeForReject : \operatorname{seq} Utterance \times Agent \times Agent \times \\
\qquad\qquad\qquad\qquad \downarrow Reject \times Time \rightarrow \mathbb{P} \downarrow Propose \\
\hline
\forall\, utterances : \operatorname{seq} Utterance;\ speaker, addressee : Agent; \\
\qquad\qquad reject : \downarrow Reject;\ time : Time \\
\bullet\ \exists\, result : \mathbb{P} \downarrow Propose \mid \\
\quad result = \{p : \downarrow Propose \mid \\
\qquad\qquad \forall\, u_1 : Utterance \mid \\
\qquad\qquad\quad u_1 \in \operatorname{ran} utterances \wedge \\
\qquad\qquad\quad u_1.time < time \wedge \\
\qquad\qquad\quad \{u_1.speechAct.speaker, u_1.speechAct.addressee\} = \\
\qquad\qquad\quad \{speaker, addressee\} \\
\qquad\qquad \bullet\ \exists\, p_1 : \downarrow Propose \mid \\
\qquad\qquad\quad p_1 \in u_1.speechAct.points \wedge \\
\qquad\qquad\quad p_1.reply.from \leq time \leq p_1.reply.until \wedge \\
\qquad\qquad\quad p_1.proposing = reject.rejecting \wedge \\
\qquad\qquad\quad \neg\, (\exists\, u_2 : Utterance \mid \\
\qquad\qquad\qquad u_2 \in \operatorname{ran} utterances \wedge \\
\qquad\qquad\qquad u_1.time < u_2.time < time \wedge \\
\qquad\qquad\qquad u_2.time \leq p_1.reply.until \\
\qquad\qquad\quad \bullet\ (\exists\, a_1 : Accept \mid \\
\qquad\qquad\qquad a_1 \in u_2.speechAct.points \wedge \\
\qquad\qquad\qquad a_1.accepting = p_1.proposing \wedge \\
\qquad\qquad\qquad u_2.speechAct.speaker = u_1.speechAct.addressee \wedge \\
\qquad\qquad\qquad u_2.speechAct.addressee = u_1.speechAct.speaker \\
\qquad\qquad\qquad \bullet\ \neg\, (\exists\, r_1 : \downarrow Reject \mid \\
\qquad\qquad\qquad\qquad r_1 \in u_2.speechAct.points \\
\qquad\qquad\qquad\quad \bullet\ r_1.rejecting = a_1.accepting)) \vee \\
\qquad\qquad\qquad (\exists\, r_2 : \downarrow Reject \mid \\
\qquad\qquad\qquad\quad r_2 \in u_2.speechAct.points \wedge \\
\qquad\qquad\qquad\quad r_2.rejecting = p_1.proposing \wedge \\
\qquad\qquad\qquad\quad \{u_1.speechAct.speaker, u_1.speechAct.addressee\} = \\
\qquad\qquad\qquad\quad \{u_2.speechAct.speaker, u_2.speechAct.addressee\} \\
\qquad\qquad\qquad \bullet\ \neg\, (\exists\, a_2 : Accept \mid \\
\qquad\qquad\qquad\qquad a_2 \in u_2.speechAct.points \wedge \\
\qquad\qquad\qquad\qquad u_2.speechAct.speaker = u_1.speechAct.addressee \wedge \\
\qquad\qquad\qquad\qquad u_2.speechAct.addressee = u_1.speechAct.speaker \\
\qquad\qquad\qquad\quad \bullet\ a_2.accepting = r_2.rejecting))) \\
\qquad\qquad \bullet\ p_1 = p\} \\
\quad \bullet\ result = getProposeForReject(utterances, speaker, addressee, reject, time)
\end{array}
$$

Figure 13. Axiom *getProposeForReject*: To retrieve a past proposal given a rejection.

3.9.3 Policy 3: Agreeing to Uptake Social Commitments

The third policy specifies the consequences of participating in PFP conversations: that of adopting and discarding shared social commitments and the obligations that these commitments entail. As shown in Figure 14, this policy is defined as a class inheriting from *ActionNorm* that defines the operations *ProcessAction* (which overrides the empty operation inherited from *ActionNorm*), *ProcessAcceptance* and *ProcessUtterance* (which overrides the abstract operation inherited from *Norm*).

The operation *ProcessUtterance* selects all acceptances in a given utterance and checks whether or not each of these is a valid reply to a past proposal (as indicated by the previously defined axiom *getProposeForAccept*). For each acceptance that satisfies this criterion, this operation invokes the operation *ProcessAcceptance*, which is followed by either the operation *AddCommitments* or *DeleteCommitments* (depending on whether or not the acceptance is adopting or discarding a commitment, respectively). These operations are performed concurrently with the operation *ProcessNorm*, followed by either *AddObligations* or *DeleteObligations* (according to the operation in the acceptance).

Lastly, the operation *ProcessAcceptance* defines that for a given acceptance there exists a shared social commitment that will be later added to (or discarded from) the state of the agent. In the same manner, the operation *ProcessAction* defines that for any given action in which the agent is one of the performers there is the obligation that the agent performs the action (an obligation that will be later added or discarded from the state of the agent).

3.9.4 Policy 4: Adopting Obligations to Propose

There are two additional policies that define responsibility about the adoption and discharge of social commitments.

The first policy states that once a commitment is adopted there is an agent that will propose its discharge. Typical scenarios where this policy is applied are those in which there is a request for results. For example, if Alice has requested the time to Bob, and Bob has accepted the request, then Bob is responsible for proposing the discharge of the action.

$PFPpolicy3$

$\upharpoonleft(ProcessUtterance, ProcessAction)$

$ActionNorm$

$ProcessAction$

$$\begin{array}{l}
agent? : \downarrow Agent \\
acting? : \downarrow Acting \\
result! : Obligation \\
\hline
agent? \in acting?.performers \\
result! = acting?
\end{array}$$

$ProcessAcceptance$

$$\begin{array}{l}
speaker?, addressee? : \downarrow Agent \\
accept? : Accept \\
result! : SharedSocialCommitment \\
\hline
\exists\, sc : SocialCommitment \mid \\
\quad sc = accept?.accepting.commitment \\
\bullet\ result! = (\mu\, ssc : SharedSocialCommitment \mid \\
\qquad\qquad ssc.among = \{speaker?, addressee?\} \land \\
\qquad\qquad ssc.debtor = sc.debtor \land \\
\qquad\qquad ssc.creditor = sc.creditor \land \\
\qquad\qquad ssc.action = sc.action)
\end{array}$$

$$\begin{array}{l}
ProcessUtterance \mathrel{\widehat=} \big[\, agent? : \downarrow Agent;\ u? : Utterance \,\big] \bullet \\
\quad (\bigwedge accept : Accept;\ action : \downarrow Acting \mid \\
\qquad accept \in u?.speechAct.points \land \\
\qquad accept.accepting \in Add \land \\
\qquad accept.accepting.commitment.action = action \land \\
\qquad \#(getProposeForAccept(agent?.history, \\
\qquad\qquad u?.speechAct.speaker, \\
\qquad\qquad u?.speechAct.addressee, \\
\qquad\qquad accept, u?.time)) \geq 1 \\
\quad \bullet\ (ProcessAcceptance \mathbin{⨾} agent?.AddCommitments) \land \\
\quad\ (action.ProcessNorm \mathbin{⨾} agent?.AddObligations)) \land \\
\quad (\bigwedge accept : Accept;\ action : \downarrow Acting \mid \\
\qquad accept \in u?.speechAct.points \land \\
\qquad accept.accepting \in Delete \land \\
\qquad accept.accepting.commitment.action = action \land \\
\qquad \#(getProposeForAccept(agent?.history, \\
\qquad\qquad u?.speechAct.speaker, \\
\qquad\qquad u?.speechAct.addressee, \\
\qquad\qquad accept, u?.time)) \geq 1 \\
\quad \bullet\ (ProcessAcceptance \mathbin{⨾} agent?.DeleteCommitments) \land \\
\quad\ (action.ProcessNorm \mathbin{⨾} agent?.DeleteObligations))
\end{array}$$

Figure 14. *Policy 3*: Acceptances lead to the uptake of commitments and obligations.

The second policy is one that is less frequently used but is no less important. This policy states that once a commitment to action is adopted there is an agent that will propose its adoption (usually to a third agent). One example of the use of this policy is that of an agent (an authoritative figure) requesting another agent (the future proposer) to propose to a third agent the negotiation of a shared social commitment.[26]

Proposing to Discharge

The class *ToProposeToDischarge* is the superclass of all actions that once accepted are to be proposed for discharge. This class inherits from the action *ToSpeak* and defines a *discharger* agent and a *discharged* agent (which are the speaker and addressee, respectively), and specifies that this speech act communicates an illocutionary point proposing the discharge of the action.

$ToProposeToDischarge$

$ToSpeak$

$discharger, discharged : \downarrow Agent$

$$\begin{array}{l}
discharger = speaker \land \\
discharged = addressee \\
\exists\, p : \downarrow Propose \mid \\
\quad p \in points \\
\bullet\ p.proposing \in Delete \land \\
\quad p.proposing.commitment.action = self
\end{array}$$

The fourth policy in the model is specified by the class *ProposingToDischargePolicy*, which is shown in Figure 15. As shown in the figure, this class inherits from *ActionNorm*

[26] The example used to evaluate this policy (which is not recorded in this thesis) was that of a Parent asking a Groom to commit to an action in which he proposes his Bride for marriage. The conversation is initiated by the Dad requesting the Groom to adopt the action *ProposingToMarry* in which the Groom is the agent that proposes to the Bride to adopt the action (which is specified as an action in which the Bride replies with a yes/no response). After accepting the commitment from the Dad, the conversation continues with the Groom proposing to the Bride, who in some cases accepts and in others rejects the proposal.

and defines the operations *ProcessAction* (which overrides the operation inherited from *ActionNorm*) and *ProcessUtterance* (which overrides the operation inherited from *Norm*).

In brief, the operation *ProcessUtterance* specifies that for each acceptance found in the utterance, such that there is a past propose matching the acceptance, the action operation *ProcessAction* is performed followed by one of the agent operations *AddObligations* or *DeleteObligations* (depending on whether or not the acceptance is accepting to add or to delete a social commitment). Lastly, the operation *ProcessAction* specifies that given a *ProposingToDischarge*[27] action there exists an obligation in which the *discharger* agent is to propose to the *discharged* agent the discharge of a social commitment comprising the action.

Proposing to Adopt

The class *ToProposeToAdopt* is the class from which all actions that are proposed for adoption are derived. This class inherits from the action *ToSpeak* and defines an *adopter* and *adopted* agents (which are the speaker an addressee, respectively), and states that this speech act communicates an illocutionary point proposing the adoption of the action.

$ToProposeToAdopt$

$ToSpeak$

$$adopter, adopted : \downarrow Agent$$

$$\begin{array}{l}
adopter = speaker \land \\
adopted = addressee \\
\exists\, p : \downarrow Propose \mid \\
\quad p \in points \\
\bullet\ p.proposing \in Add \land \\
\quad p.proposing.commitment.action = self
\end{array}$$

[27] The class *ProposingToDischarge* is a subclass of *ToProposeToDischarge* and *CompositeActing*.

$$
\begin{array}{l}
\textbf{ProposingToDischargePolicy} \\
\hline
\upharpoonleft(ProcessUtterance, ProcessAction) \\
ActionNorm \\[1ex]
\quad \textbf{ProcessAction} \\
\quad \hline
\quad agent? : \downarrow Agent \\
\quad sc? : SocialCommitment \\
\quad deleting? : \downarrow ProposingToDischarge \\
\quad result! : \mathbb{P}\ Obligation \\
\quad \hline
\quad \exists\, speak : Speaking \mid \\
\quad\quad speak = (\mu\, sa : Speaking \mid \\
\quad\quad\quad\quad sa.time = deleting?.time \land \\
\quad\quad\quad\quad sa.speaker = deleting?.discharger \land \\
\quad\quad\quad\quad sa.addressee = deleting?.discharged \land \\
\quad\quad\quad\quad (\exists\, propose : \downarrow Propose \mid \\
\quad\quad\quad\quad\quad propose \in sa.points \land \\
\quad\quad\quad\quad\quad propose.proposing \in Delete \\
\quad\quad\quad\quad\quad \bullet\ \exists\, sc : SocialCommitment \mid \\
\quad\quad\quad\quad\quad\quad sc.debtor = sc?.debtor \land \\
\quad\quad\quad\quad\quad\quad sc.creditor = sc?.creditor \land \\
\quad\quad\quad\quad\quad\quad sc.action = deleting? \\
\quad\quad\quad\quad\quad\quad \bullet\ sc = propose.proposing.commitment)) \\
\quad \bullet\ (agent? = deleting?.discharger \land result! = \{speak, speak.voice\}) \lor \\
\quad\quad (agent? = deleting?.discharged \land result! = \{speak, speak.hear\}) \\
\quad \hline
ProcessUtterance \mathrel{\widehat=} [\, agent? : \downarrow Agent;\ u? : Utterance \,] \bullet \\
\quad (\bigwedge action : \downarrow Acting \mid \\
\quad\quad \exists\, accept : Accept \mid \\
\quad\quad\quad accept \in u?.speechAct.points \land \\
\quad\quad\quad accept.accepting \in Add \land \\
\quad\quad\quad accept.accepting.commitment.action = action \\
\quad\quad \bullet\ \#(getProposeForAccept(agent?.history, \\
\quad\quad\quad\quad\quad\quad u?.speechAct.speaker, \\
\quad\quad\quad\quad\quad\quad u?.speechAct.addressee, \\
\quad\quad\quad\quad\quad\quad accept, u?.time)) \geq 1 \\
\quad \bullet\ action.ProcessNorm \mathbin{⨾} agent?.AddObligations) \land \\
\quad (\bigwedge action : \downarrow Acting \mid \\
\quad\quad \exists\, accept : Accept \mid \\
\quad\quad\quad accept \in u?.speechAct.points \land \\
\quad\quad\quad accept.accepting \in Delete \land \\
\quad\quad\quad accept.accepting.commitment.action = action \\
\quad\quad \bullet\ \#(getProposeForAccept(agent?.history, \\
\quad\quad\quad\quad\quad\quad u?.speechAct.speaker, \\
\quad\quad\quad\quad\quad\quad u?.speechAct.addressee, \\
\quad\quad\quad\quad\quad\quad accept, u?.time)) \geq 1 \\
\quad \bullet\ action.ProcessNorm \mathbin{⨾} agent?.DeleteObligations) \\
\hline
\end{array}
$$

Figure 15. *Policy 4*: Proposing to discharge an adopted social commitment to action.

$$
\begin{array}{l}
\textit{ProposingToAdoptPolicy} \\
\upharpoonright(\textit{ProcessUtterance}, \textit{ProcessAction}) \\
\textit{ActionNorm} \\
\hline
\quad \textit{speaker}, \textit{addressee} : \downarrow \textit{Agent} \\
\quad \textit{SAonly}, \textit{SAnot} : \mathbb{B} \\
\hline
\quad \textit{ProcessAction} \\
\quad \textit{agent}? : \downarrow \textit{Agent} \\
\quad \textit{sc}? : \textit{SocialCommitment} \\
\quad \textit{adding}? : \downarrow \textit{ProposingToAdopt} \\
\quad \textit{result}! : \mathbb{P}\ \textit{Obligation} \\
\hline
\quad \exists\, \textit{speak} : \textit{Speaking} \mid \\
\qquad \textit{speak} = (\mu\, \textit{sa} : \textit{Speaking} \mid \\
\qquad\qquad\qquad \textit{sa.time} = \textit{adding?.time} \land \\
\qquad\qquad\qquad \textit{sa.speaker} = \textit{adding?.adopter} \land \\
\qquad\qquad\qquad \textit{sa.addressee} = \textit{adding?.adopted} \land \\
\qquad\qquad\qquad (\exists\, \textit{propose} : \downarrow \textit{Propose} \mid \\
\qquad\qquad\qquad\quad \textit{propose} \in \textit{sa.points} \land \\
\qquad\qquad\qquad\quad \textit{propose.proposing} \in \textit{Add} \\
\qquad\qquad\qquad \bullet\ \textit{propose.proposing.commitment} = \textit{sc}?)) \\
\quad \bullet\ (\neg\, \textit{SAonly} \lor \{\textit{speaker}, \textit{addressee}\} = \{\textit{adding?.adopter}, \textit{adding?.adopter}\}) \land \\
\qquad (\neg\, \textit{SAnot} \lor \{\textit{speaker}, \textit{addressee}\} \neq \{\textit{adding?.adopter}, \textit{adding?.adopter}\}) \land \\
\qquad ((\textit{agent}? = \textit{adding?.adopter} \land \textit{result}! = \{\textit{speak}, \textit{speak.voice}\}) \lor \\
\qquad\ (\textit{agent}? = \textit{adding?.adopted} \land \textit{result}! = \{\textit{speak}, \textit{speak.hear}\}))
\end{array}
$$

Figure 16. *Policy 5*: Proposing to adopt a social commitment to action (part 1 of 2).

The fifth policy in the model is specified by class *ProposingToAdoptPolicy*, which is shown in Figure 16 and Figure 17. As shown in these figures, this class inherits from *ActionNorm* and specifies the variables *speaker* and *addressee* (indicating the speaker and addressee of the utterance currently being processed), *SAonly* and *SAnot* (which are auxiliary Boolean variables), and the operations *ProcessAction* (which overrides the operation inherited from *ActionNorm*) and *ProcessUtterance* (which overrides the abstract operation inherited from *Norm*).

$$
\begin{array}{l}
ProcessUtterance \mathrel{\widehat=} [\, agent? : \downarrow Agent;\ u? : Utterance \mid \\
\qquad speaker = u?.speechAct.speaker \land \\
\qquad addressee = u?.speechAct.addressee \,] \bullet \\
\quad (\bigwedge action : \downarrow Acting \mid \\
\qquad \neg\ SAonly \land SAnot \land \\
\qquad (\exists\, accept : Accept \mid \\
\qquad\quad accept \in u?.speechAct.points \land \\
\qquad\quad accept.accepting \in Add \land \\
\qquad\quad accept.accepting.commitment.action = action \\
\qquad\ \bullet \#(getProposeForAccept(agent?.history, speaker, \\
\qquad\qquad\qquad addressee, accept, u?.time)) \geq 1) \\
\quad \bullet\ action.ProcessNorm \ \text{⨟}\ agent?.AddObligations) \land \\
\quad (\bigwedge action : \downarrow Acting \mid \\
\qquad SAonly \land \neg\ SAnot \land \\
\qquad (\exists\, accept : Accept \mid \\
\qquad\quad accept \in u?.speechAct.points \land \\
\qquad\quad accept.accepting \in Add \land \\
\qquad\quad accept.accepting.commitment.action = action \\
\qquad\ \bullet \#(getProposeForAccept(agent?.history, speaker, \\
\qquad\qquad\qquad addressee, accept, u?.time)) \geq 1) \\
\quad \bullet\ action.ProcessNorm \ \text{⨟}\ agent?.DeleteObligations) \land \\
\quad (\bigwedge action : \downarrow Acting \mid \\
\qquad \neg\ SAonly \land \neg\ SAnot \land \\
\qquad (\exists\, accept : Accept \mid \\
\qquad\quad accept \in u?.speechAct.points \land \\
\qquad\quad accept.accepting \in Delete \land \\
\qquad\quad accept.accepting.commitment.action = action \\
\qquad\ \bullet \#(getProposeForAccept(agent?.history, speaker, \\
\qquad\qquad\qquad addressee, accept, u?.time)) \geq 1) \\
\quad \bullet\ action.ProcessNorm \ \text{⨟}\ agent?.DeleteObligations)
\end{array}
$$

Figure 17. *Policy 5*: Proposing to adopt a social commitment to action (part 2 of 2).

The variable *SAonly* indicates that obligations can be generated only if the utterance being processed was uttered between the adopter and the adopted. Likewise, the variable *SAnot* indicates that obligations can be generated only if the utterance occurred between a pair of agents that are not the adopter and adopted. These variables are used in the operations *ProcessUtterance* and *ProcessAction* (which are described below).

The operation *ProcessUtterance* specifies three conjunctive operations that are applied to all acceptances found in the utterance being processed. In brief, the first operation adds obligations to propose adopting the adopted action if this action is being adopted between the adopter and an agent other than the adopted, or the adopted and an agent other than the adopter. The second operation deletes obligations if acceptances occurred between the adopter and the adopted—so that there is no need to have obligations to propose adopting the action now being adopted. And the third operation deletes obligations to propose adopting the action if the action is the one now being discharged.

Lastly, the operation *ProcessAction* specifies that given a *ProposingToAdopt* action there exists an obligation in which the *adopter* agent is to propose to the *adopted* agent the adoption of a social commitment involving the action.

3.10 Normative Agent Societies

A basic premise in this model is that collaborative agents voluntarily participate in normative societies, that is, societies that specify the norms of behaviour that agents in the society are expected to follow (Conte & Castelfranchi, 1995; Conte & Dellarocas, 2001).[28]

3.10.1 Joint Activities

Joint activities are sets of actions which are carried out by an ensemble of agents acting in coordination with each other toward achieving certain dominant goals (Clark, 1996). The superclass of all joint activities in the model is the class *JointActivity*, which defines a sole variable *actions* (to refer to a set of joint actions) and an abstract operation *Interaction* (to define the ideal sequences of communications in the activity).

[28] In this view, agents not only have the autonomy to adopt norms but also the autonomy to abide or disregard them according to their assumed costs of obedience and transgression. Although the model for conversations does not explore this area, other more complex societies may define norms that restore equity and avoid potential injury by making ill-behaved agents liable for their actions.

$JointActivity$

$actions : \mathbb{P} \downarrow JointActing$

$Interaction \mathrel{\widehat{=}} [\,]$

3.10.2 Societies

The class *Society* is the superclass of all societies in the model. This class defines variables referencing a set of norms, a set of joint activities and a set of the agents associated to the society (where all agents participating in any joint activity are part of the society). This class also specifies the operation *ExecuteNorms*, which invokes the operation *Process* for all agents with incoming utterances.

$Society$

$agents : \mathbb{P} \downarrow Agent$
$norms : \mathbb{P} \downarrow Norm$
$activities : \mathbb{P} \downarrow JointActivity$

$\forall\, activity : activities$
$\quad \bullet \forall\, action : activity.actions$
$\qquad \bullet \forall\, performer : action.performers$
$\qquad\quad \bullet\, performer \in agents$

$ExecuteNorms \mathrel{\widehat{=}} \bigwedge a : agents \mid \#a.inbox \geq 1 \bullet a.Process$

Lastly, the class *PFPsociety* (below) is defined to denote those societies that have as their norms the conversation policies for the *Protocol for Proposals* and the policies for proposing to adopt and to discharge.

$PFPsociety$

$Society$

$\textsc{Init}$

$(\exists_1\, p_1 : PFPpolicy1 \bullet p_1 \in norms)$
$(\exists_1\, p_2 : PFPpolicy2 \bullet p_2 \in norms)$
$(\exists_1\, p_3 : PFPpolicy3 \bullet p_3 \in norms)$
$(\exists_1\, p_4 : ProposingToAdoptPolicy \bullet p_4 \in norms)$
$(\exists_1\, p_5 : ProposingToDischargePolicy \bullet p_5 \in norms)$

3.11 Summary

This chapter introduced a model for conversations for action, whose fundamental elements support the negotiation of shared social commitments. This chapter also described the elements in the model, such as actions, social commitments, illocutionary points, norms and conversation policies.

These elements can be classified in three groups: first are those elements that describe communicational tokens, that is, elements that define *conversational identity* (e.g., speech acts, illocutionary points, social commitments, operations); second are those elements that show how these tokens are assembled into structured conversational sequences, which define *conversational use* (e.g., policies that generate and discharge obligations to reply); and lastly are those elements that indicate the expected outcome of specific utterance sequences, that is, those elements that define *conversational consequences* (e.g., policies that adopt and discard negotiated shared social commitments).

Subsequent chapters provide examples of how these conversational elements support the cooperative behaviour of agents in practical domains.

Chapter 4
Example: The Contract Net Protocol

4.1 Overview

This chapter illustrates how the model for conversations can be used to model the interaction of agents in a joint activity. Specifically, this chapter focuses on the Contract Net Protocol, which is a task allocation mechanism often used in multi-agent systems.

This chapter begins with a section describing the Contract Net Protocol. Subsequent sections describe the conversational notions used for modelling the protocol as a joint activity: namely actions, social commitments, illocutionary points and agent participations. This chapter concludes with a detailed account of a conversation example between agents using the protocol.

4.2 The Contract Net Protocol

The *Contract Net Protocol* (Smith, 1980) is a high-level protocol for the negotiation and delegation of actions among distributed agents. This protocol is described as a mutual selection strategy in which an agent (the *manager*) delegates the performance of actions to suitable candidates (the *contractors*) from a collection of contender agents (the *bidders*).

An instance of the Contract Net Protocol (CNP) begins when a manager attempts to delegate actions by sending a request for bids to agents that could potentially perform these actions. Agents who are willing submit a bid showing their abilities to perform these actions. The manager then evaluates the submitted bids and selects the most suitable bidder and awards the contract to that bidder (i.e., offers the execution of the actions). Once the contract is awarded, the acceptance of this awarding makes the awarded agent the contractor, that is, the agent executing the actions. Finally, the protocol terminates when the contractor submits the results of the actions to the manager.[29]

To model the CNP in terms of the model for conversations, it is necessary to identify the various information, actions, social commitments and illocutionary points that take place in interactions between managers and contractors. Sections below define these elements.

4.3 Information

There are five types of information that can be communicated in the CNP:

- the requirements (for producing a bid),
- the bids themselves,
- the contracts,
- the notification of the awarding or rejection of bids, and
- the results of executing a contract.

The class *Requirement* is a subclass of *Data* which specifies an action and its constraints (where the constraints could be action dependencies, maximum costs afforded, times of expected execution, and so on). The requirements for a bid is a set of instances of this class.

[29] Although this description is the more representative of the CNP, minor variants exist. As described by R.G. Smith (1980), variations include, for example, those where the contractor transmits preliminary results while executing an action, and those with various contractors simultaneously performing actions.

$Requirement$

$Data$

$action : \downarrow Action$
$constraints : \mathbb{P} \downarrow Data$

The type definitions *BidItem* and *ContractItem* specify the types of the items that compose bids and contracts, respectively. These items are defined to have the same type as a requirement.

$BidItem == Requirement$
$ContractItem == Requirement$

The class *EvaluationResult* is used for notifying whether or not a bid has been awarded for execution. This class (which inherits from *Data*) solely defines a Boolean value to indicate the awarding (or not) of a contract.

$EvaluationResult$

$Data$

$true : \mathbb{B}$

Lastly, the class *ResultsItem* defines the type of the instances returned after executing a contract. This class inherits from *Requirement* and defines the variable *outcome* to hold the results of executing the inherited action.

$ResultsItem$

$Requirement$

$outcome : \downarrow Data$

4.4 Actions

There are three actions involved in the CNP:

- *To submit a bid*: in which a bidder creates and submits a bid to a manager.

- *To evaluate a bid*: in which a manager evaluates a bid and informs a bidder of the outcome of this evaluation, and
- *To execute a contract*: in which a bidder executes an awarded contract and submits its results to a manager.

These actions are defined in the subsections below.

4.4.1 Bidding

Bidding is an action in which an agent produces and announces a bid to another agent. This action is modelled as a class named *ToBid* that inherits from *ToProduce* and *ToProposeToDischarge*, and which declares the variables *bidder* (as the *producer* and *discharger*), *manager* (as the *receiver* and *discharged*), and *requirements* (as the abiding criteria for the production of the bid). It also specifies that the data produced and communicated as a bid (i.e., a set of objects of type *BidItem*).

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{ToBid}} \\
\hline
\textit{ToProduce} \\
\textit{ToProposeToDischarge} \\
\begin{array}{|l}
\hline
\textit{manager} : \downarrow \textit{Manager} \\
\textit{bidder} : \downarrow \textit{Contractor} \\
\textit{requirements} : \mathbb{P}_1\ \textit{Requirement} \\
\hline
\textit{bidder} = \textit{producer} = \textit{discharger} \land \\
\textit{manager} = \textit{receiver} = \textit{discharged} \land \\
\textit{requirements} = \textit{in} \\
\exists\, \textit{bid} : \mathbb{P}\ \textit{BidItem} \\
\bullet\ \textit{bid} = \textit{out} \\
\hline
\end{array} \\
\hline
\end{array}
$$

Based on this definition, the type *Bidding* is defined as the union of (i.e., as the polymorphic type of) the classes *ToBid*, *CompositeActing* and *ProposingToDischarge*.[30]

[30] Note that the time interval specified in the classes *CompositeActing* and *ProposingToDischarge* gets unified in this definition. This unification specifies that a proposal to discharge the action will occur within the interval when the action is performed.

$Bidding == ToBid \bigcup CompositeActing \bigcup ProposingToDischarge$

4.4.2 Evaluating a Bid

Evaluating a bid is an action in which a manager announces to a bidder whether or not a submitted bid is awarded as the contract for execution. This action is defined as a class named *ToEvaluateBid* that inherits from *ToProduce* and *ToProposeToDischarge* and which declares the variables *manager* (as the *producer* and *discharger*), *bidder* (as the *receiver* and *discharged*) and *bid* (as the bid submitted for evaluation). This definition also specifies that the awarding (or not) of a contract is communicated through an object of type *EvaluationResult*.

$ToEvaluateBid$

$ToProduce$

$ToProposeToDischarge$

$$\begin{array}{l}
manager : \downarrow Manager \\
bidder : \downarrow Contractor \\
bid : \mathbb{P}\ BidItem \\
\hline
manager = producer = discharger \land \\
bidder = receiver = discharged \land \\
bid = in \\
\exists\, awarded : EvaluationResult \\
\bullet\ \{awarded\} = out
\end{array}$$

Based on this definition, the action *EvaluatingBid* is defined as the union of the classes *ToEvaluateBid*, *CompositeActing* and *ProposingToDischarge*.

$EvaluatingBid == ToEvaluateBid \bigcup CompositeActing \bigcup ProposingToDischarge$

4.4.3 Executing a Contract

Performing a contract is an action in which a contractor executes a contract and communicates the results to the manager. This action is defined as a class named *ToExecuteContract* that inherits from *ToProduce* and *ToProposeToDischarge* and which declares the variables *contractor* (as the *producer* and *discharger*), *manager* (as the *receiver* and *discharged*) and *contract* (as the actions to execute). This definition also specifies that this action results in a non-empty set of instances of type *ResultsItem*.

$$
\begin{array}{l}
\textit{ToExecuteContract} \\
\hline
\textit{ToProduce} \\
\textit{ToProposeToDischarge} \\
\hline
\quad \textit{manager} : \downarrow \textit{Manager} \\
\quad \textit{contractor} : \downarrow \textit{Contractor} \\
\quad \textit{contract} : \mathbb{P}_1\ \textit{ContractItem} \\
\hline
\quad \textit{contractor} = \textit{producer} = \textit{discharger} \wedge \\
\quad \textit{manager} = \textit{receiver} = \textit{discharged} \wedge \\
\quad \textit{contract} = \textit{in} \\
\quad \exists\, \textit{results} : \mathbb{P}_1\ \textit{ResultsItem} \\
\quad \bullet\ \textit{results} = \textit{out}
\end{array}
$$

Based on this definition, the action *ExecutingContract* is defined as the union of the classes *ToExecuteContract*, *CompositeActing* and *ProposingToDischarge*.

$$\textit{ExecutingContract} == \textit{ToExecuteContract} \bigcup \textit{CompositeActing} \bigcup \textit{ProposingToDischarge}$$

4.5 Social Commitments

Three axioms identify the social commitments that involve the aforementioned actions. As such, these axioms specify *commitments to bid*, *commitments to evaluate a bid*, and *commitments to execute a contract*.

4.5.1 Commitment to Bid

The axiom *isCommitmentToBid* is a function that receives a *Bidding* action and returns a social commitment that has as its creditor and debtor the manager and bidder of the action, and where the action of the commitment is the given *Bidding* action.

$$
\begin{array}{|l}
isCommitmentToBid : Bidding \rightarrow SocialCommitment \\
\hline
\forall\, bidding : Bidding \\
\bullet\ \exists\, sc : SocialCommitment \mid \\
\quad sc.creditor = bidding.manager \land \\
\quad sc.debtor = bidding.bidder \land \\
\quad sc.action = bidding \\
\quad \bullet\ sc = isCommitmentToBid(bidding)
\end{array}
$$

4.5.2 Commitment to Evaluate a Bid

Along the same lines, the axiom *isCommitmentToEvaluateBid* is a function that receives an *EvaluatingBid* action and returns a social commitment that has as its creditor and debtor the bidder and manager of the action, and where the action of the commitment is the given *EvaluatingBid* action.

$$
\begin{array}{|l}
isCommitmentToEvaluateBid : EvaluatingBid \rightarrow SocialCommitment \\
\hline
\forall\, evaluating : EvaluatingBid \\
\bullet\ \exists\, sc : SocialCommitment \mid \\
\quad sc.creditor = evaluating.bidder \land \\
\quad sc.debtor = evaluating.manager \land \\
\quad sc.action = evaluating \\
\quad \bullet\ sc = isCommitmentToEvaluateBid(evaluating)
\end{array}
$$

4.5.3 Commitment to Execute a Contract

Lastly, the axiom *isCommitmentToExecuteContract* is a function that receives an *ExecutingContract* action and returns a social commitment that has as its creditor and debtor the manager and contractor of the action, where the action of the commitment is the given *EvaluatingBid* action.

$$
\begin{array}{|l}
isCommitmentToExecuteContract : ExecutingContract \rightarrow SocialCommitment \\
\hline
\forall\, executing : ExecutingContract \\
\bullet\ \exists\, sc : SocialCommitment \mid \\
\quad sc.creditor = executing.manager \land \\
\quad sc.debtor = executing.contractor \land \\
\quad sc.action = executing \\
\quad \bullet\ sc = isCommitmentToExecuteContract(executing)
\end{array}
$$

The next section shows how these axioms support the definition of the illocutionary points used in CNP interactions.

4.6 Illocutionary Points

This section specifies the composition of the illocutionary points used by agents to negotiate commitments in the CNP. These illocutionary points include *proposals to submit a bid*, *acceptances to execute a contract*, and *informing the submission of results*, among others.

4.6.1 Proposing to Bid

CNP interactions begin when a manager requests that a prospective bidder produce and submit a bid that adheres to certain required criteria. This communication is supported by the axioms *isProposeToAdoptBidding* and *isInformRequirements*.

The axiom *isProposeToAdoptBidding* defines a function that receives a *Bidding* action and an interval time within which a reply is expected, and returns a proposal to adopt a social commitment to the bidding action.

$$
\begin{array}{|l}
isProposeToAdoptBidding : Bidding \times Interval \rightarrow \downarrow Propose \\
\hline
\forall\, bidding : Bidding;\ reply : Interval \\
\bullet\ \exists\, propose : \downarrow Propose \mid \\
\quad propose.proposing \in Add \wedge \\
\quad propose.reply = reply \wedge \\
\quad propose.proposing.commitment = isCommitmentToBid(bidding) \\
\quad \bullet\ propose = isProposeToAdoptBidding(bidding, reply)
\end{array}
$$

In addition, the axiom *isInformRequirements* is a function that receives a set of requirements as input and returns an inform containing such requirements.

$$
\begin{array}{|l}
isInformRequirements : \mathbb{P}_1\, Requirement \rightarrow Inform \\
\hline
\forall\, requirements : \mathbb{P}_1\, Requirement \\
\bullet\ \exists\, inform : Inform \mid \\
\quad inform.informing = requirements \\
\quad \bullet\ inform = isInformRequirements(requirements)
\end{array}
$$

4.6.2 Accepting or Rejecting to Bid

Once a request for bids has been issued, it is expected that the request will be replied to with an acceptance or a rejection (as specified by the *Protocol for Proposals*). To support the acceptances to such requests, the axiom *isAcceptToAdoptBidding* defines a function that receives a *Bidding* action and returns an acceptance to adopt committing to this action.

$$\begin{array}{|l}
isAcceptToAdoptBidding : Bidding \rightarrow Accept \\
\hline
\forall\, bidding : Bidding \\
\bullet\ \exists\, accept : Accept \mid \\
\quad accept.accepting \in Add\ \wedge \\
\quad accept.accepting.commitment = isCommitmentToBid(bidding) \\
\quad \bullet\ accept = isAcceptToAdoptBidding(bidding)
\end{array}$$

Agents receiving a request for bids can decline to commit to such a request (e.g., they are not capable of performing the requested actions, they cannot accommodate the execution of the actions under current state constraints). As such, the axiom *isRejectToAdoptBidding* defines a function that receives a *Bidding* action as input and returns a rejection to commit to this action.

$$\begin{array}{|l}
isRejectToAdoptBidding : Bidding \rightarrow \downarrow Reject \\
\hline
\forall\, bidding : Bidding \\
\bullet\ \exists\, reply : \downarrow Reject \mid \\
\quad reply.rejecting \in Add\ \wedge \\
\quad reply.rejecting.commitment = isCommitmentToBid(bidding) \\
\quad \bullet\ reply = isRejectToAdoptBidding(bidding)
\end{array}$$

4.6.3 Submitting a Bid for Evaluation

In the event that a request for bid is accepted, the bidder is now responsible for producing and submitting a bid to the manager. This implies that once that the bid is created the bidder needs to submit this bid, propose discharging the commitment that she is to submit it, and propose that the manager evaluate it.

This communication is modelled through the axioms *isProposeToDischargeBidding*, *isInformBid* and *isProposeToAdoptEvaluating*, which specify the illocutionary points for proposing to discharge the commitment to bid, for informing a bid, and for proposing to adopt the commitment to evaluate a bid, respectively.

Proposing to Discharge Bidding

The axiom *isProposeToDischargeBidding* is a function that receives a *Bidding* action and an interval (within which a reply is expected), and returns a proposal to discharge a social commitment to do the given bidding action.

$$
\begin{array}{|l}
isProposeToDischargeBidding : Bidding \times Interval \rightarrow \downarrow Propose \\
\hline
\forall\, bidding : Bidding;\; reply : Interval \\
\bullet\ \exists\, propose : \downarrow Propose \mid \\
\quad propose.proposing \in Delete \land \\
\quad propose.reply = reply \land \\
\quad propose.proposing.commitment = isCommitmentToBid(bidding) \\
\quad \bullet\ propose = isProposeToDischargeBidding(bidding, reply)
\end{array}
$$

Informing a Bid

The axiom *isInformBid* is a function that receives a bid and returns an inform containing such a bid.

$$
\begin{array}{|l}
isInformBid : \mathbb{P}\ BidItem \rightarrow Inform \\
\hline
\forall\, bid : \mathbb{P}\ BidItem \\
\bullet\ \exists\, inform : Inform \mid \\
\quad inform.informing = bid \\
\quad \bullet\ inform = isInformBid(bid)
\end{array}
$$

Proposing to Adopt Evaluating a Bid

Finally, the axiom *isProposeToAdoptEvaluating* is a function that receives an *EvaluatingBid* action and an interval specifying the expected reply time, and returns a proposal to adopt a social commitment to perform this evaluating action.

$$
\begin{array}{|l}
isProposeToAdoptEvaluating : EvaluatingBid \times Interval \rightarrow \downarrow Propose \\
\hline
\forall\, evaluating : EvaluatingBid;\ reply : Interval \\
\bullet\ \exists\, propose : \downarrow Propose \mid \\
\quad propose.proposing \in Add \wedge \\
\quad propose.reply = reply \wedge \\
\quad propose.proposing.commitment = isCommitmentToEvaluateBid(evaluating) \\
\quad \bullet\ propose = isProposeToAdoptEvaluating(evaluating, reply)
\end{array}
$$

4.6.4 Accepting To Evaluate a Bid

Once a bid has been submitted for evaluation, it is expected that the manager receiving the bid will reply both to the proposal that the bidder is no longer committed to submit a bid, and to the proposal that he evaluate the just submitted bid.

These acceptances are modelled through the axioms *isAcceptToDischargeBidding* and *isAcceptToAdoptEvaluating*, which specify the illocutionary points for accepting the discharge of bidding, and accepting the adoption of evaluating a bid, respectively.

Accepting to Discharge Bidding

The axiom *isAcceptToDischargeBidding* is a function that receives a *Bidding* action as input and returns an acceptance to discharge a social commitment to this bidding action.

$$
\begin{array}{|l}
isAcceptToDischargeBidding : Bidding \rightarrow Accept \\
\hline
\forall\, bidding : Bidding \\
\bullet\ \exists\, accept : Accept \mid \\
\quad accept.accepting \in Delete \wedge \\
\quad accept.accepting.commitment = isCommitmentToBid(bidding) \\
\quad \bullet\ accept = isAcceptToDischargeBidding(bidding)
\end{array}
$$

Accepting to Adopt Evaluating a Bid

Likewise, the axiom *isAcceptToAdoptEvaluating* is a function that receives an *EvaluatingBid* action as input and returns an acceptance to adopt a social commitment to perform this evaluating action.

$$
\begin{array}{|l}
isAcceptToAdoptEvaluating : EvaluatingBid \rightarrow Accept \\
\hline
\forall\, evaluating : EvaluatingBid \\
\bullet\ \exists\, accept : Accept \mid \\
\quad accept.accepting \in Add \land \\
\quad accept.accepting.commitment = isCommitmentToEvaluateBid(evaluating) \\
\quad \bullet\ accept = isAcceptToAdoptEvaluating(evaluating)
\end{array}
$$

4.6.5 Awarding a Contract

After evaluating the merits of a bid, a manager is expected to communicate to the bidder whether or not she is awarded the execution of the contract. To communicate this awarding, the manager sends:

- a proposal to discharge the commitment that he evaluates the bid (given that he has reached a conclusion),
- the affirmative result of the evaluation,
- the awarded contract, and
- a proposal to adopt a commitment in which the bidder does this contract.[31]

This communication is supported by the axioms *isProposeToDischargeEvaluating*, *isInformEvaluation*, *isInformContract* and *isProposeToAdoptExecuting*, which specify the illocutionary points for proposing to discharge evaluating, for informing the result of the evaluation, for informing a contract, and for proposing to adopt executing the contract, respectively.

Proposing to Discharge Evaluating a Bid

The axiom *isProposeToDischargeEvaluating* is a function that receives an *EvaluatingBid* action and an interval specifying an expected reply time, and returns a proposal to discharge a social commitment to do the evaluation.

[31] The alternate outcome to the awarding of a contract is the rejection of the bid. This rejection is simply defined as the issuing of a proposal to discharge the evaluation of the bid along with an inform in which the value of the result of the evaluation is false.

$$
\begin{array}{|l}
isProposeToDischargeEvaluating : EvaluatingBid \times Interval \rightarrow \downarrow Propose \\
\hline
\forall\, evaluating : EvaluatingBid;\ reply : Interval \\
\bullet\ \exists\, propose : \downarrow Propose \mid \\
\quad propose.proposing \in Delete \land \\
\quad propose.reply = reply \land \\
\quad propose.proposing.commitment = isCommitmentToEvaluateBid(evaluating) \\
\quad \bullet\ propose = isProposeToDischargeEvaluating(evaluating, reply)
\end{array}
$$

Informing the Results of an Evaluation

The axiom *isInformEvaluation* is a function that receives an instance of type *EvaluationResult* and returns an inform for this result.

$$
\begin{array}{|l}
isInformEvaluation : EvaluationResult \rightarrow Inform \\
\hline
\forall\, decision : EvaluationResult \\
\bullet\ \exists\, inform : Inform \mid \\
\quad inform.informing = \{decision\} \\
\quad \bullet\ inform = isInformEvaluation(decision)
\end{array}
$$

Informing a Contract

Likewise, the axiom *isInformContract* is a function that informs a contract.

$$
\begin{array}{|l}
isInformContract : \mathbb{P}\ ContractItem \rightarrow Inform \\
\hline
\forall\, contract : \mathbb{P}\ ContractItem \\
\bullet\ \exists\, inform : Inform \mid \\
\quad inform.informing = contract \\
\quad \bullet\ inform = isInformContract(contract)
\end{array}
$$

Proposing to Adopt Executing a Contract

Lastly, the axiom *isProposeToAdoptExecuting* is a function that receives an instance of type *ExecutingContract* and an interval indicating an expected time of reply, and returns a proposal to adopt this executing action.

$$
\begin{array}{|l}
isProposeToAdoptExecuting : ExecutingContract \times Interval \rightarrow \downarrow Propose \\
\hline
\forall\, executing : ExecutingContract;\ reply : Interval \\
\bullet\ \exists\, propose : \downarrow Propose \mid \\
\quad propose.proposing \in Add \land \\
\quad propose.reply = reply \land \\
\quad propose.proposing.commitment = isCommitmentToExecuteContract(executing) \\
\quad \bullet\ propose = isProposeToAdoptExecuting(executing, reply)
\end{array}
$$

4.6.6 Accepting the Evaluation of a Bid

After a manager announces the outcome of an evaluation, it is expected that the (failed or awarded) bidder will acknowledge this outcome by accepting that the manager is no longer committed to evaluate the bid.

To that end, the axiom *isAcceptToDischargeEvaluating* is defined as a function that returns an illocutionary point accepting to discharge a commitment to evaluate a bid.

$$
\begin{array}{|l}
isAcceptToDischargeEvaluating : EvaluatingBid \rightarrow Accept \\
\hline
\forall\, evaluating : EvaluatingBid \\
\bullet\ \exists\, accept : Accept \mid \\
\quad accept.accepting \in Delete \land \\
\quad accept.accepting.commitment = isCommitmentToEvaluateBid(evaluating) \\
\quad \bullet\ accept = isAcceptToDischargeEvaluating(evaluating)
\end{array}
$$

4.6.7 Accepting or Rejecting the Awarding of a Contract

Once a contract has been awarded, the bidder must confirm whether or not she will execute the contract by accepting or rejecting the proposal for its execution.[32]

[32] One reason for agents to reject the awarding of a contract is that they lack the necessary resources for its execution. As explained by J. Ferber (1999), one strategy to allocate resources for bids is to secure these resources at the time of bidding. This strategy, which he called *early commitment*, allows the straightforward execution of a contract (since resources are allocated beforehand), but at the cost of their sub-optimal use if the awarding does not happen. A second strategy consists of submitting a bid without securing the resources needed for execution. The disadvantage of this strategy, which he called *late commitment*, is that agents may be unable to allocate the resources for executing a contract if these resources are scarce at the time of the awarding. In such cases, agents may have no other option than to reject the awarding of the contract.

As such, the axioms *isAcceptToAdoptExecuting* and *isRejectToAdoptExecuting* define the illocutionary points for accepting and rejecting the adoption of a commitment to execute a contract, respectively.

$$
\begin{array}{|l}
isAcceptToAdoptExecuting : ExecutingContract \rightarrow Accept \\
\hline
\forall\, executing : ExecutingContract \\
\bullet\ \exists\, accept : Accept \mid \\
\quad accept.accepting \in Add \wedge \\
\quad accept.accepting.commitment = isCommitmentToExecuteContract(executing) \\
\quad \bullet\ accept = isAcceptToAdoptExecuting(executing)
\end{array}
$$

$$
\begin{array}{|l}
isRejectToAdoptExecuting : ExecutingContract \rightarrow \downarrow Reject \\
\hline
\forall\, executing : ExecutingContract \\
\bullet\ \exists\, reject : \downarrow Reject \mid \\
\quad reject.rejecting \in Add \wedge \\
\quad reject.rejecting.commitment = isCommitmentToExecuteContract(executing) \\
\quad \bullet\ reject = isRejectToAdoptExecuting(executing)
\end{array}
$$

4.6.8 Executing a Contract and Submitting Results

In the event that a contractor accepts to execute the awarded action, it is expected that the results of the action will be communicated to the manager.

To that end, the axioms *isProposeToDischargeExecuting* and *isInformResults* define the illocutionary points for proposing to discharge executing the contract, and for submitting the results of its execution, respectively.

$$
\begin{array}{|l}
isProposeToDischargeExecuting : ExecutingContract \times Interval \rightarrow \downarrow Propose \\
\hline
\forall\, executing : ExecutingContract;\ reply : Interval \\
\bullet\ \exists\, propose : \downarrow Propose \mid \\
\quad propose.proposing \in Delete \wedge \\
\quad propose.reply = reply \wedge \\
\quad propose.proposing.commitment = isCommitmentToExecuteContract(executing) \\
\quad \bullet\ propose = isProposeToDischargeExecuting(executing, reply)
\end{array}
$$

$$
\begin{array}{|l}
isInformResults : \mathbb{P}\ ResultsItem \rightarrow Inform \\
\hline
\forall\ results : \mathbb{P}\ ResultsItem \\
\bullet\ \exists\ inform : Inform \mid \\
\quad inform.informing = results \\
\quad \bullet\ inform = isInformResults(results)
\end{array}
$$

4.6.9 Accepting the Results of a Contract

Lastly, after the contract has been executed and the results submitted to the manager, it is expected (if these results are satisfactory) that the manager will acknowledge their acceptance.

As such, the axiom *isAcceptToDischargeExecuting* defines the illocutionary point that accepts the discharge of executing a contract.

$$
\begin{array}{|l}
isAcceptToDischargeExecuting : ExecutingContract \rightarrow Accept \\
\hline
\forall\ executing : ExecutingContract \\
\bullet\ \exists\ accept : Accept \mid \\
\quad accept.accepting \in Delete\ \wedge \\
\quad accept.accepting.commitment = isCommitmentToExecuteContract(executing) \\
\quad \bullet\ accept = isAcceptToDischargeExecuting(executing)
\end{array}
$$

To briefly recap, this section has introduced illocutionary point definitions for the CNP. The next section will show how these definitions model the communicational participations of agents interacting in the CNP.

4.7 Participants

The CNP involves two types of participants: a *manager* and a *contractor*.

As expected from a behavioural model, the interactions of these participants are specified as utterances constrained by committal preconditions. This means that (for example) for a contractor to submit a bid it is required that there exists an obligation in which she submits a bid.

The following subsections describe the communicational participations of managers and contractors under this view.

4.7.1 Manager

The class *Manager* (which is shown in Figure 18 and Figure 19) is a subclass of *Agent* that defines operations for requesting a bid, for accepting a bid for evaluation, for notifying whether or not a contract is awarded for execution, and for receiving the results of executing a contract. These operations (along with other private operations that support them) are described in the subsections below.

Requesting a Bid

The operation *RequestingBid* defines the behaviour for requesting a bid. This operation is defined as the sequential composition of the operations *ProposeToAdoptBidding* and *SendUtterance*, which are described below.

The operation *ProposeToAdoptBidding* returns a speech act where the speaker and the addressee are the manager and bidder of the provided *Bidding* action (and where the manager/speaker is also the current manager instance executing the operation). This definition also specifies that the resulting speech act contains a proposal to adopt a commitment to bid and an inform indicating the requirements for the bid.[33]

Lastly, the operation *SendUtterance*, which is inherited from the class *Agent*, communicates a speech act (the one resulting from *ProposeToAdoptBidding*) between the speaker and the addressee of the speech act.

[33] This *Inform* illocutionary point informs the same requirements as those listed in the bidding action being proposed. This duplication was allowed for clarity of the example, although other more optimal definitions may not include it. From the point of view of the specification, this redundancy does not create a significant overhead given the referential nature of Object-Z variables.

Manager

$\upharpoonright(\mathit{RequestingBid}, \mathit{EvaluatingBid}, \mathit{AwardingContract}, \mathit{RejectingBid}, \mathit{AcceptingResults})$

Agent

ProposeToAdoptBidding

$\mathit{bidding}? : \mathit{Bidding}$
$\mathit{speechAct}! : \mathit{ToSpeak}$

$\mathit{speechAct}!.\mathit{speaker} = \mathit{bidding}?.\mathit{manager} = \mathit{self}\ \wedge$
$\mathit{speechAct}!.\mathit{addressee} = \mathit{bidding}?.\mathit{bidder}\ \wedge$
$\mathit{isProposeToAdoptBidding}(\mathit{bidding}?, \mathit{presently}) \in \mathit{speechAct}!.\mathit{points}\ \wedge$
$\mathit{isInformRequirements}(\mathit{bidding}?.\mathit{requirements}) \in \mathit{speechAct}!.\mathit{points}$

AcceptToDischargeBidding

$\mathit{bidding}? : \mathit{Bidding}$
$\mathit{speechAct}! : \mathit{ToSpeak}$

$\mathit{speechAct}!.\mathit{speaker} = \mathit{bidding}?.\mathit{manager} = \mathit{self}\ \wedge$
$\mathit{speechAct}!.\mathit{addressee} = \mathit{bidding}?.\mathit{bidder}\ \wedge$
$\mathit{isAcceptToDischargeBidding}(\mathit{bidding}?) \in \mathit{speechAct}!.\mathit{points}$

AcceptToAdoptEvaluating

$\mathit{evaluating}? : \mathit{EvaluatingBid}$
$\mathit{speechAct}! : \mathit{ToSpeak}$

$\mathit{speechAct}!.\mathit{speaker} = \mathit{evaluating}?.\mathit{manager} = \mathit{self}\ \wedge$
$\mathit{speechAct}!.\mathit{addressee} = \mathit{evaluating}?.\mathit{bidder}\ \wedge$
$\mathit{isAcceptToAdoptEvaluating}(\mathit{evaluating}?) \in \mathit{speechAct}!.\mathit{points}$

ProposeToDischargeEvaluating

$\mathit{evaluating}? : \mathit{EvaluatingBid}$
$\mathit{awarded}? : \mathit{EvaluationResult}$
$\mathit{speechAct}! : \mathit{ToSpeak}$

$\mathit{speechAct}!.\mathit{speaker} = \mathit{evaluating}?.\mathit{manager} = \mathit{self}\ \wedge$
$\mathit{speechAct}!.\mathit{addressee} = \mathit{evaluating}?.\mathit{bidder}\ \wedge$
$\mathit{isProposeToDischargeEvaluating}(\mathit{evaluating}?, \mathit{presently}) \in \mathit{speechAct}!.\mathit{points}\ \wedge$
$\mathit{isInformEvaluation}(\mathit{awarded}?) \in \mathit{speechAct}!.\mathit{points}$

Figure 18. Definition of the class *Manager* (part 1 of 2).

Evaluating a Bid

The operation *EvaluatingBid* defines the behaviour for accepting a bid for evaluation. In brief, this operation receives a *Bidding* action for discharge and an *EvaluatingBid* action for adoption, and evaluates whether or not these actions specify the same agent as their bidder, and whether or not the manager (i.e., the current instance) holds obligations to reply to a proposal to discharge the *Bidding* action and to a proposal to adopt the *EvaluatingBid* action (as specified by the axioms *existsReplyToProposeToDischargeBidding* and *existsReplyToProposeToAdoptEvaluating*, which are defined next). The fulfilment of these conditions leads to the operations *AcceptToDischargeBidding* and *AcceptToAdoptEvaluating* (which define speech acts for accepting to discharge bidding, and for accepting to adopt the evaluation of a bid, respectively) followed by the operation *SendUtterance*.

The axiom *existsReplyToProposeToDischargeBidding* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the manager is able to reply to a proposal to discharge a given *Bidding* action.

$$
\begin{array}{|l}
existsReplyToProposeToDischargeBidding : \mathbb{P}\ Obligation \times Bidding \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ bidding : Bidding \\
\bullet\ existsReplyToProposeToDischargeBidding(obligations, bidding) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval \mid \\
\qquad spoken.from \leq now \leq spoken.until\ \wedge \\
\qquad replied.from \leq now \leq replied.until \\
\quad \bullet\ \exists\ propose : \downarrow Propose \mid \\
\qquad propose = isProposeToDischargeBidding(bidding, replied) \\
\qquad \bullet\ \exists\ o : obligations \\
\qquad\quad \bullet\ \exists\ speaking : Speaking \mid \\
\qquad\qquad speaking.speaker = bidding.manager\ \wedge \\
\qquad\qquad speaking.addressee = bidding.bidder\ \wedge \\
\qquad\qquad speaking.time = spoken\ \wedge \\
\qquad\qquad speaking.points = isReplyTo(propose) \\
\qquad\quad \bullet\ speaking = o)
\end{array}
$$

$$\begin{array}{|l}
\hline \textit{ProposeToAdoptExecuting} \\
\textit{executing?} : \textit{ExecutingContract} \\
\textit{speechAct!} : \textit{ToSpeak} \\
\hline
\textit{speechAct!.speaker} = \textit{executing?.manager} = \textit{self} \land \\
\textit{speechAct!.addressee} = \textit{executing?.contractor} \land \\
\textit{isProposeToAdoptExecuting}(\textit{executing?}, \textit{presently}) \in \textit{speechAct!.points} \land \\
\textit{isInformContract}(\textit{executing?.contract}) \in \textit{speechAct!.points} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline \textit{AcceptToDischargeExecuting} \\
\textit{executing?} : \textit{ExecutingContract} \\
\textit{speechAct!} : \textit{ToSpeak} \\
\hline
\textit{speechAct!.speaker} = \textit{executing?.manager} = \textit{self} \land \\
\textit{speechAct!.addressee} = \textit{executing?.contractor} \land \\
\textit{isAcceptToDischargeExecuting}(\textit{executing?}) \in \textit{speechAct!.points} \\
\hline
\end{array}$$

$$\begin{array}{l}
\textit{RequestingBid} \mathrel{\widehat=} \\
\quad \textit{ProposeToAdoptBidding} \mathbin{⨾} \textit{SendUtterance} \\[1ex]
\textit{AwardingContract} \mathrel{\widehat=} [\, \textit{evaluating?} : \textit{EvaluatingBid};\ \textit{awarded} : \textit{EvaluationResult}; \\
\qquad\qquad\qquad\qquad \textit{executing?} : \textit{ExecutingContract} \mid \\
\quad \textit{awarded.true} \land \\
\quad \textit{evaluating?.bidder} = \textit{executing?.contractor} \land \\
\quad \textit{existsSpeakToProposeToDischargeEvaluating}(\operatorname{dom} \textit{obligations}, \textit{evaluating?}) \,] \bullet \\
\quad (\textit{ProposeToDischargeEvaluating} \land \\
\quad\ \textit{ProposeToAdoptExecuting}) \mathbin{⨾} \textit{SendUtterance} \\[1ex]
\textit{RejectingBid} \mathrel{\widehat=} [\, \textit{evaluating?} : \textit{EvaluatingBid};\ \textit{awarded} : \textit{EvaluationResult} \mid \\
\quad \neg\ \textit{awarded.true} \land \\
\quad \textit{existsSpeakToProposeToDischargeEvaluating}(\operatorname{dom} \textit{obligations}, \textit{evaluating?}) \,] \bullet \\
\quad \textit{ProposeToDischargeEvaluating} \mathbin{⨾} \textit{SendUtterance} \\[1ex]
\textit{AcceptingResults} \mathrel{\widehat=} [\, \textit{executing?} : \textit{ExecutingContract} \mid \\
\quad \textit{existsReplyToProposeToDischargeExecuting}(\operatorname{dom} \textit{obligations}, \textit{executing?}) \,] \bullet \\
\quad \textit{AcceptToDischargeExecuting} \mathbin{⨾} \textit{SendUtterance} \\[1ex]
\textit{EvaluatingBid} \mathrel{\widehat=} [\, \textit{bidding?} : \textit{Bidding};\ \textit{evaluating?} : \textit{EvaluatingBid} \mid \\
\quad \textit{bidding?.bidder} = \textit{evaluating?.bidder} \land \\
\quad \textit{existsReplyToProposeToDischargeBidding}(\operatorname{dom} \textit{obligations}, \textit{bidding?}) \land \\
\quad \textit{existsReplyToProposeToAdoptEvaluating}(\operatorname{dom} \textit{obligations}, \textit{evaluating?}) \,] \bullet \\
\quad (\textit{AcceptToDischargeBidding} \land \\
\quad\ \textit{AcceptToAdoptEvaluating}) \mathbin{⨾} \textit{SendUtterance}
\end{array}$$

Figure 19. Definition of the class *Manager* (part 2 of 2).

Likewise, the axiom *existsReplyToProposeToAdoptEvaluating* assesses whether or not a provided set of obligations contains a *Speaking* action in which the manager is able to reply to a proposal to adopt the given *EvaluatingBid* action.

$$
\begin{array}{l}
existsReplyToProposeToAdoptEvaluating : \\
\qquad\qquad\qquad \mathbb{P}\ Obligation \times EvaluatingBid \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ evaluating : EvaluatingBid \\
\bullet\ existsReplyToProposeToAdoptEvaluating(obligations, evaluating) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval\ | \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad\ \bullet\ \exists\ propose : \downarrow Propose\ | \\
\qquad\ propose = isProposeToAdoptEvaluating(evaluating, replied) \\
\qquad \bullet\ \exists\ o : obligations \\
\qquad\ \ \bullet\ \exists\ speaking : Speaking\ | \\
\qquad\qquad speaking.speaker = evaluating.manager \land \\
\qquad\qquad speaking.addressee = evaluating.bidder \land \\
\qquad\qquad speaking.time = spoken \land \\
\qquad\qquad speaking.points = isReplyTo(propose) \\
\qquad\quad \bullet\ speaking = o)
\end{array}
$$

Awarding a Contract

The operation *AwardingContract* defines the behaviour for awarding the execution of a contract. Besides defining an instance of type *EvaluationResult* that holds a true value, this operation receives an *EvaluatingBid* action for discharge and an *ExecutingContract* action for adoption, and evaluates whether or not these actions specify the same agent as their bidder and contractor (respectively), and whether or not the manager holds an obligation to propose discharging the *EvaluatingBid* action (as specified below by the axiom *existsSpeakToProposeToDischargeEvaluating*). The fulfilment of these conditions leads to the operations *ProposeToDischargeEvaluating* and *ProposeToAdoptExecuting* (which define speech acts for proposing to discharge the evaluation of a bid, and for proposing to adopt the execution of a contract, respectively) followed by the operation *SendUtterance*.

The axiom *existsSpeakToProposeToDischargeEvaluating* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the manager is able to reply to a proposal to discharge the given *EvaluatingBid* action.

$$
\begin{array}{l}
existsSpeakToProposeToDischargeEvaluating : \\
\qquad\qquad \mathbb{P}\ Obligation \times EvaluatingBid \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ evaluating : EvaluatingBid \\
\bullet\ existsSpeakToProposeToDischargeEvaluating(obligations, evaluating) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval \mid \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad\ \bullet\ \exists\ o : obligations \\
\qquad \bullet\ \exists\ speaking : Speaking \mid \\
\qquad\quad speaking.speaker = evaluating.manager \land \\
\qquad\quad speaking.addressee = evaluating.bidder \land \\
\qquad\quad speaking.time = spoken \land \\
\qquad\quad isProposeToDischargeEvaluating(evaluating, replied) \in speaking.points \\
\qquad\ \bullet\ speaking = o)
\end{array}
$$

Rejecting a Bid

The operation *RejectingBid* specifies the behaviour for rejecting a bid (which is the alternative outcome to awarding the execution of a contract). Besides defining an instance of type *EvaluationResult* that holds a false value, this operation receives an *EvaluatingBid* action for discharge, and evaluates whether or not the manager holds an obligation to propose discharging the action (as specified by the previously defined axiom *existsSpeakToProposeToDischargeEvaluating*). The fulfilment of these conditions leads to the sequential composition of the operations *ProposeToDischargeEvaluating* (which was also described earlier) and *SendUtterance*.

Accepting Results of a Contract

Lastly, the operation *AcceptingResults* specifies the behaviour for accepting a proposal to discharge the execution of a contract. This operation evaluates whether or not the manager holds an obligation in which he replies to a proposal to discharge the execution of a contract (as specified below by the axiom *existsReplyToProposeToDischargeExecuting*). If true, this condition leads to the operations *AcceptToDischargeExecuting* (which defines a speech act for accepting to discharge the execution of a contract) and *SendUtterance*.

The axiom *existsReplyToProposeToDischargeExecuting* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the manager is able to reply to a proposal to discharge the given *ExecutingContract* action.

$$
\begin{array}{l}
existsReplyToProposeToDischargeExecuting : \\
\qquad\qquad\qquad \mathbb{P}\ Obligation \times ExecutingContract \to \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ executing : ExecutingContract \\
\bullet\ existsReplyToProposeToDischargeExecuting(obligations, executing) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval \mid \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad\ \bullet\ \exists\ propose : \downarrow Propose \mid \\
\qquad\ propose = isProposeToDischargeExecuting(executing, replied) \\
\qquad \bullet\ \exists\ o : obligations \\
\qquad\ \ \bullet\ \exists\ speaking : Speaking \mid \\
\qquad\qquad speaking.speaker = executing.manager \land \\
\qquad\qquad speaking.addressee = executing.contractor \land \\
\qquad\qquad speaking.time = spoken \land \\
\qquad\qquad speaking.points = isReplyTo(propose) \\
\qquad\quad \bullet\ speaking = o)
\end{array}
$$

The next section describes the counterpart agent to a manager. This agent, which is called a contractor, is the agent with which managers interact to request bids and the execution of contracts.

4.7.2 Contractor

Contractors are those agents in the CNP that submit bids and execute contracts.

The class *Contractor* (which is shown in Figure 20, Figure 21 and Figure 22) is a subclass of *Agent* that defines operations for

- accepting and rejecting requests for bids,
- submitting bids for evaluation,

- accepting and rejecting the awarding of contracts, and
- submitting the results of executing contracts.

These operations are described below.

Accepting a Request for Bid

The operation *AcceptingToBid* specifies the behaviour for committing to submit a bid. This operation evaluates whether or not the bidder holds an obligation to reply to a proposal to submit a bid (as specified below by the axiom *existsReplyToProposeToAdoptBidding*). If true this condition leads to the operations *AcceptToAdoptBidding* (which defines a speech act for accepting to submit a bid) and *SendUtterance*.

The axiom *existsReplyToProposeToAdoptBidding* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the bidder is able to reply to a proposal to adopt the given *Bidding* action.

$$
\begin{array}{|l}
existsReplyToProposeToAdoptBidding : \mathbb{P}\ Obligation \times Bidding \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ bidding : Bidding \\
\bullet\ existsReplyToProposeToAdoptBidding(obligations, bidding) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval\ | \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad \bullet\ \exists\ propose : \downarrow Propose\ | \\
\qquad\ propose = isProposeToAdoptBidding(bidding, replied) \\
\qquad \bullet\ \exists\ o : obligations \\
\qquad\quad \bullet\ \exists\ speaking : Speaking\ | \\
\qquad\qquad speaking.speaker = bidding.bidder \land \\
\qquad\qquad speaking.addressee = bidding.manager \land \\
\qquad\qquad speaking.time = spoken \land \\
\qquad\qquad speaking.points = isReplyTo(propose) \\
\qquad\qquad \bullet\ speaking = o)
\end{array}
$$

Rejecting a Request for Bid

Along the same lines, the operation *RejectingToBid* specifies the behaviour of a bidder that rejects committing to submit a bid. This operation checks whether or not the bidder holds an obligation to reply to a proposal to submit a bid (as defined by the previously defined axiom *existsReplyToProposeToAdoptBidding*), which leads to the operations *RejectToAdoptBidding* (which defines a speech act rejecting to submit a bid) and *SendUtterance*.

Submitting a Bid for Evaluation

The operation *SubmittingBid* specifies the behaviour for submitting a bid for evaluation. This operation receives a *Bidding* action for discharge and an *EvaluatingBid* action for adoption, and evaluates whether or not these actions specify the same agent as their manager, and whether or not the bidder (i.e., the current bidder instance) holds an obligation to propose discharging the *Bidding* action (as specified below by the axiom *existsSpeakToProposeToDischargeBidding*). The fulfilment of these conditions leads to the conjunctive composition of the operations *ProposeToDischargeBidding* and *ProposeToAdoptEvaluating* (which define speech acts for proposing to discharge the submission of a bid, and for proposing to adopt the execution of a contract, respectively) followed by the operation *SendUtterance*.

The axiom *existsSpeakToProposeToDischargeBidding* is a function that assesses whether or not a set of obligations contains a *Speaking* action in which the bidder is able to propose to discharge a given *Bidding* action.

$$
\begin{array}{|l}
existsSpeakToProposeToDischargeBidding : \mathbb{P}\,Obligation \times Bidding \rightarrow \mathbb{B} \\
\hline
\forall\, obligations : \mathbb{P}\,Obligation;\ bidding : Bidding \\
\bullet\ existsSpeakToProposeToDischargeBidding(obligations, bidding) \Leftrightarrow \\
\quad (\exists\, spoken, replied : Interval \mid \\
\quad\quad spoken.from \leq now \leq spoken.until \wedge \\
\quad\quad replied.from \leq now \leq replied.until \\
\quad\ \bullet\ \exists\, o : obligations \\
\quad\quad \bullet\ \exists\, speaking : Speaking \mid \\
\quad\quad\quad speaking.speaker = bidding.bidder \wedge \\
\quad\quad\quad speaking.addressee = bidding.manager \wedge \\
\quad\quad\quad speaking.time = spoken \wedge \\
\quad\quad\quad isProposeToDischargeBidding(bidding, replied) \in speaking.points \\
\quad\quad\ \bullet\ speaking = o)
\end{array}
$$

Accepting the Rejection of a Bid

The operation *AcceptingRejectionOfBid* specifies the behaviour for accepting that a bid was not awarded for execution. This operation receives an *EvaluatingBid* action and evaluates whether or not the bidder holds an obligation to reply to a proposal to discharge this action. In addition, this operation defines an *ExecutingContract* action that is compatible to the *EvaluatingBid* action (i.e., the executing action covers all possible contracts that could result from the evaluating action), and evaluates whether or not the bidder holds a proposal to adopt the *ExecutingContract* action (as specified below by the axioms *existsReplyTo-ProposeToDischargeEvaluating* and *existsReplyToProposeToAdoptExecuting*).

The fulfilment of these conditions leads to the composition of the operations *AcceptToDischargeEvaluating* (which defines a speech act accepting to discharge the evaluation of a bid) and *SendUtterance*.

The axiom *existsReplyToProposeToDischargeEvaluating* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the bidder is able to reply to a proposal to discharge a given *EvaluatingBid* action.

$$
\begin{array}{l}
existsReplyToProposeToDischargeEvaluating : \\
\qquad\qquad\qquad \mathbb{P}\ Obligation \times EvaluatingBid \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ evaluating : EvaluatingBid \\
\bullet\ existsReplyToProposeToDischargeEvaluating(obligations, evaluating) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval \mid \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad\ \bullet\ \exists\ propose : \downarrow Propose \mid \\
\qquad\ propose = isProposeToDischargeEvaluating(evaluating, replied) \\
\qquad \bullet\ \exists\ o : obligations \\
\qquad\ \ \bullet\ \exists\ speaking : Speaking \mid \\
\qquad\qquad speaking.speaker = evaluating.bidder \land \\
\qquad\qquad speaking.addressee = evaluating.manager \land \\
\qquad\qquad speaking.time = spoken \land \\
\qquad\qquad speaking.points = isReplyTo(propose) \\
\qquad\quad \bullet\ speaking = o)
\end{array}
$$

Contractor

$\upharpoonright(RejectingToBid, AcceptingToBid, SubmittingBid, AcceptingRejectionOfBid,$
$AcceptingAward, RejectingAward, SubmittingResults)$

Agent

AcceptToAdoptBidding

$bidding? : Bidding$
$speechAct! : ToSpeak$

$speechAct!.speaker = bidding?.bidder = self \land$
$speechAct!.addressee = bidding?.manager \land$
$isAcceptToAdoptBidding(bidding?) \in speechAct!.points$

RejectToAdoptBidding

$bidding? : Bidding$
$speechAct! : ToSpeak$

$speechAct!.speaker = bidding?.bidder = self \land$
$speechAct!.addressee = bidding?.manager \land$
$isRejectToAdoptBidding(bidding?) \in speechAct!.points$

ProposeToDischargeBidding

$bidding? : Bidding$
$speechAct! : ToSpeak$

$speechAct!.speaker = bidding?.bidder = self \land$
$speechAct!.addressee = bidding?.manager \land$
$isProposeToDischargeBidding(bidding?, presently) \in speechAct!.points$

ProposeToAdoptEvaluating

$evaluating? : EvaluatingBid$
$speechAct! : ToSpeak$

$speechAct!.speaker = evaluating?.bidder = self \land$
$speechAct!.addressee = evaluating?.manager \land$
$isProposeToAdoptEvaluating(evaluating?, presently) \in speechAct!.points \land$
$isInformBid(evaluating?.bid) \in speechAct!.points$

Figure 20. Definition of the class *Contractor* (part 1 of 3).

The axiom *existsReplyToProposeToAdoptExecuting* assesses whether or not a provided set of obligations contains a *Speaking* action in which the bidder is able to reply to a proposal to adopt a given *ExecutingBid* action.

$$
\begin{array}{|l}
existsReplyToProposeToAdoptExecuting : \\
\qquad\qquad\qquad \mathbb{P}\, Obligation \times ExecutingContract \rightarrow \mathbb{B} \\
\hline
\forall\, obligations : \mathbb{P}\, Obligation;\ executing : ExecutingContract \\
\bullet\ existsReplyToProposeToAdoptExecuting(obligations, executing) \Leftrightarrow \\
\quad (\exists\, spoken, replied : Interval \mid \\
\qquad spoken.from \leq now \leq spoken.until \wedge \\
\qquad replied.from \leq now \leq replied.until \\
\quad \bullet\ \exists\, propose : \downarrow Propose \mid \\
\qquad propose = isProposeToAdoptExecuting(executing, replied) \\
\qquad \bullet\ \exists\, o : obligations \\
\qquad\quad \bullet\ \exists\, speaking : Speaking \mid \\
\qquad\qquad speaking.speaker = executing.contractor \wedge \\
\qquad\qquad speaking.addressee = executing.manager \wedge \\
\qquad\qquad speaking.time = spoken \wedge \\
\qquad\qquad speaking.points = isReplyTo(propose) \\
\qquad\quad \bullet\ speaking = o)
\end{array}
$$

Accepting an Awarded Contract

The operation *AcceptingAward* specifies the behaviour for accepting to execute a contract. This operation receives an *EvaluatingBid* action for discharge and an *ExecutingContract* action for adoption, and evaluates whether or not these actions specify the same agent as their manager, and whether or not the bidder holds obligations to reply to a proposal to discharge the *EvaluatingBid* action, and to reply to a proposal to adopt the *ExecutingContract* action (as indicated by the previously defined axioms *existsReplyToProposeToDischargeEvaluating* and *existsReplyToProposeToAdoptExecuting*). The fulfilment of these conditions leads to the operations *AcceptToDischargeEvaluating* (which was also described earlier) and *AcceptToAdoptExecuting* (which defines a speech act accepting to adopt the execution of a bid) followed by the operation *SendUtterance*.

AcceptToDischargeEvaluating

$$\begin{array}{l}
evaluating? : EvaluatingBid \\
speechAct! : ToSpeak \\
\hline
speechAct!.speaker = evaluating?.bidder = self \land \\
speechAct!.addressee = evaluating?.manager \land \\
isAcceptToDischargeEvaluating(evaluating?) \in speechAct!.points
\end{array}$$

AcceptToAdoptExecuting

$$\begin{array}{l}
executing? : ExecutingContract \\
speechAct! : ToSpeak \\
\hline
speechAct!.speaker = executing?.contractor = self \land \\
speechAct!.addressee = executing?.manager \land \\
isAcceptToAdoptExecuting(executing?) \in speechAct!.points
\end{array}$$

RejectToAdoptExecuting

$$\begin{array}{l}
executing? : ExecutingContract \\
speechAct! : ToSpeak \\
\hline
speechAct!.speaker = executing?.contractor = self \land \\
speechAct!.addressee = executing?.manager \land \\
isRejectToAdoptExecuting(executing?) \in speechAct!.points
\end{array}$$

ProposeToDischargeExecuting

$$\begin{array}{l}
executing? : ExecutingContract \\
speechAct! : ToSpeak \\
results : \mathbb{P}\, ResultsItem \\
\hline
\forall c : executing?.contract \\
\bullet\ \exists r : results \bullet r.action = c.action \land r.constraints = c.constraints \\
\#results = \#executing?.contract \\
speechAct!.speaker = executing?.contractor = self \land \\
speechAct!.addressee = executing?.manager \land \\
isProposeToDischargeExecuting(executing?, presently) \in speechAct!.points \land \\
isInformResults(results) \in speechAct!.points
\end{array}$$

Figure 21. Definition of the class *Contractor* (part 2 of 3).

Rejecting an Awarded Contract

Along the same lines, the operation *RejectingAward* specifies the behaviour of a bidder that rejects to adopt committing to execute a contract. This operation receives an *EvaluatingBid* action for discharge and an *ExecutingContract* action for adoption, and evaluates whether or not these actions specify the same agent as their manager, and whether or not the bidder holds obligations to reply to a proposal to discharge the *EvaluatingBid* action, and to reply to a proposal to adopt the *ExecutingContract* action (as specified by the previously defined axioms *existsReplyToProposeToDischargeEvaluating* and *existsReplyToProposeToAdopt-Executing*). If true, these conditions lead to the composition of the operations *AcceptToDischargeEvaluating* (which was also described above) and *RejectToAdopt-Executing* (which defines a speech act rejecting to adopt executing a contract.) followed by the operation *SendUtterance*.

Submitting Results of Executing a Contract

The operation *SubmittingResults* specifies the behaviour for proposing to discharge being committed to the execution of a contract and for sending the results of its execution. This operation receives an *ExecutingContract* action for discharge, and evaluates whether or not the bidder holds an obligation to propose to discharge this action (as specified below by the axiom *existsSpeakToProposeToDischargeExecuting*). If true, this condition leads to the operations *ProposeToDischargeExecuting* (which defines a speech act that proposes to discharge the execution of the contract and communicates the results of its execution) and *SendUtterance*.

The axiom *existsSpeakToProposeToDischargeExecuting* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the bidder can propose to discharge the *ExecutingBid* action.

$$\begin{aligned}
&AcceptingToBid \mathrel{\widehat{=}} [\, bidding? : Bidding \mid \\
&\quad existsReplyToProposeToAdoptBidding(\operatorname{dom} obligations, bidding?) \,] \bullet \\
&\quad AcceptToAdoptBidding \fatsemi SendUtterance
\end{aligned}$$

$$\begin{aligned}
&RejectingToBid \mathrel{\widehat{=}} [\, bidding? : Bidding \mid \\
&\quad existsReplyToProposeToAdoptBidding(\operatorname{dom} obligations, bidding?) \,] \bullet \\
&\quad RejectToAdoptBidding \fatsemi SendUtterance
\end{aligned}$$

$$\begin{aligned}
&SubmittingBid \mathrel{\widehat{=}} [\, bidding? : Bidding;\ evaluating? : EvaluatingBid \mid \\
&\quad bidding?.manager = evaluating?.manager \land \\
&\quad existsSpeakToProposeToDischargeBidding(\operatorname{dom} obligations, bidding?) \,] \bullet \\
&\quad (ProposeToDischargeBidding \land \\
&\quad\ ProposeToAdoptEvaluating) \fatsemi SendUtterance
\end{aligned}$$

$$\begin{aligned}
&AcceptingRejectionOfBid \mathrel{\widehat{=}} [\, evaluating? : EvaluatingBid; \\
&\qquad\qquad\qquad\qquad\qquad\quad executing : ExecutingContract \mid \\
&\quad executing.manager = evaluating?.manager \land \\
&\quad executing.contract \subseteq evaluating?.bid \land \\
&\quad existsReplyToProposeToDischargeEvaluating(\operatorname{dom} obligations, evaluating?) \land \\
&\quad \neg\ existsReplyToProposeToAdoptExecuting(\operatorname{dom} obligations, executing) \,] \bullet \\
&\quad AcceptToDischargeEvaluating \fatsemi SendUtterance
\end{aligned}$$

$$\begin{aligned}
&AcceptingAward \mathrel{\widehat{=}} [\, evaluating? : EvaluatingBid;\ executing? : ExecutingContract \mid \\
&\quad evaluating?.manager = executing?.manager \land \\
&\quad existsReplyToProposeToDischargeEvaluating(\operatorname{dom} obligations, evaluating?) \land \\
&\quad existsReplyToProposeToAdoptExecuting(\operatorname{dom} obligations, executing?) \,] \bullet \\
&\quad (AcceptToDischargeEvaluating \land \\
&\quad\ AcceptToAdoptExecuting) \fatsemi SendUtterance
\end{aligned}$$

$$\begin{aligned}
&RejectingAward \mathrel{\widehat{=}} [\, evaluating? : EvaluatingBid;\ executing? : ExecutingContract \mid \\
&\quad evaluating?.manager = executing?.manager \land \\
&\quad existsReplyToProposeToDischargeEvaluating(\operatorname{dom} obligations, evaluating?) \land \\
&\quad existsReplyToProposeToAdoptExecuting(\operatorname{dom} obligations, executing?) \,] \bullet \\
&\quad (AcceptToDischargeEvaluating \land RejectToAdoptExecuting) \fatsemi SendUtterance
\end{aligned}$$

$$\begin{aligned}
&SubmittingResults \mathrel{\widehat{=}} [\, executing? : ExecutingContract \mid \\
&\quad existsSpeakToProposeToDischargeExecuting(\operatorname{dom} obligations, executing?) \,] \bullet \\
&\quad ProposeToDischargeExecuting \fatsemi SendUtterance
\end{aligned}$$

Figure 22. Definition of the class *Contractor* (part 3 of 3).

$$
\begin{array}{|l}
existsSpeakToProposeToDischargeExecuting : \\
\qquad\qquad\qquad \mathbb{P}\ Obligation \times ExecutingContract \rightarrow \mathbb{B} \\
\hline
\forall\, obligations : \mathbb{P}\ Obligation;\ executing : ExecutingContract \\
\bullet\ existsSpeakToProposeToDischargeExecuting(obligations, executing) \Leftrightarrow \\
\quad (\exists\, spoken, replied : Interval \mid \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad\ \bullet\ \exists\, o : obligations \\
\qquad \bullet\ \exists\, speaking : Speaking \mid \\
\qquad\quad speaking.speaker = executing.contractor \land \\
\qquad\quad speaking.addressee = executing.manager \land \\
\qquad\quad speaking.time = spoken \land \\
\qquad\quad isProposeToDischargeExecuting(executing, replied) \in speaking.points \\
\qquad\ \bullet\ speaking = o)
\end{array}
$$

This section presented the communicational operations that agents in the roles of managers and contractors can perform (as long as their committal preconditions are met). However, these operations are disembodied of any concrete interaction. The next section will show how these operations are assembled into structured contract net interactions.

4.8 The Contract Net Protocol as a Joint Activity

The class *ContractNet* (which is shown in Figure 23) is a subclass of *JointActivity* that specifies the interactions that can occur in the CNP. This class defines two participants (a *manager* and a *contractor*) and three actions in which they participate (*bidding*, *evaluating* and *executing*). This class also defines that the actions specified in the *bid* are a subset of those in the *requirements*, and that the actions in the *contract* are a subset of those in the *bid*.

$$
\begin{array}{l}
\textit{ContractNet} \\
\hline
\textit{JointActivity} \\
\begin{array}{|l}
\hline
\textit{manager} : \downarrow \textit{Manager} \\
\textit{contractor} : \downarrow \textit{Contractor} \\
\textit{bidding} : \textit{Bidding} \\
\textit{evaluating} : \textit{EvaluatingBid} \\
\textit{executing} : \textit{ExecutingContract} \\
\hline
\textit{manager} = \textit{bidding.manager} = \textit{evaluating.manager} = \textit{executing.manager} \land \\
\textit{contractor} = \textit{bidding.bidder} = \textit{evaluating.bidder} = \textit{executing.contractor} \land \\
\textit{actions} = \{\textit{bidding}, \textit{evaluating}, \textit{executing}\} \\
\forall \textit{bid} : \textit{evaluating.bid} \\
\bullet \exists \textit{req} : \textit{bidding.requirements} \\
\quad \bullet \textit{req.action} = \textit{bid.action} \\
\forall \textit{contract} : \textit{executing.contract} \\
\bullet \exists \textit{bid} : \textit{evaluating.bid} \\
\quad \bullet \textit{bid.action} = \textit{contract.action} \\
\hline
\end{array} \\
\textit{Interaction} \mathrel{\widehat{=}} \left[\begin{array}{l} \textit{manager?} : \downarrow \textit{Manager};\ \textit{contractor?} : \downarrow \textit{Contractor} \;\middle|\; \\ \textit{manager?} = \textit{manager} \land \textit{contractor?} = \textit{contractor} \end{array}\right] \bullet \\
\textit{manager.RequestingBid} \mathbin{⨾} \\
(\textit{contractor.RejectingToBid} \\
\quad \Box \\
\ (\textit{contractor.AcceptingToBid} \mathbin{⨾} \\
\quad \textit{contractor.SubmittingBid} \mathbin{⨾} \textit{manager.EvaluatingBid} \mathbin{⨾} \\
\quad ((\textit{manager.RejectingBid} \mathbin{⨾} \textit{contractor.AcceptingRejectionOfBid}) \\
\qquad \Box \\
\quad\ (\textit{manager.AwardingContract} \mathbin{⨾} \\
\qquad (\textit{contractor.RejectingAward} \\
\qquad\quad \Box \\
\qquad\ ((\textit{contractor.AcceptingAward} \mathbin{⨾} \textit{contractor.SubmittingResults}) \mathbin{⨾} \\
\qquad\quad \textit{manager.AcceptingResults}))))) \\
\hline
\end{array}
$$

Figure 23. Definition of the Contract Net Protocol as a joint activity.

4.8.1 Interactions

The operation *Interaction* defines the sequences of interdependent agent operations making the allowed interactions for the activity. This operation (which is illustrated as a conversation protocol in Figure 24) specifies that a request for bid from a manager is

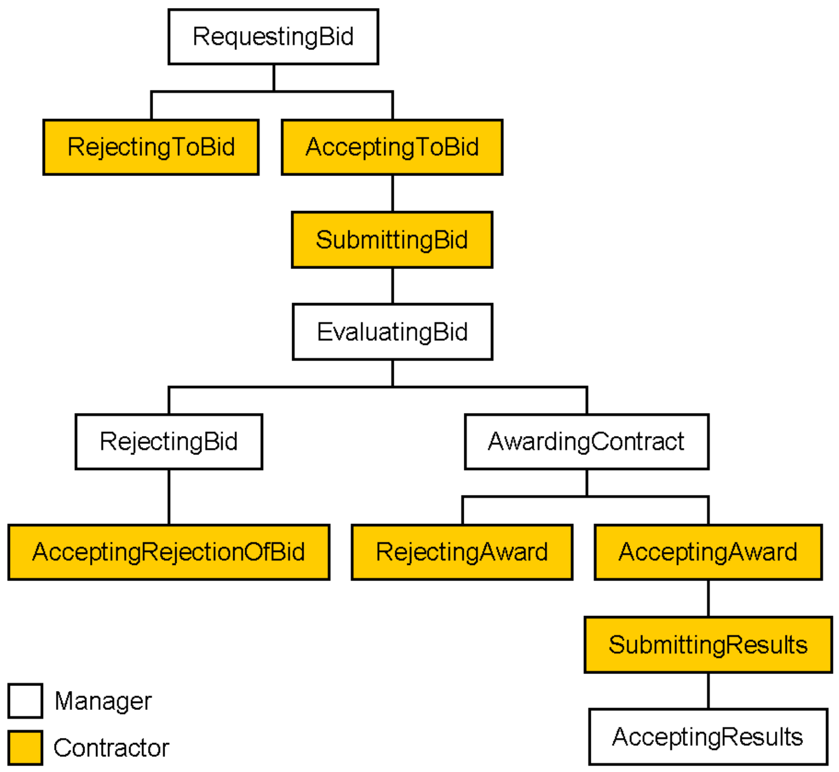


Figure 24. Protocol with the interactions in the joint activity *ContractNet*.

followed either by a rejection or an acceptance from the bidder.[34] In the case of the rejection, no other participation follows, thus signalling the end of the interaction. On the other hand, the bidder's acceptance to bid is followed by a submission of a bid, and the manager's acceptance to evaluate it. At this point, the manager either rejects the bid (which if accepted by the contractor leads to the end of the interaction) or awards it as contract. In the case of being awarded the contract, the contractor either rejects the awarding (ending the interaction) or accepts the awarding. Lastly, this acceptance is followed by the submission of results and the manager's acceptance of those results.

It is worth noticing that this interaction specification would not be any different than *ad hoc* conversation protocols if it was not supported by strict compositional principles. To

[34] Although it is not explicitly modelled, we assume that counterproposals are followed by a rejection.

support this claim, section *4.11:Example Proof* shows a proof on a segment of this interaction specification.

4.9 Contract Net Society

The class *ContractNetSociety* is specified as a subclass of *PFPsociety* that defines *ContractNet* as a joint activity where a manager requests to one or more bidders the submission of a bid that abides by the same requirements.[35]

$$
\begin{array}{l}
\textbf{ContractNetSociety} \\
\hline
PFPsociety \\
ContractNet \mathrel{\widehat=} [\, contractNet : \mathbb{P}_1\ ContractNet \mid \\
\quad contractNet \subseteq activities \land \\
\quad (\exists\, manager : \downarrow Manager;\ requirements : \mathbb{P}_1\ Requirement \\
\quad\ \bullet \forall\, cn : contractNet \\
\quad\quad \bullet cn.manager = manager \land \\
\quad\quad\ \ cn.bidding.requirements = requirements) \,] \bullet \\
\quad \bigwedge cn : contractNet \bullet cn.Interaction \\
\hline
\end{array}
$$

4.10 Example Conversation: Executing a Contract

Figure 25 shows a UML interaction diagram for an interaction in the contract net activity. Specifically, it shows a conversation that begins with a request for bid and advances until the contract is executed and its results submitted. This conversation is specified by the sequence of the operations *RequestingBid*, *AcceptingToBid*, *SubmittingBid*, *EvaluatingBid*, *AwardingContract*, *AcceptingAward*, *SubmittingResults* and *AcceptingResults*. Figure 26 and Figure 27 show the state of shared social commitments and obligations on the manager and the contractor as this conversation evolves.

[35] This definition was simplified to allow for variations of the CNP, such as those in which various contractors are awarded the execution of actions (Smith, 1980).

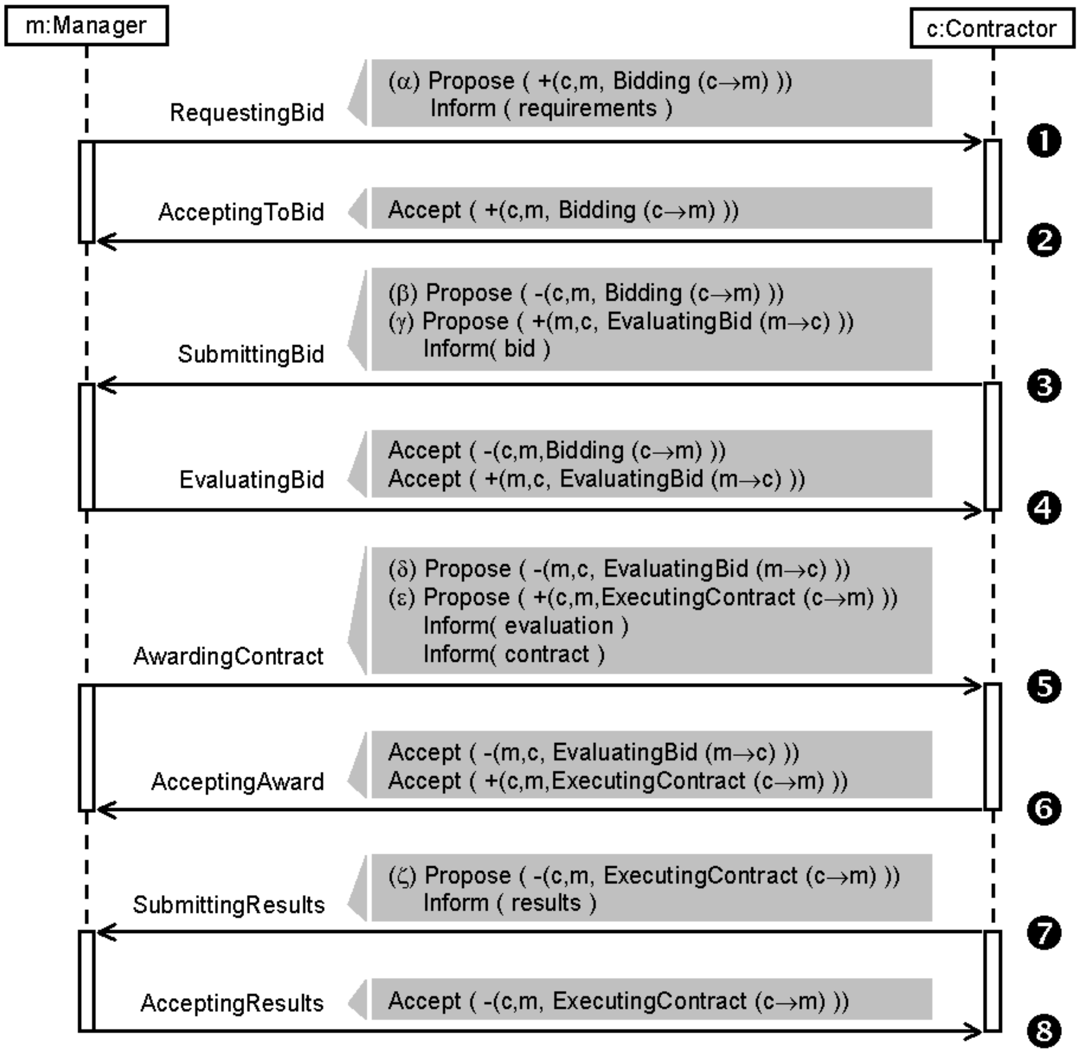

m:Manager
c:Contractor
RequestingBid
(α) Propose (+(c,m, Bidding (c→m)))
Inform (requirements)
❶
AcceptingToBid
Accept (+(c,m, Bidding (c→m)))
❷
SubmittingBid
(β) Propose (-(c,m, Bidding (c→m)))
(γ) Propose (+(m,c, EvaluatingBid (m→c)))
Inform(bid)
❸
EvaluatingBid
Accept (-(c,m,Bidding (c→m)))
Accept (+(m,c, EvaluatingBid (m→c)))
❹
AwardingContract
(δ) Propose (-(m,c, EvaluatingBid (m→c)))
(ε) Propose (+(c,m,ExecutingContract (c→m)))
Inform(evaluation)
Inform(contract)
❺
AcceptingAward
Accept (-(m,c, EvaluatingBid (m→c)))
Accept (+(c,m,ExecutingContract (c→m)))
❻
SubmittingResults
(ζ) Propose (-(c,m, ExecutingContract (c→m)))
Inform (results)
❼
AcceptingResults
Accept (-(c,m, ExecutingContract (c→m)))
❽

Figure 25. UML interaction diagram for a Contract Net conversation.

4.10.1 Requesting a Bid

As shown in Figure 25, the interaction begins with an utterance from the manager (identified as *m*) to a contractor (identified as *c*) in which he requests that she submit a bid based on the given requirements.

As specified in *m*'s operation *RequestingBid*, this speech act contains a *Propose* illocutionary point (labelled α), proposing the adoption of a shared social commitment in which *c* is responsible to *m* for an action *Bidding* in which *c* performs the action and informs the results of the action to *m* (as before, the representation used in this figure has been simplified for clarity). As shown in Figure 26, the uttering of this proposal triggers the following conversational policy:

- Policy 1 (the uttering of a proposal commits the addressee to reply to the proposal): the uttering of proposal α results in the adoption of obligations in which *c* replies to *m*'s proposal α (added as obligations 1 and 2 in Figure 26 on both the manager and the contractor).[36]

4.10.2 Accepting to Bid

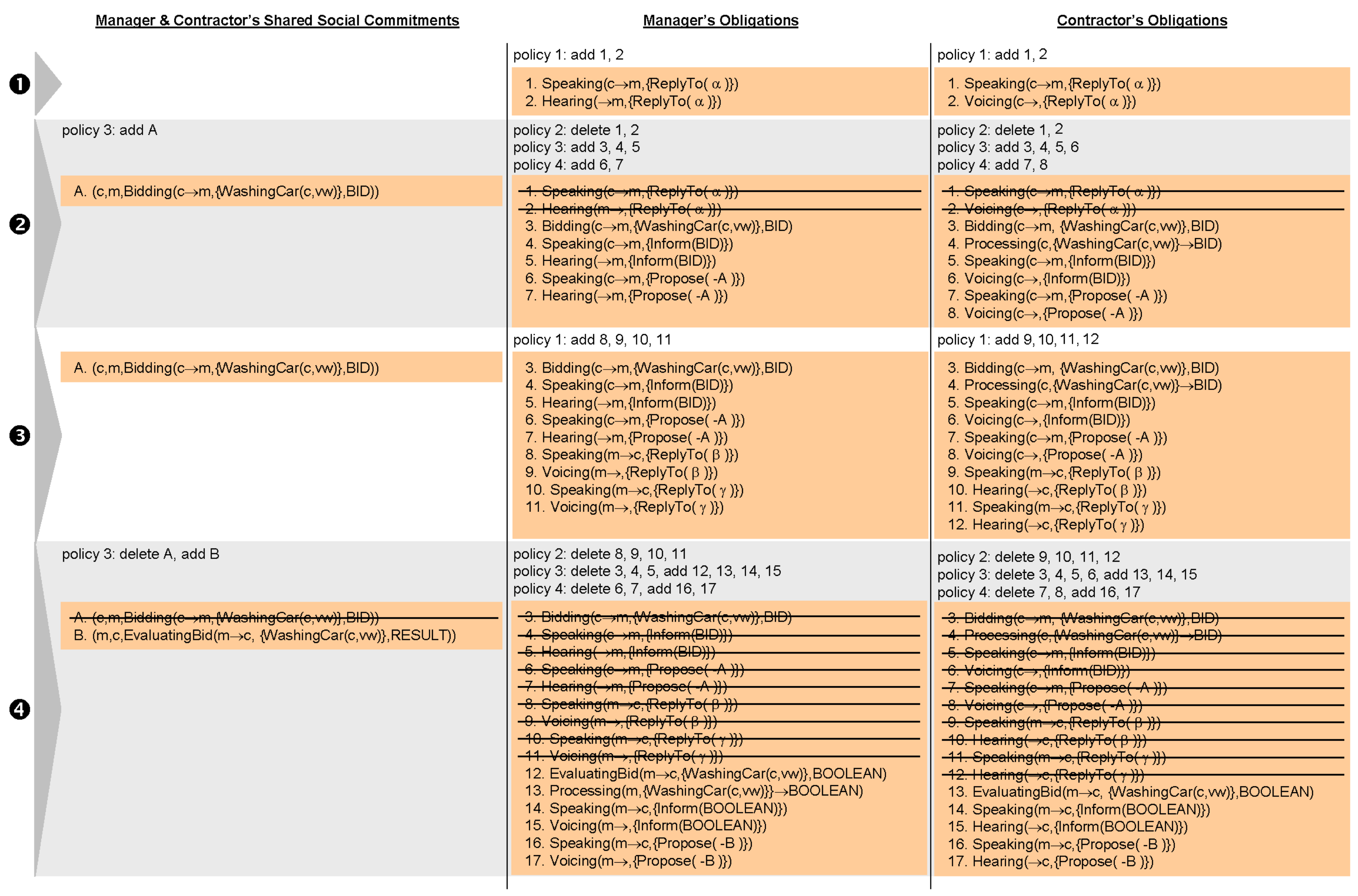
The next interaction (labelled as interaction 2 in Figure 25) specifies the execution of *c*'s operation *AcceptingToBid*, in which she accepts committing to submit a bid (only if there is an obligation to reply to a request for bid—which exists as obligations 1 and 2). Uttering this acceptance results in the application of the following policies:

- Policy 2 (replying to a proposal discharges the obligation to reply): the acceptance to uptake the operation proposed in α discharges the obligation to reply to α (which deletes obligations 1 and 2 in Figure 26 on both the manager and the contractor).

- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in α causes the adoption of the proposed commitment, in this case to submit a bid (added as commitment A in Figure 26). In addition, this acceptance results in the adoption of obligations to perform the joint action. As such, the contractor adopts obligations to produce and communicate a bid (obligations 3 to 6), and the manager adopts obligations to receive it (obligations 3 to 5).[37]

[36] In the case of the contractor the acquired obligation is for voicing a reply, and in the case of the manager for hearing it.

[37] In this example, the action requested is that of the contractor washing a VW car.



	Manager & Contractor's Shared Social Commitments	Manager's Obligations	Contractor's Obligations
❶		policy 1: add 1, 2 1. Speaking(c→m,{ReplyTo(α)}) 2. Hearing(→m,{ReplyTo(α)})	policy 1: add 1, 2 1. Speaking(c→m,{ReplyTo(α)}) 2. Voicing(c→,{ReplyTo(α)})
❷	policy 3: add A A. (c,m,Bidding(c→m,{WashingCar(c,vw)},BID))	policy 2: delete 1, 2 policy 3: add 3, 4, 5 policy 4: add 6, 7 ~~1. Speaking(c→m,{ReplyTo(α)})~~ ~~2. Hearing(m→,{ReplyTo(α)})~~ 3. Bidding(c→m,{WashingCar(c,vw)},BID) 4. Speaking(c→m,{Inform(BID)}) 5. Hearing(→m,{Inform(BID)}) 6. Speaking(c→m,{Propose(-A)}) 7. Hearing(→m,{Propose(-A)})	policy 2: delete 1, 2 policy 3: add 3, 4, 5, 6 policy 4: add 7, 8 ~~1. Speaking(c→m,{ReplyTo(α)})~~ ~~2. Voicing(c→,{ReplyTo(α)})~~ 3. Bidding(c→m, {WashingCar(c,vw)},BID) 4. Processing(c,{WashingCar(c,vw)}→BID) 5. Speaking(c→m,{Inform(BID)}) 6. Voicing(c→,{Inform(BID)}) 7. Speaking(c→m,{Propose(-A)}) 8. Voicing(c→,{Propose(-A)})
❸	A. (c,m,Bidding(c→m,{WashingCar(c,vw)},BID))	policy 1: add 8, 9, 10, 11 3. Bidding(c→m,{WashingCar(c,vw)},BID) 4. Speaking(c→m,{Inform(BID)}) 5. Hearing(→m,{Inform(BID)}) 6. Speaking(c→m,{Propose(-A)}) 7. Hearing(→m,{Propose(-A)}) 8. Speaking(m→c,{ReplyTo(β)}) 9. Voicing(m→,{ReplyTo(β)}) 10. Speaking(m→c,{ReplyTo(γ)}) 11. Voicing(m→,{ReplyTo(γ)})	policy 1: add 9, 10, 11, 12 3. Bidding(c→m, {WashingCar(c,vw)},BID) 4. Processing(c,{WashingCar(c,vw)}→BID) 5. Speaking(c→m,{Inform(BID)}) 6. Voicing(c→,{Inform(BID)}) 7. Speaking(c→m,{Propose(-A)}) 8. Voicing(c→,{Propose(-A)}) 9. Speaking(m→c,{ReplyTo(β)}) 10. Hearing(→c,{ReplyTo(β)}) 11. Speaking(m→c,{ReplyTo(γ)}) 12. Hearing(→c,{ReplyTo(γ)})
❹	policy 3: delete A, add B ~~A. (c,m,Bidding(c→m,{WashingCar(c,vw)},BID))~~ B. (m,c,EvaluatingBid(m→c, {WashingCar(c,vw)},RESULT))	policy 2: delete 8, 9, 10, 11 policy 3: delete 3, 4, 5, add 12, 13, 14, 15 policy 4: delete 6, 7, add 16, 17 ~~3. Bidding(c→m,{WashingCar(c,vw)},BID)~~ ~~4. Speaking(c→m,{Inform(BID)})~~ ~~5. Hearing(→m,{Inform(BID)})~~ ~~6. Speaking(c→m,{Propose(-A)})~~ ~~7. Hearing(→m,{Propose(-A)})~~ ~~8. Speaking(m→c,{ReplyTo(β)})~~ ~~9. Voicing(m→,{ReplyTo(β)})~~ ~~10. Speaking(m→c,{ReplyTo(γ)})~~ ~~11. Voicing(m→,{ReplyTo(γ)})~~ 12. EvaluatingBid(m→c,{WashingCar(c,vw)},BOOLEAN) 13. Processing(m,{WashingCar(c,vw)}}→BOOLEAN) 14. Speaking(m→c,{Inform(BOOLEAN)}) 15. Voicing(m→,{Inform(BOOLEAN)}) 16. Speaking(m→c,{Propose(-B)}) 17. Voicing(m→,{Propose(-B)})	policy 2: delete 9, 10, 11, 12 policy 3: delete 3, 4, 5, 6, add 13, 14, 15 policy 4: delete 7, 8, add 16, 17 ~~3. Bidding(c→m, {WashingCar(c,vw)},BID)~~ ~~4. Processing(c,{WashingCar(c,vw)}→BID)~~ ~~5. Speaking(c→m,{Inform(BID)})~~ ~~6. Voicing(c→,{Inform(BID)})~~ ~~7. Speaking(c→m,{Propose(-A)})~~ ~~8. Voicing(c→,{Propose(-A)})~~ ~~9. Speaking(m→c,{ReplyTo(β)})~~ ~~10. Hearing(→c,{ReplyTo(β)})~~ ~~11. Speaking(m→c,{ReplyTo(γ)})~~ ~~12. Hearing(→c,{ReplyTo(γ)})~~ 13. EvaluatingBid(m→c, {WashingCar(c,vw)},BOOLEAN) 14. Speaking(m→c,{Inform(BOOLEAN)}) 15. Hearing(→c,{Inform(BOOLEAN)}) 16. Speaking(m→c,{Propose(-B)}) 17. Hearing(→c,{Propose(-B)})

Figure 26. State of shared social commitments and obligations of the manager and contractor in the Contract Net conversation example (part 1 of 2).

- Policy 4 (accepting a *ProposingToDischarge* action obligates the *discharger* to propose its discharge): the acceptance to adopt the action *Bidding* (which is a subtype of *ProposingToDischarge*) results in the adoption of obligations in which *c* (the *discharger*) is to propose to *m* (the *discharged*) discharging the action. These obligations are added as obligations 6 and 7 on the manager, and 7 and 8 on the contractor.

4.10.3 Submitting a Bid

The next interaction (labelled as interaction 3 in Figure 25) specifies the execution of *c*'s operation *SubmittingBid*, in which a) she proposes to discharge the commitment to submit a bid (only if there is an obligation in which *c* proposes to discharge the commitment that she submit a bid—which exists as obligations 7 and 8); b) she informs a bid; and c) she proposes to adopt a commitment in which *m* evaluates this bid. The uttering of these proposals (which are labelled β and γ) triggers the following conversational policies:

- Policy 1 (the uttering of a proposal commits the addressee to reply to the proposal): the uttering of proposal β results in the adoption of obligations in which *m* replies to β (added as obligations 8 and 9 on the manager, and 9 and 10 on the contractor), and

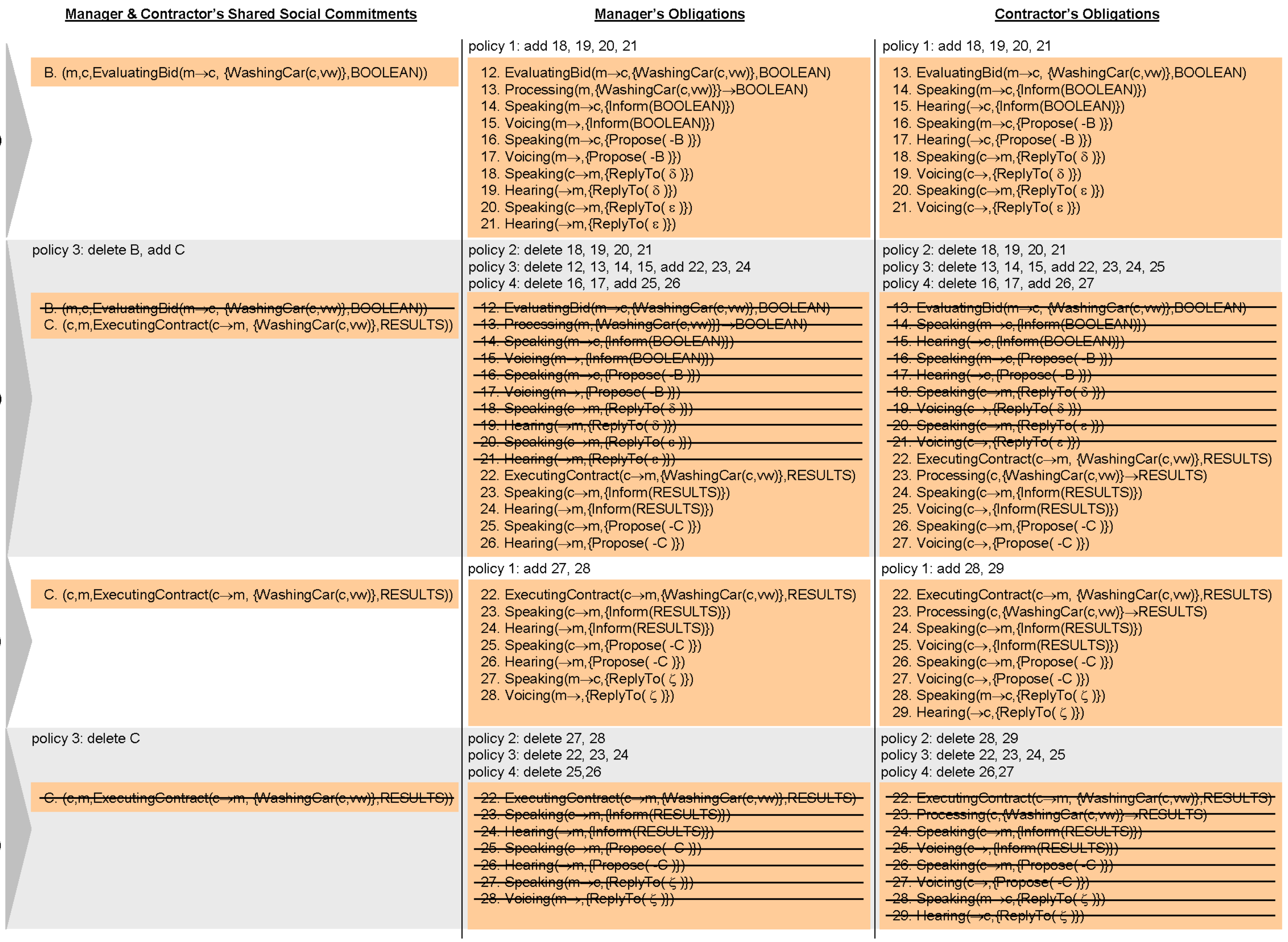
- Policy 1 (*ditto*): the uttering of proposal γ results in the adoption of obligations in which *m* replies to γ (added as obligations 10 and 11 on the manager, and 11 and 12 on the contractor).

4.10.4 Accepting a Bid for Evaluation

The next interaction (labelled as interaction 4 in Figure 25) specifies the execution of *m*'s operation *EvaluatingBid*, in which *m* accepts to discharge the commitment that *c* submit a bid, and accepts to evaluate the submitted bid. These acceptances are uttered if obligations exist in which *m* replies both to a proposal to discharge submitting a bid (which exist as obligations 8 and 9) and to a proposal to adopt evaluating a bid (which exist as obligations

10 and 11). The uttering of these acceptances triggers the following conversational policies:

- Policy 2 (replying to a proposal discharges the obligations to reply): the acceptance to uptake the operation proposed in β discharges the obligations to reply to β (thus deleting obligations 8 and 9 on the manager, and 9 and 10 on the contractor).
- Policy 2 (*ditto*): the acceptance to uptake the operation proposed in γ discharges the obligations to reply to γ (thus deleting obligations 10 and 11 on the manager, and 11 and 12 on the contractor).
- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in β causes the discharge of the commitment to submit a bid (labelled as commitment A in Figure 26) and all corresponding obligations (i.e., obligations 3, 4 and 5 on the manager, and 3 to 6 on the contractor).
- Policy 3 (*ditto*): the acceptance to uptake the operation proposed in γ causes the adoption of a commitment in which *m* evaluates a bid for *c* (added as commitment B), and its corresponding obligations (where the manager is obligated to evaluate a bid and communicate the result of this evaluation—which are obligations 12 to 15—and the contractor is obligated to hear this evaluation—as described by obligations 13 to 15).
- Policy 4 (accepting to discharge a *ProposingToDischarge* action discards the obligations in which the *discharger* of the action is to propose its discharge): the acceptance to discharge the commitment to submit a bid results in the discharge of the obligations in which *c* is to propose discharging the commitment that she submits a bid (which deletes obligations 6 and 7 on the manager, and 7 and 8 on the contractor), and lastly



	Manager & Contractor's Shared Social Commitments	Manager's Obligations	Contractor's Obligations
❺	B. (m,c,EvaluatingBid(m→c, {WashingCar(c,vw)},BOOLEAN))	policy 1: add 18, 19, 20, 21 12. EvaluatingBid(m→c,{WashingCar(c,vw)},BOOLEAN) 13. Processing(m,{WashingCar(c,vw)}}→BOOLEAN) 14. Speaking(m→c,{Inform(BOOLEAN)}) 15. Voicing(m→,{Inform(BOOLEAN)}) 16. Speaking(m→c,{Propose(-B)}) 17. Voicing(m→,{Propose(-B)}) 18. Speaking(c→m,{ReplyTo(δ)}) 19. Hearing(→m,{ReplyTo(δ)}) 20. Speaking(c→m,{ReplyTo(ε)}) 21. Hearing(→m,{ReplyTo(ε)})	policy 1: add 18, 19, 20, 21 13. EvaluatingBid(m→c, {WashingCar(c,vw)},BOOLEAN) 14. Speaking(m→c,{Inform(BOOLEAN)}) 15. Hearing(→c,{Inform(BOOLEAN)}) 16. Speaking(m→c,{Propose(-B)}) 17. Hearing(→c,{Propose(-B)}) 18. Speaking(c→m,{ReplyTo(δ)}) 19. Voicing(c→,{ReplyTo(δ)}) 20. Speaking(c→m,{ReplyTo(ε)}) 21. Voicing(c→,{ReplyTo(ε)})
❻	policy 3: delete B, add C ~~B. (m,c,EvaluatingBid(m→c, {WashingCar(c,vw)},BOOLEAN))~~ C. (c,m,ExecutingContract(c→m, {WashingCar(c,vw)},RESULTS))	policy 2: delete 18, 19, 20, 21 policy 3: delete 12, 13, 14, 15, add 22, 23, 24 policy 4: delete 16, 17, add 25, 26 ~~12. EvaluatingBid(m→c,{WashingCar(c,vw)},BOOLEAN)~~ ~~13. Processing(m,{WashingCar(c,vw)}}→BOOLEAN)~~ ~~14. Speaking(m→c,{Inform(BOOLEAN)})~~ ~~15. Voicing(m→,{Inform(BOOLEAN)})~~ ~~16. Speaking(m→c,{Propose(-B)})~~ ~~17. Voicing(m→,{Propose(-B)})~~ ~~18. Speaking(c→m,{ReplyTo(δ)})~~ ~~19. Hearing(→m,{ReplyTo(δ)})~~ ~~20. Speaking(c→m,{ReplyTo(ε)})~~ ~~21. Hearing(→m,{ReplyTo(ε)})~~ 22. ExecutingContract(c→m,{WashingCar(c,vw)},RESULTS) 23. Speaking(c→m,{Inform(RESULTS)}) 24. Hearing(→m,{Inform(RESULTS)}) 25. Speaking(c→m,{Propose(-C)}) 26. Hearing(→m,{Propose(-C)})	policy 2: delete 18, 19, 20, 21 policy 3: delete 13, 14, 15, add 22, 23, 24, 25 policy 4: delete 16, 17, add 26, 27 ~~13. EvaluatingBid(m→c, {WashingCar(c,vw)},BOOLEAN)~~ ~~14. Speaking(m→c,{Inform(BOOLEAN)})~~ ~~15. Hearing(→c,{Inform(BOOLEAN)})~~ ~~16. Speaking(m→c,{Propose(-B)})~~ ~~17. Hearing(→c,{Propose(-B)})~~ ~~18. Speaking(c→m,{ReplyTo(δ)})~~ ~~19. Voicing(c→,{ReplyTo(δ)})~~ ~~20. Speaking(c→m,{ReplyTo(ε)})~~ ~~21. Voicing(c→,{ReplyTo(ε)})~~ 22. ExecutingContract(c→m, {WashingCar(c,vw)},RESULTS) 23. Processing(c,{WashingCar(c,vw)}→RESULTS) 24. Speaking(c→m,{Inform(RESULTS)}) 25. Voicing(c→,{Inform(RESULTS)}) 26. Speaking(c→m,{Propose(-C)}) 27. Voicing(c→,{Propose(-C)})
❼	C. (c,m,ExecutingContract(c→m, {WashingCar(c,vw)},RESULTS))	policy 1: add 27, 28 22. ExecutingContract(c→m,{WashingCar(c,vw)},RESULTS) 23. Speaking(c→m,{Inform(RESULTS)}) 24. Hearing(→m,{Inform(RESULTS)}) 25. Speaking(c→m,{Propose(-C)}) 26. Hearing(→m,{Propose(-C)}) 27. Speaking(m→c,{ReplyTo(ζ)}) 28. Voicing(m→,{ReplyTo(ζ)})	policy 1: add 28, 29 22. ExecutingContract(c→m, {WashingCar(c,vw)},RESULTS) 23. Processing(c,{WashingCar(c,vw)}→RESULTS) 24. Speaking(c→m,{Inform(RESULTS)}) 25. Voicing(c→,{Inform(RESULTS)}) 26. Speaking(c→m,{Propose(-C)}) 27. Voicing(c→,{Propose(-C)}) 28. Speaking(m→c,{ReplyTo(ζ)}) 29. Hearing(→c,{ReplyTo(ζ)})
❽	policy 3: delete C ~~C. (c,m,ExecutingContract(c→m, {WashingCar(c,vw)},RESULTS))~~	policy 2: delete 27, 28 policy 3: delete 22, 23, 24 policy 4: delete 25,26 ~~22. ExecutingContract(c→m,{WashingCar(c,vw)},RESULTS)~~ ~~23. Speaking(c→m,{Inform(RESULTS)})~~ ~~24. Hearing(→m,{Inform(RESULTS)})~~ ~~25. Speaking(c→m,{Propose(-C)})~~ ~~26. Hearing(→m,{Propose(-C)})~~ ~~27. Speaking(m→c,{ReplyTo(ζ)})~~ ~~28. Voicing(m→,{ReplyTo(ζ)})~~	policy 2: delete 28, 29 policy 3: delete 22, 23, 24, 25 policy 4: delete 26,27 ~~22. ExecutingContract(c→m, {WashingCar(c,vw)},RESULTS)~~ ~~23. Processing(c,{WashingCar(c,vw)}→RESULTS)~~ ~~24. Speaking(c→m,{Inform(RESULTS)})~~ ~~25. Voicing(c→,{Inform(RESULTS)})~~ ~~26. Speaking(c→m,{Propose(-C)})~~ ~~27. Voicing(c→,{Propose(-C)})~~ ~~28. Speaking(m→c,{ReplyTo(ζ)})~~ ~~29. Hearing(→c,{ReplyTo(ζ)})~~

Figure 27. State of shared social commitments and obligations of the manager and contractor in the Contract Net conversation example (part 2 of 2).

- Policy 4 (accepting to adopt a *ProposingToDischarge* action obligates the *discharger* of the action to propose its discharge): the acceptance to adopt the action to evaluate a bid results in the adoption of obligations in which *m* (the *discharger*) is to propose *c* (the *discharged*) to discharge this action (which adds obligations 16 and 17 on both the manager and the contractor).

4.10.5 Awarding a Contract

The next interaction (labelled as interaction 5 in Figure 25) specifies the execution of *m*'s operation *AwardingContract*, in which *m* proposes to discharge that he evaluates a bid, informs the result of the evaluation, informs a contract, and proposes that the contractor adopt executing the given contract. The committal precondition for this utterance is that there exist obligations in which *m* is to propose discharging the commitment that he evaluates a bid for *c* (which are obligations 16 and 17). The uttering of these proposals (which are labelled δ and ε) triggers the following conversational policies:

- Policy 1 (the uttering of a proposal obligates the addressee to reply to the proposal): the uttering of proposal δ results in the adoption of obligations in which *c* replies to δ (added as obligations 18 and 19 in Figure 27 on both the manager and the contractor), and
- Policy 1 (*ditto*): the uttering of proposal ε results in the adoption of obligations in which *c* replies to ε (added as obligations 20 and 21 on both the manager and the contractor).

4.10.6 Accepting the Awarding of a Contract

The next interaction (labelled as interaction 6 in Figure 25) specifies the execution of *c*'s operation *AcceptingAward*, which specifies that *c* accepts the discharge of the commitment that *m* evaluates a bid for *c* (only if there are obligations in which *c* replies to a proposal to discharge this commitment—which exist as obligations 18 and 19), and that *c* accepts to adopt executing a contract (only if there are obligations in which *c* replies to a proposal to adopt executing a contract—which exist as obligations 20 and 21). The uttering of these acceptances results in the application of the following policies:

- Policy 2 (replying to a proposal discharges the obligation to reply): the acceptance to uptake the operation proposed in δ discharges the obligations to reply to δ (which deletes obligations 18 and 19 on both the manager and the contractor).
- Policy 2 (*ditto*): the acceptance to uptake the operation proposed in ε discharges the obligations to reply to ε (which deletes obligations 20 and 21 on both the manager and the contractor).
- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in δ causes the discharge of the commitment to evaluate a bid (labelled as commitment B) as well as its corresponding obligations (which are obligations 12 to 15 on the manager, and 13 to 15 on the contractor).
- Policy 3 (*ditto*): the acceptance to uptake the operation proposed in ε causes the adoption of a commitment in which *c* executes a contract for *m* (added as shared commitment C). The adoption of this commitment also results in the adoption of obligations in which the contractor executes the contract and informs its results (added as obligations 22 to 25) and the manager receives such results (added as obligations 22 to 24).
- Policy 4 (accepting to discharge a *ProposingToDischarge* action discharges the obligations in which the *discharger* proposes the discharge of the action): the acceptance to discharge the commitment to evaluate a bid results in the discharge of the obligations in which *m* is to propose discharging the commitment that he evaluates a bid (which deletes obligations 16 and 17 on both the manager and the contractor), and lastly,

- Policy 4 (accepting to adopt a *ProposingToDischarge* action obligates the *discharger* to propose the discharge of the action): the acceptance to adopt the action to execute a contract results in the adoption of obligations in which *c* (the *discharger*) is to propose to *m* (the *discharged*) discharging this action (which adds obligations 25 and 26 on the manager, and 26 and 27 on the contractor).

4.10.7 Submitting Results of Executing a Contract

The next interaction (labelled as interaction 7 in Figure 25) specifies the execution of *c*'s operation *SubmittingResults*, which specifies that *c* proposes to *m* the discharge of the commitment that she executes the contract (only if there are obligations in which she proposes to discharge the action—which exist as obligations 25 and 26). The uttering of this proposal (labelled ζ) triggers the following conversational policy:

- Policy 1 (the uttering of a proposal obligates the addressee to reply to the proposal): the uttering of proposal ζ results in the obligations in which *m* replies to ζ (which adds obligations 27 and 28 on the manager, and 28 and 29 on the contractor).

4.10.8 Accepting the Results of a Contract

The last interaction in this conversation (labelled as interaction 8 in Figure 25) indicates the execution of *m*'s operation *AcceptingResults*, which specifies that *m* accepts to discharge the execution of a contract. This acceptance is uttered if obligations exist in which *m* replies to a proposal to discharge the execution of the contract (which exist as obligations 27 and 28). The uttering of this acceptance results in the application of the following policies:

- Policy 2 (replying to a proposal discharges the obligation to reply): the acceptance to uptake the operation proposed in ζ discharges the obligation to reply to ζ (thus deleting obligations 27 and 28 on the manager, and 28 and 29 on the contractor).
- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in ζ causes the discharge of the

commitment to execute a contract (labelled as commitment C) as well as its corresponding obligations (which are obligations 22 to 24 on the manager, and 22 to 25 on the contractor), and lastly,

- Policy 4 (accepting to discharge a *ProposingToDischarge* action discards the obligations in which the *discharger* is to propose discharging the action): the acceptance to discharge the commitment to execute a contract results in the discharge of the obligations in which *m* is to propose discarding the commitment to execute the contract (therefore deleting obligations 25 and 26 on the manager, and 26 and 27 on the contractor).

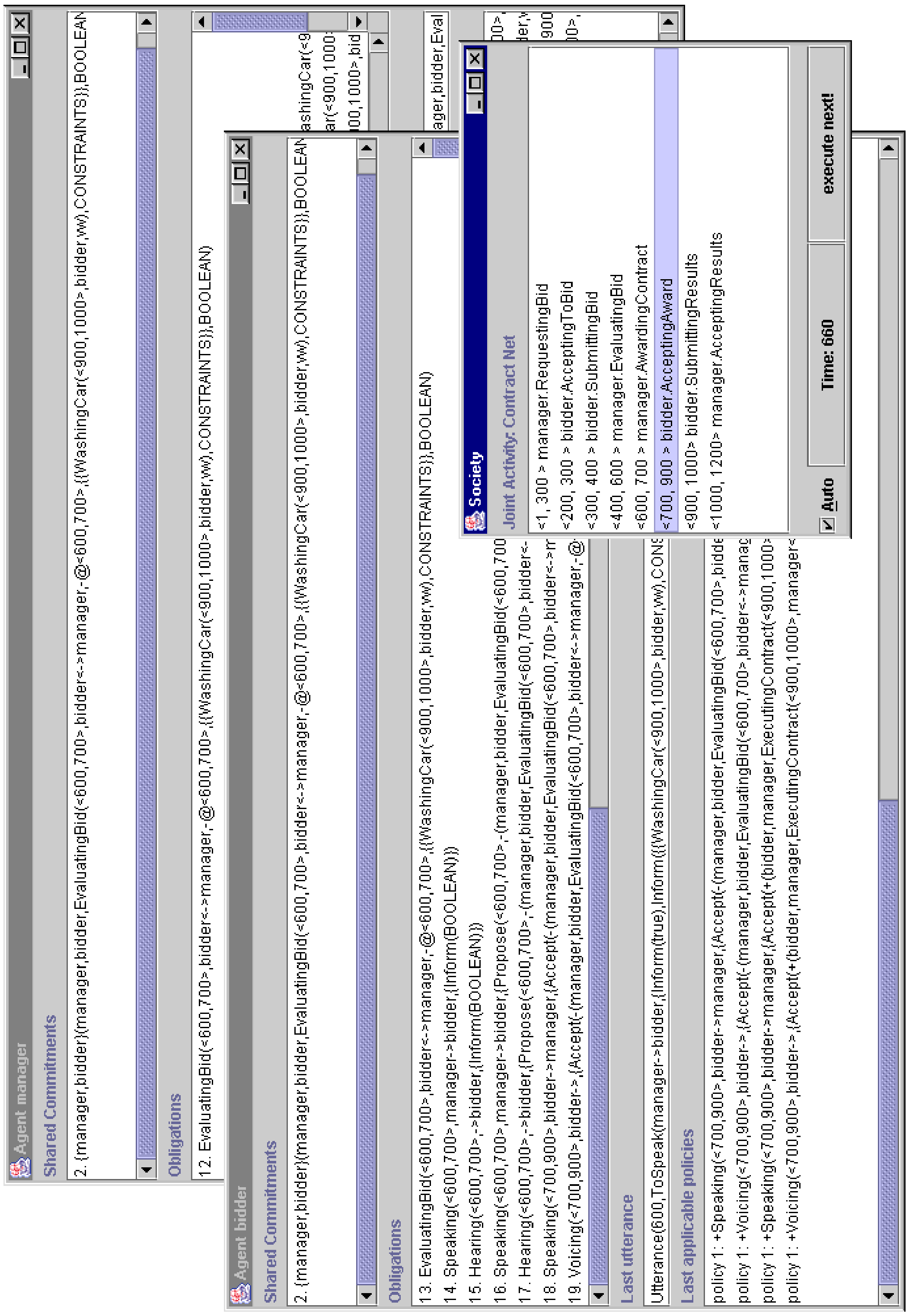
At this point, the interaction ends leaving none of the shared social commitments and obligations adopted during the interaction, thus indicating that this conversation does not result in commitments and obligations that outlive the activity.

To conclude, Figure 28 shows a snapshot of the program that was built and used to simulate the conversation of agents in the model for conversations. The figure shows an intermediate state of the Contract Net conversation described in this example.[38]

4.11 Example Proof

This section presents a brief proof to illustrate the formal inference supporting the model for conversations. This proof is based on the Contract Net Protocol example presented in the previous sections, and it shows that, given a request for bids from a manager to a contractor, it is possible to infer that the contractor can reply with either an acceptance or a rejection to bid.

[38] Refer to section *6.3: Implementation* for additional information on the issues related to the test bed implementation.


Agent manager

Shared Commitments

2. {manager,bidder}(manager,bidder,EvaluatingBid(<600,700>,bidder<->manager,-@<600,700>,{(WashingCar(<900,1000>,bidder,ww),CONSTRAINTS)},BOOLEAN

Obligations

12. EvaluatingBid(<600,700>,bidder<->manager,-@<600,700>,{(WashingCar(<900,1000>,bidder,ww),CONSTRAINTS)},BOOLEAN)

ashingCar(<9
ar(<900,1000
00,1000>,bid

ager,bidder,Eval

0>,
der,v
900
0>,

Agent bidder

Shared Commitments

2. {manager,bidder}(manager,bidder,EvaluatingBid(<600,700>,bidder<->manager,-@<600,700>,{(WashingCar(<900,1000>,bidder,ww),CONSTRAINTS)},BOOLEAN

Obligations

13. EvaluatingBid(<600,700>,bidder<->manager,-@<600,700>,{(WashingCar(<900,1000>,bidder,ww),CONSTRAINTS)},BOOLEAN)
14. Speaking(<600,700>,manager->bidder,{Inform(BOOLEAN)})
15. Hearing(<600,700>,->bidder,{Inform(BOOLEAN)})
16. Speaking(<600,700>,manager->bidder,{Propose(<600,700>,-(manager,bidder,EvaluatingBid(<600,700
17. Hearing(<600,700>,->bidder,{Propose(<600,700>,-(manager,bidder,EvaluatingBid(<600,700>,bidder<-
18. Speaking(<700,900>,bidder->manager,{Accept(-(manager,bidder,EvaluatingBid(<600,700>,bidder<->m
19. Voicing(<700,900>,bidder->,{Accept(-(manager,bidder,EvaluatingBid(<600,700>,bidder<->manager,-@

Last utterance

Utterance(600,ToSpeak(manager->bidder,{Inform(true),Inform({(WashingCar(<900,1000>,bidder,ww),CONS

Last applicable policies

policy 1: +Speaking(<700,900>,bidder->manager,{Accept(-(manager,bidder,EvaluatingBid(<600,700>,bidde
policy 1: +Voicing(<700,900>,bidder->,{Accept(-(manager,bidder,EvaluatingBid(<600,700>,bidder<->manag
policy 1: +Speaking(<700,900>,bidder->manager,{Accept(+(bidder,manager,ExecutingContract(<900,1000>
policy 1: +Voicing(<700,900>,bidder->,{Accept(+(bidder,manager,ExecutingContract(<900,1000>,manager<

Society

Joint Activity: Contract Net

<1, 300 > manager.RequestingBid
<200, 300 > bidder.AcceptingToBid
<300, 400 > bidder.SubmittingBid
<400, 600 > manager.EvaluatingBid
<600, 700 > manager.AwardingContract
<700, 900 > bidder.AcceptingAward
<900, 1000> bidder.SubmittingResults
<1000, 1200> manager.AcceptingResults

Figure 28. Snapshot of the simulation of the Contract Net conversation example.

4.11.1 Assumptions

This proof specifies that there exists a manager (Alice) and a contractor (Bob), and that the manager has requested that the contractor submit a bid (i.e., the manager has performed the method *RequestingBid* found in the class *Manager*). These assumptions are defined as follows.

$$\exists\, b : Bidding \\ \bullet\ alice = b.manager \wedge bob = b.bidder \wedge alice.RequestingBid(b)$$

The outcome to prove is that Bob either accepts or rejects this request (i.e., he performs the method *AcceptingToBid* or *RejectingToBid* found in the class *Contractor*). This result (which is shown below) represents the first message sequence in the interaction defined in the Contract Net joint activity.

$$\exists\, b : Bidding \\ \bullet\ bob.AcceptingToBid(b) \vee bob.RejectingToBid(b)$$

Lastly, it is assumed that once Alice has uttered the request, Bob will reply to it (rather than just ignore it, for example).

4.11.2 Supporting Predicates

The following definitions are used throughout the proof. These definitions are based on the Object-Z specifications given in Chapter 3 (the model for conversations) and the current chapter.

The method *SendUtterance* in the class *Agent* is shown below. This method specifies that an uttered utterance becomes the current utterance of the speaker and addressee.[39]

[39] This definition is a simplification of the original definition given in Chapter 3, which specifies that utterances must be queued in the inbox of agents while they wait to become the current utterance to be processed through the norms of the society. The version herein presented simplifies this process with the only intention to make the proof more accessible.

$$
\begin{array}{l}
\forall\, agent : Agent;\; s : ToSpeak \\
\bullet\; agent = s.speaker \land \\
\quad agent.SendUtterance(s) \Rightarrow \\
\quad (\exists\, u : Utterance \\
\quad\;\bullet\; u.speechAct = s \land \\
\quad\quad u = s.speaker.current \land \\
\quad\quad u = s.addressee.current)
\end{array}
$$

Next is the definition of the conversation policy *PFPpolicy1*. This policy specifies that uttering a proposal creates obligations to reply to the proposal.

$$
\begin{array}{l}
\forall\, agent : Agent;\; u : Utterance;\; p : Propose \\
\bullet\; agent.current = u \land \\
\quad p \in u.speechAct.points \Rightarrow \\
\quad (\exists\, s : Speaking \\
\quad\;\bullet\; s.speaker = u.speechAct.addressee \land \\
\quad\quad s.addressee = u.speechAct.speaker \land \\
\quad\quad s.points = isReplyTo(p) \land \\
\quad\quad (agent = s.speaker \Rightarrow \{s, s.voice\} \subseteq agent.obligations) \land \\
\quad\quad (agent = s.addressee \Rightarrow \{s, s.hear\} \subseteq agent.obligations))
\end{array}
$$

This policy makes use of the predicate *isReplyTo*, which specifies that the reply to a proposal is either an acceptance or a rejection. This predicate is defined below.

$$
\begin{array}{l}
\forall\, p : \downarrow Propose \\
\bullet\; \exists\, points : \mathbb{P}_1\, IllocutionaryPoint;\; a : Accept;\; r : \downarrow Reject \\
\quad\bullet\; isReplyTo(p) = points \land \\
\quad\quad a.accepting = p.proposing \land \\
\quad\quad r.rejecting = p.proposing \land \\
\quad\quad ((a \in points \land r \notin points) \lor \\
\quad\quad\; (r \in points \land a \notin points))
\end{array}
$$

Next are the operations *RequestingBid* and *ProposeToAdoptBidding* found in the class *Manager*. These operations specify the behaviour for requesting a bid.

$$
\begin{array}{l}
\forall\, m : Manager;\; b : Bidding;\; s : ToSpeak \\
\bullet\; m.RequestingBid(b) \Leftrightarrow \\
\quad m.ProposeToAdoptBidding(b, s) \land \\
\quad m.SendUtterance(s)
\end{array}
$$

$$\begin{array}{l}
\forall\, m : Manager;\ b : Bidding;\ s : ToSpeak;\ p : Propose \\
\bullet\ m.ProposeToAdoptBidding(b, s) \Leftrightarrow \\
\quad s.speaker = b.manager = m \wedge \\
\quad s.addressee = b.bidder \wedge \\
\quad p \in s.points \wedge \\
\quad p.proposing \in Add \wedge \\
\quad p.proposing.commitment = isCommitmentToBid(b)
\end{array}$$

Shown below are the operations *AcceptingToBid* and *AcceptToAdoptBidding* (which specify the behaviour for accepting to commit to bid), and *RejectingToBid* and *RejectToAdoptBidding* (which specify the behaviour for rejecting to commit to bid). These operations are found in the class *Contractor*.

$$\begin{array}{l}
\forall\, c : Contractor;\ b : Bidding;\ s : ToSpeak \\
\bullet\ c.AcceptingToBid(b) \Leftrightarrow \\
\quad existsReplyToProposeToAdoptBidding(c, b) \wedge \\
\quad c.AcceptToAdoptBidding(b, s) \wedge \\
\quad c.SendUtterance(s)
\end{array}$$

$$\begin{array}{l}
\forall\, c : Contractor;\ b : Bidding;\ s : ToSpeak;\ a : Accept;\ r : \downarrow Reject \\
\bullet\ c.AcceptToAdoptBidding(b, s) \Leftrightarrow \\
\quad s.speaker = b.bidder = c \wedge \\
\quad s.addressee = b.manager \wedge \\
\quad a.accepting \in Add \wedge \\
\quad a.accepting.commitment = isCommitmentToBid(b) \wedge \\
\quad a.accepting = r.rejecting \wedge \\
\quad (a \in s.points \wedge r \notin s.points)
\end{array}$$

$$\begin{array}{l}
\forall\, c : Contractor;\ b : Bidding;\ s : ToSpeak \\
\bullet\ c.RejectingToBid(b) \Leftrightarrow \\
\quad existsReplyToProposeToAdoptBidding(c, b) \wedge \\
\quad c.RejectToAdoptBidding(b, s) \wedge \\
\quad c.SendUtterance(s)
\end{array}$$

$$\begin{array}{l}
\forall\, c : Contractor;\ b : Bidding;\ s : ToSpeak;\ r : \downarrow Reject;\ a : Accept \\
\bullet\ c.RejectToAdoptBidding(b, s) \Leftrightarrow \\
\quad s.speaker = b.bidder = c \land \\
\quad s.addressee = b.manager \land \\
\quad r.rejecting \in Add \land \\
\quad r.rejecting.commitment = isCommitmentToBid(b) \land \\
\quad r.rejecting = a.accepting \land \\
\quad (r \in s.points \land a \notin s.points)
\end{array}$$

Lastly, the predicate *existsReplyToProposeToAdoptBidding* is shown below. This predicate is referenced by the operations *AcceptingToBid* and *RejectingToBid* above.

$$\begin{array}{l}
\forall\, agent : Agent;\ b : Bidding;\ s : Speaking;\ p : Propose \\
\bullet\ existsReplyToProposeToAdoptBidding(agent, b) \Leftrightarrow \\
\quad s \in agent.obligations \land \\
\quad s.speaker = b.bidder \land \\
\quad s.addressee = b.manager \land \\
\quad s.points = isReplyTo(p) \land \\
\quad p.proposing \in Add \land \\
\quad p.proposing.commitment = isCommitmentToBid(b)
\end{array}$$

4.11.3 Proof

The steps below show the process followed to conclude that given that Alice has requested Bob to submit a bid, Bob either accepts or rejects the request. The first half of the proof (steps 1 to 52) introduces the assumptions and shows that these assumptions lead to the uptake of obligations to reply. The second half (steps 53 to 88) shows that these obligations lead to either a reply accepting or a reply rejecting to submit a bid.

As such, the first step is to introduce the assumptions.

Step	Result	Justification
1	∃ *b*:*Bidding* • alice = *b*.manager ∧ bob = *b*.bidder ∧ alice.*RequestingBid*(*b*)	Assumption (Alice requests Bob to submit a bid).

From these assumptions we derive the following about Alice and Bob.

Step	Result	Justification

2	alice = bid.manager $\wedge$
	bob = bid.bidder $\wedge$
	alice.*RequestingBid*(bid)
		$\exists$-Elimination (bid=*b*).
		Step 1.

3	alice = bid.manager	$\wedge$-Elimination. Step 2.
4	bob = bid.bidder	$\wedge$-Elimination. Step 2.
5	alice.*RequestingBid*(bid)	$\wedge$-Elimination. Step 2.

The steps below show how these assumptions lead to the request being Alice and Bob's current utterance. First, Alice's operation *RequestingBid* is decomposed into the operations *ProposeToAdoptBidding* and *SendUtterance*.

Step	Result	Justification
6	$\forall$ *m*:*Manager*; *b*:*Bidding*; *s*:*ToSpeak* • *m*.*RequestingBid*(*b*) $\Leftrightarrow$ *m*.*ProposeToAdoptBidding*(*b*, *s*) $\wedge$ *m*.*SendUtterance*(*s*)	Definition of the operation *RequestingBid* in the class *Manager*
7	alice.*RequestingBid*(bid) $\Rightarrow$ alice.*ProposeToAdoptBidding*(bid, request) $\wedge$ alice.*SendUtterance*(request)	$\forall$-Elimination (alice=*m*, bob=*c*, bid=*b*, request=*s*) and $\Leftrightarrow$-Elimination. Step 6.
8	alice.*ProposeToAdoptBidding*(bid, request) $\wedge$ alice.*SendUtterance*(request)	Modus ponens. Steps 5 and 7.
9	alice.*ProposeToAdoptBidding*(bid, request)	$\wedge$-Elimination. Step 8.
10	alice.*SendUtterance*(request)	$\wedge$-Elimination. Step 8.

Next, that Alice is the speaker of the proposal and that she has uttered it to Bob results in this utterance being Bob's current utterance.

Step	Result	Justification
11	$\forall$ *m*:*Manager*; *b*:*Bidding*; *s*:*ToSpeak*; *p*:*Propose* • *m*.*ProposeToAdoptBidding*(*b*, *s*) $\Leftrightarrow$ *s*.speaker = *b*.manager = *m* $\wedge$ *s*.addressee = *b*.bidder $\wedge$ *p* $\in$ *s*.points $\wedge$ *p*.proposing $\in$ *Add* $\wedge$ *p*.proposing.commitment = *isCommitmentToBid*(*b*)	Definition of the operation *ProposeToAdoptBidding* in the class *Manager*
12	alice.*ProposeToAdoptBidding*(bid, request) $\Rightarrow$ request.speaker = bid.manager = alice $\wedge$ request.addressee = bid.bidder $\wedge$ $p_1 \in$ request.points $\wedge$ p_1.proposing $\in$ *Add* $\wedge$ p_1.proposing.commitment = *isCommitmentToBid*(bid)	$\forall$-Elimination (alice=*m*, bob=*c*, bid=*b*, request=*s*, p_1=*p*) and $\Leftrightarrow$-Elimination. Step 11.
13	request.speaker = bid.manager = alice $\wedge$ request.addressee = bid.bidder $\wedge$ $p_1 \in$ request.points $\wedge$ p_1.proposing $\in$ *Add* $\wedge$ p_1.proposing.commitment = *isCommitmentToBid*(bid)	Modus ponens. Steps 9 and 12.
14	request.speaker = bid.manager = alice	$\wedge$-Elimination. Step 13.
15	alice = request.speaker	Equality. Step 14.
16	alice = request.speaker $\wedge$ alice.*SendUtterance*(request)	$\wedge$-Introduction. Steps 10 and 15.
17	request.addressee = bid.bidder	$\wedge$-Elimination. Step 13.
18	request.addressee = bob	Equality. Steps 4 and 17.
19	$p_1 \in$ request.points	$\wedge$-Elimination. Step 13.
20	p_1.proposing $\in$ *Add*	$\wedge$-Elimination. Step 13.
21	p_1.proposing.commitment = *isCommitmentToBid*(bid)	$\wedge$-Elimination. Step 13.

Step	Result	Justification
22	$\forall$ *agent*:*Agent*; *s*:*ToSpeak* $\bullet$ *agent* = *s*.speaker $\wedge$ *agent*.*SendUtterance*(*s*) $\Rightarrow$ ($\exists$ *u*:*Utterance* $\bullet$ *u*.speechAct = *s* $\wedge$ *u* = *s*.speaker.current $\wedge$ *u* = *s*.addressee.current)	Definition of the operation *SendUtterance* in the class *Agent.*
23	alice = request.speaker $\wedge$ alice.*SendUtterance*(request) $\Rightarrow$ u_1.speechAct = request $\wedge$ u_1 = request.speaker.current $\wedge$ u_1 = request.addressee.current	$\forall$-Elimination (alice=*agent*, request=*s*) and $\exists$-Elimination (u_1=*u*). Step 22.
24	u_1.speechAct = request $\wedge$ u_1 = request.speaker.current $\wedge$ u_1 = request.addressee.current	Modus ponens. Steps 16 and 23.
25	u_1.speechAct = request	$\wedge$-Elimination. Step 24.
26	$p_1 \in u_1$.speechAct.points	Substitution. Steps 19 and 25.
27	u_1 = request.addressee.current	$\wedge$-Elimination. Step 24.
28	u_1 = bob.current	Substitution. Steps 18 and 27.
29	bob.current = u_1	Equality. Step 28.
30	bob.current = u_1 $\wedge$ $p_1 \in u_1$.speechAct.points	$\wedge$-Introduction. Steps 26 and 29.

The next steps show that this utterance leads to obligations in which Bob replies to Alice's request.

Step	Result	Justification
31	$\forall$ *agent*:*Agent*; *u*:*Utterance*; *p*:*Propose* • *agent*.current = *u* $\wedge$ *p* $\in$ *u*.speechAct.points $\Rightarrow$ ($\exists$ *s*:*Speaking* • *s*.speaker = *u*.speechAct.addressee $\wedge$ *s*.addressee = *u*.speechAct.speaker $\wedge$ *s*.points = *isReplyTo*(*p*) $\wedge$ (*agent* = *s*.speaker $\Rightarrow$ { *s*, *s*.voice } $\subseteq$ *agent*.obligations) $\wedge$ (*agent* = *s*.addressee $\Rightarrow$ { *s*, *s*.hear } $\subseteq$ *agent*.obligations))	Definition of the conversation policy *PFPpolicy1* (a proposal commits agents to reply).
32	bob.current = u_1 $\wedge$ p_1 $\in$ u_1.speechAct.points $\Rightarrow$ reply.speaker = u_1.speechAct.addressee $\wedge$ reply.addressee = u_1.speechAct.speaker $\wedge$ reply.points = *isReplyTo*(p_1) $\wedge$ (bob = reply.speaker $\Rightarrow$ {reply, reply.voice} $\subseteq$ bob.obligations) $\wedge$ (bob = reply.addressee $\Rightarrow$ {reply, reply.hear} $\subseteq$ bob.obligations)	$\forall$-Elimination (bob=*agent*, u_1=*u*, p_1=*p*) and $\exists$-Elimination (reply =*s*, a_1=*a*, r_1=*r*). Step 31.
33	reply.speaker = u_1.speechAct.addressee $\wedge$ reply.addressee = u_1.speechAct.speaker $\wedge$ reply.points = *isReplyTo*(p_1) $\wedge$ (bob = reply.speaker $\Rightarrow$ { reply, reply.voice} $\subseteq$ bob.obligations) $\wedge$ (bob = reply.addressee $\Rightarrow$ { reply, reply.hear} $\subseteq$ bob.obligations)	Modus ponens. Steps 30 and 32.
34	reply.speaker = u_1.speechAct.addressee	$\wedge$-Elimination. Step 33.
35	reply.addressee = u_1.speechAct.speaker	$\wedge$-Elimination. Step 33.
36	reply.points = *isReplyTo*(p_1)	$\wedge$-Elimination. Step 33.
37	bob = reply.speaker $\Rightarrow$ {reply, reply.voice} $\subseteq$ bob.obligations	$\wedge$-Elimination. Step 33.

Step	Result	Justification
38	reply.speaker = request.addressee	Substitution. Steps 25 and 34.
39	reply.speaker = bob	Substitution. Steps 18 and 38.
40	bob = reply.speaker	Equality. Step 39.
41	reply.addressee = request.speaker	Substitution. Steps 25 and 35.
42	reply.addressee = alice	Substitution. Steps 15 and 41.
43	$\{$reply, reply.voice$\} \subseteq$ bob.obligations	Modus ponens. Steps 37 and 40.
44	reply $\in$ bob.obligations	Set theory. Step 43.

So far, it has been derived that Bob has obligations to reply to Alice's proposal (as identified by the predicate *isReplyTo*). The steps below show that this reply contains either an *Accept* or a *Reject* illocutionary point with identical characteristics to that of the *Propose* illocutionary point found in Alice's request.

Step	Result	Justification
45	$\forall\, p{:}\downarrow\!Propose$ $\bullet\ \exists\, points{:}\mathbb{P}_1\ IllocutionaryPoint;$ $a{:}Accept;\ r{:}\downarrow\!Reject$ $\bullet\ isReplyTo(\ p\ ) = points\ \wedge$ $a.\text{accepting} = p.\text{proposing}\ \wedge$ $r.\text{rejecting} = p.\text{proposing}\ \wedge$ $((\ a \in points \wedge r \notin points\ ) \vee$ $(\ r \in points \wedge a \notin points\ ))$	Definition of the predicate *isReplyTo*.
46	$isReplyTo(\ \text{p}_1\ ) = \text{points}_1\ \wedge$ $\text{a}_1.\text{accepting} = \text{p}_1.\text{proposing}\ \wedge$ $\text{r}_1.\text{rejecting} = \text{p}_1.\text{proposing}\ \wedge$ $((\ \text{a}_1 \in \text{points}_1 \wedge \text{r}_1 \notin \text{points}_1\ ) \vee$ $(\ \text{r}_1 \in \text{points}_1 \wedge \text{a}_1 \notin \text{points}_1\ ))$	$\forall$-Elimination (p_1=p) and $\exists$-Elimination (points_1=$points$, a_1=a, r_1=r). Step 45.
47	$isReplyTo(\ \text{p}_1\ ) = \text{points}_1$	$\wedge$-Elimination. Step 46.
48	$\text{a}_1.\text{accepting} = \text{p}_1.\text{proposing}$	$\wedge$-Elimination. Step 46.

Step	Result	Justification
49	r_1.rejecting = p_1.proposing	$\wedge$-Elimination. Step 46.
50	$((a_1 \in \text{points}_1 \wedge r_1 \notin \text{points}_1) \vee$ $(r_1 \in \text{points}_1 \wedge a_1 \notin \text{points}_1))$	$\wedge$-Elimination. Step 46.
51	reply.points = points_1	Equality. Steps 36 and 47.
52	$((a_1 \in \text{reply.points} \wedge r_1 \notin \text{reply.points}) \vee$ $(r_1 \in \text{reply.points} \wedge a_1 \notin \text{reply.points}))$	Equality. Steps 50 and 51.

At this point Bob is committed to reply with an acceptance or a rejection. The steps below show that this obligation results in Bob's uttering of an acceptance (*AcceptingToBid*) or a rejection (*RejectingToBid*). To reach this outcome it must first be derived that the predicate *existsReplyToProposeToAdoptBidding* is true given Bob's current obligations. This is shown in the steps below.

Step	Result	Justification
53	$\forall$ *agent*:*Agent*; *b*:*Bidding*; *s*:*Speaking*; *p*:*Propose* • *existsReplyToProposeToAdoptBidding*(*agent,b*) $\Leftrightarrow$ *s* $\in$ *agent*.obligations $\wedge$ *s*.speaker = *b*.bidder $\wedge$ *s*.addressee = *b*.manager $\wedge$ *s*.points = *isReplyTo*(*p*) $\wedge$ *p*.proposing $\in$ *Add* $\wedge$ *p*.proposing.commitment = *isCommitmentToBid*(*b*)	Definition of the predicate *existsReplyToPropose-ToAdoptBidding*
54	*existsReplyToProposeToAdoptBidding*(bob, bid) $\Leftarrow$ reply $\in$ bob.obligations $\wedge$ reply.speaker = bid.bidder $\wedge$ reply.addressee = bid.manager $\wedge$ reply.points = *isReplyTo*(p_1) $\wedge$ p_1.proposing $\in$ *Add* $\wedge$ p_1.proposing.commitment = *isCommitmentToBid*(bid)	$\forall$-Elimination (bob=*agent*, bid=*b*, reply=*s*, p_1=*p*, a_1=*a*, r_1=*r*) and $\Leftrightarrow$-Elimination. Step 53.

Step	Result	Justification
55	reply.speaker = bid.bidder	Substitution. Steps 4 and 39.
56	reply.addressee = bid.manager	Substitution. Steps 3 and 42.
57	reply $\in$ bob.obligations $\wedge$ reply.speaker = bid.bidder $\wedge$ reply.addressee = bid.manager $\wedge$ reply.points = *isReplyTo*(p_1) $\wedge$ p_1.proposing $\in$ *Add* $\wedge$ p_1.proposing.commitment = *isCommitmentToBid*(bid)	$\wedge$-Introduction. Steps 20, 21, 36, 44, 55 and 56.
58	*existsReplyToProposeToAdoptBidding*(bob, bid)	Modus ponens. Steps 54 and 57.

Up to this point it has been determined that Bob holds an obligation in which he replies to Alice's proposal. As stated at the beginning of the proof, it is assumed that Bob is willing to utter (and is capable of uttering) such a reply. This assumption is added below.

Step	Result	Justification
59	bob.*SendUtterance*(reply)	Assumption (Bob replies)

As indicated in Step 52, this reply is given as a speech act named *reply* that contains either an acceptance or a rejection of the proposal. These two cases (that of an acceptance and that of a rejection) are explored in the steps below.

The first assumption is that Bob accepts the proposal.

Step	Result	Justification
60	$(a_1 \in \text{reply.points} \wedge r_1 \notin \text{reply.points})$	Assumption (Bob accepts)
61	$\forall\, c{:}Contractor;\, b{:}Bidding;\, s{:}ToSpeak;\, a{:}Accept;\, r{:}\downarrow Reject$ $\bullet\ c.AcceptToAdoptBidding(b, s) \Leftrightarrow$ $s.\text{speaker} = b.\text{bidder} = c \wedge$ $s.\text{addressee} = b.\text{manager} \wedge$ $a.\text{accepting} \in Add \wedge$ $a.\text{accepting.commitment} = isCommitmentToBid(b) \wedge$ $a.\text{accepting} = r.\text{rejecting} \wedge$ $(a \in s.\text{points} \wedge r \notin s.\text{points})$	Definition of the operation *AcceptToAdoptBidding* in the class *Contractor*.
62	$\text{bob}.AcceptToAdoptBidding(\text{bid, reply}) \Leftarrow$ $\text{reply.speaker} = \text{bid.bidder} = \text{bob} \wedge$ $\text{reply.addressee} = \text{bid.manager} \wedge$ $a_1.\text{accepting} \in Add \wedge$ $a_1.\text{accepting.commitment} = isCommitmentToBid(\text{bid}) \wedge$ $a_1.\text{accepting} = r_1.\text{rejecting} \wedge$ $(a_1 \in \text{reply.points} \wedge r_1 \notin \text{reply.points})$	$\forall$-Elimination (bob=c, bid=b, reply=s, a_1=a, r_1=r) and $\Leftrightarrow$-Elimination. Step 61.
63	$\text{reply.speaker} = \text{bid.bidder} = \text{bob}$	Equality. Steps 4 and 55.
64	$a_1.\text{accepting} \in Add$	Equality. Steps 20 and 48.
65	$a_1.\text{accepting.commitment} = isCommitmentToBid(\text{bid})$	Equality. Steps 21 and 48.
66	$a_1.\text{accepting} = r_1.\text{rejecting}$	Equality. Steps 48 and 49.

Step	Result	Justification
67	reply.speaker = bid.bidder = bob $\wedge$ reply.addressee = bid.manager $\wedge$ a_1.accepting $\in$ *Add* $\wedge$ a_1.accepting.commitment = *isCommitmentToBid*(bid) $\wedge$ a_1.accepting = r_1.rejecting $\wedge$ ($a_1 \in$ reply.points $\wedge$ $r_1 \notin$ reply.points)	$\wedge$-Introduction. Steps 56, 60, 63, 64, 65 and 66.
68	bob.*AcceptToAdoptBidding*(bid, reply)	Modus ponens. Steps 62 and 67.
69	$\forall$ *c*:*Contractor*; *b*:*Bidding*; *s*:*ToSpeak* • *c*.*AcceptingToBid*(*b*) $\Leftrightarrow$ *existsReplyToProposeToAdoptBidding*(*c*, *b*) $\wedge$ *c*.*AcceptToAdoptBidding*(*b*, *s*) $\wedge$ *c*.*SendUtterance*(*s*)	Definition of the operation *AcceptingToBid* in the class *Contractor*.
70	bob.*AcceptingToBid*(bid) $\Leftarrow$ *existsReplyToProposeToAdoptBidding*(bob,bid) $\wedge$ bob.*AcceptToAdoptBidding*(bid, reply) $\wedge$ bob.*SendUtterance*(reply)	$\forall$-Elimination (bob=*c*, bid=*b*, reply=*s*) and $\Leftrightarrow$-Elimination. Step 69.
71	*existsReplyToProposeToAdoptBidding*(bob,bid) $\wedge$ bob.*AcceptToAdoptBidding*(bid, reply) $\wedge$ bob.*SendUtterance*(reply)	$\wedge$-Introduction. Steps 58, 59 and 68.
72	bob.*AcceptingToBid*(bid)	Modus ponens. Steps 70 and 71.
73	bob.*AcceptingToBid*(bid) $\vee$ bob.*RejectingToBid*(bid)	$\vee$-Introduction. Step 72.

The steps below show the same process as above but to derive that Bob rejects the proposal.

Step	Result	Justification
74	$(r_1 \in \text{reply.points} \wedge a_1 \notin \text{reply.points})$	Assumption (Bob rejects)
75	$\forall\, c{:}Contractor;\ b{:}Bidding;\ s{:}ToSpeak;\ r{:}\downarrow Reject;\ a{:}Accept \bullet c.RejectToAdoptBidding(b, s) \Leftrightarrow s.\text{speaker} = b.\text{bidder} = c \wedge s.\text{addressee} = b.\text{manager} \wedge r.\text{rejecting} \in Add \wedge r.\text{rejecting.commitment} = isCommitmentToBid(b) \wedge r.\text{rejecting} = a.\text{accepting} \wedge (r \in s.\text{points} \wedge a \notin s.\text{points})$	Definition of the operation *RejectToAdoptBidding* in the class *Contractor*.
76	$\text{bob}.RejectToAdoptBidding(\text{bid, reply}) \Leftarrow \text{reply.speaker} = \text{bid.bidder} = \text{bob} \wedge \text{reply.addressee} = \text{bid.manager} \wedge r_1.\text{rejecting} \in Add \wedge r_1.\text{rejecting.commitment} = isCommitmentToBid(\text{bid}) \wedge r_1.\text{rejecting} = a_1.\text{accepting} \wedge (r_1 \in \text{reply.points} \wedge a_1 \notin \text{reply.points})$	$\forall$-Elimination (bob=*c*, bid=*b*, reply=*s*, $r_1=r$, $a_1=a$) and $\Leftrightarrow$-Elimination. Step 75.
77	$r_1.\text{rejecting} \in Add$	Equality. Steps 20 and 49.
78	$r_1.\text{rejecting.commitment} = isCommitmentToBid(\text{bid})$	Equality. Steps 21 and 49.
79	$r_1.\text{rejecting} = a_1.\text{accepting}$	Equality. Step 66.

Step	Result	Justification
80	reply.speaker = bid.bidder = bob $\wedge$ reply.addressee = bid.manager $\wedge$ r_1.rejecting $\in$ *Add* $\wedge$ r_1.rejecting.commitment = *isCommitmentToBid*(bid) $\wedge$ r_1.rejecting = a_1.accepting $\wedge$ ($r_1 \in$ reply.points $\wedge$ $a_1 \notin$ reply.points)	$\wedge$-Introduction. Steps 56, 63, 74, 77, 78 and 79.
81	bob.*RejectToAdoptBidding*(bid, reply)	Modus ponens. Steps 76 and 80.
82	$\forall$ *c*:*Contractor*; *b*:*Bidding*; *s*:*ToSpeak* • *c*.*RejectingToBid*(*b*) $\Leftrightarrow$ *existsReplyToProposeToAdoptBidding*(*c*, *b*) $\wedge$ *c*.*RejectToAdoptBidding*(*b*, *s*) $\wedge$ *c*.*SendUtterance*(*s*)	Definition of the operation *RejectingToBid* in the class *Contractor*.
83	bob.*RejectingToBid*(bid) $\Leftarrow$ *existsReplyToProposeToAdoptBidding*(bob,bid) $\wedge$ bob.*RejectToAdoptBidding*(bid, reply) $\wedge$ bob.*SendUtterance*(reply)	$\forall$-Elimination (bob=*c*, bid=*b*, reply=*s*) and $\Leftrightarrow$-Elimination. Step 82.
84	*existsReplyToProposeToAdoptBidding*(bob,bid) $\wedge$ bob.*RejectToAdoptBidding*(bid, reply) $\wedge$ bob.*SendUtterance*(reply)	$\wedge$-Introduction. Steps 58, 59 and 81.
85	bob.*RejectingToBid*(bid)	Modus ponens. Steps 83 and 84.
86	bob.*AcceptingToBid*(bid) $\vee$ bob.*RejectingToBid*(bid)	$\vee$-Introduction. Step 85.

Lastly, the steps below join the findings from the assumptions in Steps 73 and 86, and conclude by showing that these findings are the expected outcome of the proof.

Step	Result	Justification
87	bob.*AcceptingToBid*(bid) $\vee$ bob.*RejectingToBid*(bid)	$\vee$-Enumeration, Steps 52, 60, 61-73, 74 and 75-86.
88	$\exists$ *b*:*Bidding* • bob.*AcceptingToBid*(*b*) $\vee$ bob.*RejectingToBid*(*b*)	$\exists$-Introduction (bid=*b*). Step 87.
	QED	

4.12 Summary

This chapter presented an example of how the model for conversations can be applied to structure the conversations of agents interacting in a joint activity, in this case the Contract Net Protocol. This example involved two agents: a manager (who requests bids and selects the bid to be awarded for execution) and a contractor (who produces bids and executes awarded contracts). This example also involved three joint actions: submitting bid (in which the contractor produces and communicates a bid to the manager), evaluating bid (in which the manager decides and communicates whether or not a bid is awarded for execution), and executing contract (in which the contractor executes a contract and communicates its results to the manager).

The dynamics of this model were illustrated through an example conversation that began with a manager requesting a bid to a contractor. The contractor's acceptance and later submission of a bid followed this request. At which point, the manager accepted the bid for evaluation, which led to its awarding as a contract. Lastly, the contractor accepted the contract, executed it and submitted its results to the manager, thus ending the conversation.

In addition to the example, this chapter also presented a formal proof showing the logical inference followed to structure conversations using the model for conversations.

To conclude, it is worthwhile to remark that the CNP is a high-level task allocation mechanism that was designed with no specific domain of application in mind. In a certain way, that the model for conversations can be applied to abstract protocols like this one suggests the versatility that this model can offer to define interactions on concrete joint activities. The next section explores in that direction and presents the definition of a fictional (but quite feasible) e-commerce joint activity.

Chapter 5
Example: eBookstore Shopping

5.1 Overview

This chapter provides an example of how the model for conversations can be used to model the interaction of agents in concrete, practical joint activities. Specifically, this example models a scenario in which a buyer agent buys books from a seller agent, who then requests the services of a carrier for delivering those books to the buyer.

This chapter begins with a section describing this scenario. Subsequent sections describe the conversation elements used in interactions, that is, actions, social commitments, illocutionary points and agent participations. Finally, this chapter concludes with an example conversation where books are bought and delivered.

5.2 eBookstore Shopping

This example begins when a buyer agent approaches a seller agent in the fictitious Internet bookstore *eBookStore* and requests that she sell him certain books (whose description he provides). After making sure that the requested books can be sold (e.g., there are copies in stock), the seller requests that the buyer pay for them. Once a payment has been produced, the seller gives the buyer a proof of purchase (in the form of a receipt) and informs him that a carrier will deliver the books he bought.[40] After completing the sale, the seller contacts the carrier agent and requests that he deliver the books.[41] This request is followed by another request in which the carrier asks the seller to provide a payment for the delivery. Once a payment has been submitted, the carrier notifies the seller of his acceptance to deliver.[42] The interaction ends when the carrier contacts the buyer and delivers the books he was given by the seller.[43]

5.2.1 Payable Actions

This specification defines two actions (selling and delivering) as *payable actions*.

A payable action is one that inherits from the class *Action* and the class *Payable*, which (as shown below) specifies a *payer* agent and a *payee* agent.

$Payable$

$payer, payee : \downarrow Agent$

Complementing this definition, the class *PayableBy* is specified as a subclass of *Payable* that defines the time interval in which payment is to be proposed.

[40] A person visiting a bookstore can expect to walk out of the store with the books he bought. Differently, buying through an electronic medium limits the ability of the buyer to take immediate possession of items (unless they are in digital format). In this example it is assumed that books are delivered through a carrier agent—due to the incompatible natures of the medium of interaction and the books.

[41] Strictly speaking, the interaction modelled in this chapter would have allowed that the seller contacts the carrier to assure delivery as soon as the buyer requests a sale (although a cautious seller may wait to contact the carrier until at least the books have been paid for). For simplicity, the seller in this conversation assumes that the provision of deliveries is reliable (i.e., it follows the *late commitment* strategy).

[42] Although in this example the actions of selling and delivering require a payment to be produced prior to performance, that is not the case with all payable actions (i.e., actions that need to be paid). The restaurant industry provides examples in which payment is requested both before and after performing the action of serving meals. That is the case of fast-food outlets (in which payment is requested prior to serving a meal) and sit-down diners (in which payment is asked after the meal has being served). It is worth noticing that the principles described in this chapter can be applied without distinction to both cases.

[43] This is a simple case of delivery, in which the carrier just hands out the parcel to the recipient. Other more elaborate cases may require the receiver to provide a token (e.g., a signature, a photo id) that the carrier could use to demonstrate proof of delivery.

```
┌─ PayableBy ─────────────────────────────
│ Payable
│ ┌───────────────────────────────────
│ │ time : Interval
│ └───────────────────────────────────
└─────────────────────────────────────────
```

5.2.2 Conversation Policies

The interesting part about payable actions is not their state (as described above) but their behaviour. This behaviour (which is specified through conversation policies) specifies that once a commitment referencing a payable action has been proposed, there exists an obligation in which the payee will propose to the payer to pay for the execution of the action (i.e., the payee will propose the adoption of a shared commitment in which the payer performs the action of paying to the payee).

In the case of selling, for example, once the buyer has proposed to the seller that she sell him an item (where selling is a payable action) then the seller has the obligation to propose to the buyer that he pay for the sale.[44]

In general, an obligation in which a payee is to propose to a payer to pay for a payable action can be discharged if:

a) The payer accepts to pay for the action (the buyer commits to pay the seller), or

b) The proposal to adopt the payable action is rejected (the seller rejects to sell)

The adoption and discharge of obligations to propose paying is modelled by the conversation policies *PayablePolicy1* and *PayablePolicy2*.

The class *PayablePolicy1* (shown in Figure 29 and Figure 30) is a subclass of *ActionNorm* which specifies that:

[44] As a side thought, this behaviour is consistent with scenarios in which the seller is offering the sale of items, as in the case of public markets, where sellers openly proclaim their products (and their prices) to passing-by prospective shoppers. In such markets, competitive sellers could counterpropose their own proposals for payment to attract buyers. Accordingly, these sellers could stop offering their products if buyers reject buying them (in which case overzealous—if not annoying—sellers could propose to sell again), if buyers accept one of the proposed prices, or if the sellers desist (e.g., by rejecting their own proposal to sell).

$$
\begin{array}{l}
\textbf{PayablePolicy1} \\
\upharpoonleft(ProcessUtterance, ProcessAction) \\
ActionNorm \\
\quad \textbf{ProcessAction} \\
\quad agent? : \downarrow Agent \\
\quad sc? : SocialCommitment \\
\quad payable? : \downarrow PayableBy \\
\quad result! : \mathbb{P}\ Obligation \\
\hline
\quad \exists\, speak : Speaking \mid \\
\quad\quad speak = (\mu\, sa : Speaking \mid \\
\qquad\qquad\qquad sa.time = payable?.time \land \\
\qquad\qquad\qquad sa.speaker = payable?.payee \land \\
\qquad\qquad\qquad sa.addressee = payable?.payer \land \\
\qquad\qquad\qquad (\exists\, propose : \downarrow Propose \mid \\
\qquad\qquad\qquad\quad propose \in sa.points \land \\
\qquad\qquad\qquad\quad propose.proposing \in Add \\
\qquad\qquad\qquad\ \bullet \exists\, sc : SocialCommitment \mid \\
\qquad\qquad\qquad\qquad sc = propose.proposing.commitment \land \\
\qquad\qquad\qquad\qquad sc.debtor = payable?.payer \land \\
\qquad\qquad\qquad\qquad sc.creditor = payable?.payee \\
\qquad\qquad\qquad\quad \bullet \exists\, paying : Paying \mid \\
\qquad\qquad\qquad\qquad\quad paying = sc.action \\
\qquad\qquad\qquad\qquad \bullet\ paying.time = payable?.time \land \\
\qquad\qquad\qquad\qquad\quad paying.payer = payable?.payer \land \\
\qquad\qquad\qquad\qquad\quad paying.payee = payable?.payee \land \\
\qquad\qquad\qquad\qquad\quad paying.payable = payable?)) \\
\quad \bullet\ (agent? = payable?.payee \land result! = \{speak, speak.voice\}) \lor \\
\quad\quad (agent? = payable?.payer \land result! = \{speak, speak.hear\})
\end{array}
$$

Figure 29. *PayablePolicy1* (*Policy 6*): Adopting/discharging obligations to request a payment based on the negotiation of a commitment to a payable action (part 1 of 2).

a) Proposing to adopt a payable action causes the adoption of an obligation to propose paying, and

b) Rejecting a proposal to adopt a payable action causes the discharge of the obligation to propose paying.

```
$ProcessUtterance \mathrel{\widehat=} [\ agent? : \downarrow Agent;\ u? : Utterance\ ] \bullet$
    $(\bigwedge action : \downarrow Action \mid$
        $\exists propose : \downarrow Propose \mid$
          $propose \in u?.speechAct.points \land$
          $propose.proposing \in Add$
        $\bullet\ propose.proposing.commitment.action = action$
      $\bullet\ action.ProcessNorm \mathbin{\S} agent?.AddObligations) \land$
    $(\bigwedge action : \downarrow Action \mid$
        $\exists reject : \downarrow Reject \mid$
          $reject \in u?.speechAct.points \land$
          $reject.rejecting \in Add \land$
          $reject.rejecting.commitment.action = action$
        $\bullet\ \#(getProposeForReject(agent?.history,$
                                        $u?.speechAct.speaker,$
                                        $u?.speechAct.addressee,$
                                        $reject, u?.time)) \geq 1$
      $\bullet\ action.ProcessNorm \mathbin{\S} agent?.DeleteObligations)$
```

Figure 30. *PayablePolicy1* (*Policy 6*): Adopting/discharging obligations to request a payment based on the negotiation of a commitment to a payable action (part 2 of 2).

Lastly, the class *PayablePolicy2* (shown in Figure 31) is an action norm that states that the obligation to propose paying is discharged if a commitment to pay has been adopted.

To recap, the sections above specified data structures and policies supporting the general modelling of payable actions. In contrast, the following sections will define the elements specific to buying books at an *eBookStore* outlet.

5.3 Information

There are four types of information that are communicated in a shopping interaction: books, payments, invoices and receipts.

The class *BookDescription* is a subclass of *Data* which specifies a title as a sequence of characters (more specific definitions could also list ISBN, edition, publisher, and so on). Instances of this class are used for describing books.

PayablePolicy2

$$\upharpoonleft(ProcessUtterance, ProcessAction)$$

$$ActionNorm$$

ProcessAction

$$\begin{array}{l}
agent? : \downarrow Agent \\
sc? : SocialCommitment \\
paying? : Paying \\
result! : \mathbb{P}\ Obligation \\
\hline
\exists\, speak : Speaking \mid \\
\quad speak = (\mu\, sa : Speaking \mid \\
\qquad\qquad sa.time = paying?.time \land \\
\qquad\qquad sa.speaker = paying?.payee \land \\
\qquad\qquad sa.addressee = paying?.payer \land \\
\qquad\qquad (\exists\, propose : \downarrow Propose \mid \\
\qquad\qquad\quad propose \in sa.points \land \\
\qquad\qquad\quad propose.proposing \in Add \\
\qquad\qquad\ \bullet \exists\, sc : SocialCommitment \mid \\
\qquad\qquad\qquad sc = propose.proposing.commitment \\
\qquad\qquad\quad \bullet\ sc.debtor = sc?.debtor \land \\
\qquad\qquad\qquad sc.creditor = sc?.creditor \land \\
\qquad\qquad\qquad sc.action = paying?)) \\
\bullet\ (agent? = paying?.payee \land result! = \{speak, speak.voice\}) \lor \\
\quad (agent? = paying?.payer \land result! = \{speak, speak.hear\})
\end{array}$$

$$\begin{array}{l}
ProcessUtterance \mathrel{\widehat=} [\ agent? : \downarrow Agent;\ \ u? : Utterance\ ] \bullet \\
\qquad \bigwedge action : \downarrow Action \mid \\
\qquad\quad \exists\, accept : Accept \mid \\
\qquad\qquad accept \in u?.speechAct.points \land \\
\qquad\qquad accept.accepting.commitment.action = action \\
\qquad\quad \bullet\ \#(getProposeForAccept(agent?.history, \\
\qquad\qquad\qquad\qquad\qquad\qquad u?.speechAct.speaker, \\
\qquad\qquad\qquad\qquad\qquad\qquad u?.speechAct.addressee, \\
\qquad\qquad\qquad\qquad\qquad\qquad accept, u?.time)) \geq 1 \\
\qquad \bullet\ action.ProcessNorm \ \comp\ agent?.DeleteObligations
\end{array}$$

Figure 31. *PayablePolicy2* (*Policy 7*): Discharging obligations to request a payment given that a commitment to pay has been adopted.

$BookDescription$

$Data$

$title : \mathrm{seq}\, char$

The class *Book* inherits from *BookDescription* and specifies the contents of a book as a sequence of characters (other book definitions could define figures and illustrations).

$Book$

$BookDescription$

$contents : \mathrm{seq}\, char$

Payments involve the exchange of currency among interacting agents. As such, the class *Currency* is the superclass of all monetary notes (such as cash and cheques). This class defines all currencies to have a non-negative value.

$Currency$

$value : \mathbb{R}$

$value \geq 0$

Based on this definition, the classes *eCash* and *eCredit* specify more concrete (but still elementary) types of currency (subclasses may define data such as a monetary body backing the note (e.g., Bank of Canada, MasterCard) and other additional identifiers (e.g., serial number, expiration date)).

$eCash$

$Currency$

$eCredit$

$Currency$

Currency is the instrument for making payments. As such, the class *Payment* is defined as a data type that contains an instance of type *Currency*.

Payment

$Data$

$currency : \downarrow Currency_{©}$

Transactions usually involve other pieces of information besides a payment: an invoice (representing the terms of a sale), and a receipt (representing the proof of purchase). This information is represented by the classes *Invoice* and *Receipt*, which in this example are defined as subclasses of *Data* which specify the amount being charged (in the case of an invoice) and the items bought (in the case of the receipt).

Invoice

$Data$

$amount : \mathbb{R}$

$amount \geq 0$

Receipt

$Data$

$items : \mathbb{P}_1 \downarrow Data$

5.4 Actions

There are three actions involved in this example. These actions are:

- *To sell*: in which a seller sells items (e.g., books) to a buyer.
- *To deliver*: in which a carrier delivers a parcel (e.g., books) to a receiver, and
- *To pay*: in which a payer provides a payment to a payee (e.g., a buyer pays the seller for books, the seller pays a delivery charge to the carrier).

5.4.1 Selling

Selling is a payable action in which a seller transfers the property of goods to a buyer. This action is defined as a class named *ToSell* that inherits from *ToProduce*, *ToProposeToDischarge* and *Payable*. As shown below, this action declares the variables *seller* (as the *producer*, *discharger* and *payee*), *buyer* (as the *receiver*, *discharged* and *payer*) and *items* (referring to the items involved in the sale). In addition, this definition specifies that the data produced by the seller and communicated to the buyer is an instance of type *Receipt* listing the items sold.

$ToSell$

$$\begin{array}{l} ToProduce \\ ToProposeToDischarge \\ Payable \\ \hline seller : \downarrow Seller \\ buyer : \downarrow Buyer \\ items : \mathbb{P}_1 \downarrow Data \\ \hline seller = producer = discharger = payee \land \\ buyer = receiver = discharged = payer \land \\ items = in \\ \exists\, receipt : Receipt \mid \\ \quad receipt.items = items \\ \bullet \{receipt\} = out \end{array}$$

Based on this definition, the action *Selling* is defined as the union of the classes *ToSell*, *CompositeActing*, *ProposingToDischarge* and *PayableBy*.

$$Selling == ToSell \bigcup CompositeActing \bigcup ProposingToDischarge \bigcup PayableBy$$

5.4.2 Delivering

Delivering is a payable action in which a carrier receives a parcel from a sender and hands it out to a receiver. This action is defined as a class named *ToDeliver* that inherits from *ToProduce* and *Payable*, and which specifies the variables *sender* (as the *payer* of the action), *carrier* (as the *producer* and *payee*) and *parcel* (as the items that are transmitted to the *receiver*). This class also specifies that the act of communicating the parcel is an instance of type *ToHandOutParcel* (which is defined next) whose variable *dispatcher* is the same as the *sender* of the delivery.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\mathit{ToDeliver}} \\
\mathit{ToProduce} \\
\mathit{Payable} \\
\begin{array}{|l}
\hline
\mathit{sender}, \mathit{carrier} : \downarrow \mathit{Agent} \\
\mathit{parcel} : \mathbb{P}_1 \downarrow \mathit{Data} \\
\hline
\mathit{sender} = \mathit{payer} \wedge \\
\mathit{carrier} = \mathit{producer} = \mathit{payee} \wedge \\
\mathit{parcel} = \mathit{in} = \mathit{out} \\
\exists\, \mathit{handOut} : \mathit{ToHandOutParcel} \mid \\
\quad \mathit{handOut}.\mathit{dispatcher} = \mathit{sender} \\
\bullet\ \mathit{handOut} = \mathit{speak} \\
\hline
\end{array} \\
\hline
\end{array}
$$

The class *ToHandOutParcel* is an action inheriting from *ToSpeak* and *ToProposeToDischarge* that defines the variables *dispatcher* (which is the sender of the delivery), *carrier* (which is the *speaker* and *discharger*), *receiver* (as the *addressee* and *discharged*), and *parcel* (as the item being informed).[45]

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\mathit{ToHandOutParcel}} \\
\mathit{ToSpeak} \\
\mathit{ToProposeToDischarge} \\
\begin{array}{|l}
\hline
\mathit{dispatcher}, \mathit{carrier}, \mathit{receiver} : \downarrow \mathit{Agent} \\
\mathit{parcel} : \mathbb{P} \downarrow \mathit{Data} \\
\hline
\mathit{carrier} = \mathit{speaker} = \mathit{discharger} \wedge \\
\mathit{receiver} = \mathit{addressee} = \mathit{discharged} \\
\exists\, i : \mathit{Inform} \mid \\
\quad i \in \mathit{points} \\
\bullet\ i.\mathit{informing} = \mathit{parcel} \\
\hline
\end{array} \\
\hline
\end{array}
$$

Based on the above definitions, the actions *Delivering* and *HandingOutParcel* are defined as:

[45] This definition isolates the action of handing out a parcel from the action of delivering it (where the latter subsumes the former). This allows the sender and the carrier to interact with the buyer while disclosing as little information about the delivery as possible (such as account numbers, pick up sites, rebates, and so on)—all of which might be described in a delivery instance.

$Delivering == ToDeliver \bigcup CompositeActing \bigcup PayableBy$

$HandingOutParcel == ToHandOutParcel \bigcup CompositeActing \bigcup ProposingToDischarge$

5.4.3 Paying

Paying is an action in which a payer provides to a payee a payment covering the amount specified in an invoice. This action is defined as a class named *ToPay* inheriting from *ToProduce* and *ToProposeDischarge* that declares the variables *payer* (as the *producer* and *discharger*), *payee* (as the *receiver* and *discharged*), *payable* (as the good or service for which payment is requested), and *invoice* (which specifies the amount requested as payment). This definition also specifies that the data produced by the payer and communicated to the payee is an instance of type *Payment* whose value equals the amount requested in the invoice.[46]

$$
\begin{array}{l}
\textbf{ToPay} \\
\hline
ToProduce \\
ToProposeToDischarge \\
\hline
payer, payee : \downarrow Agent \\
payable : \downarrow Payable \\
invoice : Invoice \\
\hline
payer = producer = discharger \land \\
payee = receiver = discharged \land \\
in = \{invoice\} \\
\exists\, payment : Payment \mid \\
\quad payment.currency.value = invoice.amount \\
\bullet \{payment\} = out \\
\hline
\end{array}
$$

Based on this definition, the action *Paying* is defined as the union of the classes *ToPay*, *CompositeActing* and *ProposingToDischarge*.

[46] By defining the value of a payment as equal to the amount requested in an invoice, this specification explicitly avoids the need for additional interactions to return change in cases where the value of a payment exceeds the amount due. Although it would have been possible to model these additional interactions, they were omitted for simplicity.

$Paying == ToPay \bigcup CompositeActing \bigcup ProposingToDischarge$

5.5 Social Commitments

There are four axioms that identify the social commitments involved in a shopping interaction: they identify *commitments to sell*, *commitments to pay*, *commitments to deliver*, and *commitments to hand out a parcel*.

5.5.1 Commitment to Sell

The axiom *isCommitmentToSell* is a function that returns a social commitment binding a seller to sell items to a buyer. Specifically, this function receives a *Selling* action and returns a social commitment that has as its creditor and debtor the buyer and seller of the action, which is then specified as the action of the commitment.

$$\begin{array}{|l}
isCommitmentToSell : Selling \rightarrow SocialCommitment \\
\hline
\forall selling : Selling \\
\bullet \exists sc : SocialCommitment \mid \\
\quad sc.creditor = selling.buyer \land \\
\quad sc.debtor = selling.seller \land \\
\quad sc.action = selling \\
\quad \bullet\ sc = isCommitmentToSell(selling)
\end{array}$$

5.5.2 Commitment to Pay

The axiom *isCommitmentToPay* is a function that returns a social commitment binding a payer to produce a payment to a payee. Specifically, this function receives a *Paying* action and returns a social commitment that has as its creditor and debtor the payee and payer of the action, which is then specified as the action of the commitment.

$$
\begin{array}{|l}
isCommitmentToPay : Paying \rightarrow SocialCommitment \\
\hline
\forall paying : Paying \\
\bullet\ \exists sc : SocialCommitment \mid \\
\quad sc.creditor = paying.payee \land \\
\quad sc.debtor = paying.payer \land \\
\quad sc.action = paying \\
\quad \bullet\ sc = isCommitmentToPay(paying)
\end{array}
$$

5.5.3 Commitment to Deliver

Likewise, the axiom *isCommitmentToDeliver* is a function that returns a social commitment binding a carrier to deliver a parcel for the sender. Specifically, this function receives a *Delivering* action and returns a social commitment that has as its creditor and debtor the sender and carrier of the action, which is then specified as the action of the commitment.

$$
\begin{array}{|l}
isCommitmentToDeliver : Delivering \rightarrow SocialCommitment \\
\hline
\forall delivering : Delivering \\
\bullet\ \exists sc : SocialCommitment \mid \\
\quad sc.creditor = delivering.sender \land \\
\quad sc.debtor = delivering.carrier \land \\
\quad sc.action = delivering \\
\quad \bullet\ sc = isCommitmentToDeliver(delivering)
\end{array}
$$

5.5.4 Commitment to Hand Out a Parcel

Lastly, the axiom *isCommitmentToHandOutParcel* is a function that returns a social commitment binding a dispatcher (i.e., the sender of a delivery) and a carrier to hand out a parcel for the sender. Specifically, this function receives a *HandingOutParcel* action and returns a social commitment that has as its creditor and debtor the dispatcher and carrier of the action, which is then specified as the action of the commitment.

$$
\begin{array}{|l}
isCommitmentToHandOutParcel : HandingOutParcel \rightarrow SocialCommitment \\
\hline
\forall\, handingOut : HandingOutParcel \\
\bullet\ \exists\, sc : SocialCommitment \mid \\
\quad sc.creditor = handingOut.dispatcher \wedge \\
\quad sc.debtor = handingOut.carrier \wedge \\
\quad sc.action = handingOut \\
\quad \bullet\ sc = isCommitmentToHandOutParcel(handingOut)
\end{array}
$$

The next section shows how these axioms support the definition of illocutionary points for shopping interactions.

5.6 Illocutionary Points

This section defines the various illocutionary points used in shopping interactions. These illocutionary points include *proposals to sell books*, *acceptances to deliver a parcel*, and *informing a payment*, among others.

5.6.1 *Proposing to Sell*

Interactions for buying books begin when a prospective buyer requests a seller to sell the books whose descriptions are provided. This communication is supported by the axiom definitions *isProposeToAdoptSelling* and *isInformBookDescriptions*.

The axiom *isProposeToAdoptSelling* defines a function that receives a *Selling* action and an interval within which a reply is expected, and returns a proposal to adopt a social commitment to this selling action.

$$
\begin{array}{|l}
isProposeToAdoptSelling : Selling \times Interval \rightarrow {\downarrow}Propose \\
\hline
\forall\, selling : Selling;\ reply : Interval \\
\bullet\ \exists\, propose : {\downarrow}Propose \mid \\
\quad propose.proposing \in Add \wedge \\
\quad propose.reply = reply \wedge \\
\quad propose.proposing.commitment = isCommitmentToSell(selling) \\
\quad \bullet\ propose = isProposeToAdoptSelling(selling, reply)
\end{array}
$$

The axiom *isInformBookDescriptions* defines a function that receives a non-empty set of instances of type *BookDescription* and returns an inform informing these descriptions.

$$\begin{array}{|l}
isInformBookDescriptions : \mathbb{P}_1\ BookDescription \rightarrow Inform \\
\hline
\forall\, books : \mathbb{P}_1\ BookDescription \\
\bullet\ \exists\, inform : Inform \mid \\
\quad inform.informing = books \\
\quad \bullet\ inform = isInformBookDescriptions(books)
\end{array}$$

5.6.2 Accepting to Sell

Once a request is issued, it is expected that it will be replied to with an acceptance or a rejection. To support acceptances of this request, the axiom *isAcceptToAdoptSelling* receives a *Selling* action and returns an acceptance to adopt committing to this action.

$$\begin{array}{|l}
isAcceptToAdoptSelling : Selling \rightarrow Accept \\
\hline
\forall\, selling : Selling \\
\bullet\ \exists\, accept : Accept \mid \\
\quad accept.accepting \in Add\ \wedge \\
\quad accept.accepting.commitment = isCommitmentToSell(selling) \\
\quad \bullet\ accept = isAcceptToAdoptSelling(selling)
\end{array}$$

5.6.3 Proposing to End a Sale

Once a seller has agreed to sell books, she is responsible for producing and submitting a proof of purchase to the buyer, indicating that such a sale has taken place. This implies that the seller needs to provide a receipt, and propose to discharge the commitment in which she is to sell the requested books. In addition, since the seller cannot hand out the books directly to the buyer, she must inform the buyer that a carrier will be responsible for the books' delivery. This communication is modelled through the axioms *isProposeToDischargeSelling*, *isInformReceipt* and *isProposeToAdoptHandingOutParcel*, which specify the illocutionary points for proposing to discharge the commitment to sell, for informing a receipt, and for proposing to adopt the commitment for handing out a parcel (which is later specified as the books sold).

The axiom *isProposeToDischargeSelling* is a function that receives a *Selling* action and an interval specifying an expected reply time, and returns a proposal to discharge a social commitment to do the selling action.

$$
\begin{array}{|l}
isProposeToDischargeSelling : Selling \times Interval \rightarrow \downarrow Propose \\
\hline
\forall\, selling : Selling;\ reply : Interval \\
\bullet\ \exists\, propose : \downarrow Propose \mid \\
\quad propose.proposing \in Delete \land \\
\quad propose.reply = reply \land \\
\quad propose.proposing.commitment = isCommitmentToSell(selling) \\
\quad \bullet\ propose = isProposeToDischargeSelling(selling, reply)
\end{array}
$$

The axiom *isInformReceipt* is a function that returns an *Inform* informing a receipt.

$$
\begin{array}{|l}
isInformReceipt : Receipt \rightarrow Inform \\
\hline
\forall\, receipt : Receipt \\
\bullet\ \exists\, inform : Inform \mid \\
\quad inform.informing = \{receipt\} \\
\quad \bullet\ inform = isInformReceipt(receipt)
\end{array}
$$

Lastly, the axiom *isProposeToAdoptHandingOutParcel* is a function that receives a *HandingOutParcel* action and an interval specifying an expected time of reply, and returns a proposal to adopt a social commitment to this handing out action.

$$
\begin{array}{|l}
isProposeToAdoptHandingOutParcel : HandingOutParcel \times Interval \rightarrow \downarrow Propose \\
\hline
\forall\, handingOut : HandingOutParcel;\ reply : Interval \\
\bullet\ \exists\, propose : \downarrow Propose \mid \\
\quad propose.proposing \in Add \land \\
\quad propose.reply = reply \land \\
\quad propose.proposing.commitment = isCommitmentToHandOutParcel(handingOut) \\
\quad \bullet\ propose = isProposeToAdoptHandingOutParcel(handingOut, reply)
\end{array}
$$

5.6.4 Accepting to End a Sale

After a seller has proposed to a buyer to end the sale, it is expected that the buyer will reply both to the proposal that the seller is no longer committed to sell the books, and to the proposal that he adopt a commitment in which the carrier will deliver the books. These acceptances are modelled through the axioms *isAcceptToDischargeSelling* and *isAcceptToAdoptHandingOutParcel*, which specify the illocutionary points for accepting the discharge of selling, and for accepting the adoption of being handed out a parcel, respectively.

The axiom *isAcceptToDischargeSelling* is a function that receives a Selling action as input and returns an acceptance to discharge a social commitment to this selling action.

$$
\begin{array}{|l}
isAcceptToDischargeSelling : Selling \rightarrow Accept \\
\hline
\forall\, selling : Selling \\
\bullet\ \exists\, accept : Accept \mid \\
\quad accept.accepting \in Delete \land \\
\quad accept.accepting.commitment = isCommitmentToSell(selling) \\
\quad \bullet\ accept = isAcceptToDischargeSelling(selling)
\end{array}
$$

Likewise, the axiom *isAcceptToAdoptHandingOutParcel* is a function that receives a *HandingOutParcel* action as input and returns an acceptance to adopt a social commitment to this handing out action.

$$
\begin{array}{|l}
isAcceptToAdoptHandingOutParcel : HandingOutParcel \rightarrow Accept \\
\hline
\forall\, handingOut : HandingOutParcel \\
\bullet\ \exists\, accept : Accept \mid \\
\quad accept.accepting \in Add \land \\
\quad accept.accepting.commitment = isCommitmentToHandOutParcel(handingOut) \\
\quad \bullet\ accept = isAcceptToAdoptHandingOutParcel(handingOut)
\end{array}
$$

5.6.5 Proposing to Deliver

After the sale has been completed, the seller will contact the carrier to request that he delivers the books sold. To communicate this request, the seller sends a proposal to adopt the commitment that the carrier delivers a set of books. This communication is supported by the axioms *isProposeToAdoptDelivering* and *isInformBooks*, which specify the illocutionary points for proposing to adopt the delivery of books, and for informing the books that will be delivered, respectively.

The axiom *isProposeToAdoptDelivery* is a function that receives a *Delivering* action and an interval specifying an expected reply time, and returns a proposal to adopt a social commitment to do the delivery.

$$
\begin{array}{|l}
isProposeToAdoptDelivering : Delivering \times Interval \rightarrow \downarrow Propose \\
\hline
\forall\, delivering : Delivering;\ reply : Interval \\
\bullet\ \exists\, propose : \downarrow Propose\ | \\
\quad propose.proposing \in Add \land \\
\quad propose.reply = reply \land \\
\quad propose.proposing.commitment = isCommitmentToDeliver(delivering) \\
\quad \bullet\ propose = isProposeToAdoptDelivering(delivering, reply)
\end{array}
$$

Lastly, the axiom *isInformBooks* is a function that receives a set of instances of type *Book* and returns an inform informing these books.

$$
\begin{array}{|l}
isInformBooks : \mathbb{P}_1\, Book \rightarrow Inform \\
\hline
\forall\, books : \mathbb{P}_1\, Book \\
\bullet\ \exists\, inform : Inform\ | \\
\quad inform.informing = books \\
\quad \bullet\ inform = isInformBooks(books)
\end{array}
$$

5.6.6 Accepting to Deliver

After issuing the above request for delivery, it is expected that the carrier will accept to do the delivery. To that end, the axiom *isAcceptToAdoptDelivering* defines a function that returns an illocutionary point accepting to adopt a commitment to deliver a parcel.

$$
\begin{array}{|l}
isAcceptToAdoptDelivering : Delivering \rightarrow Accept \\
\hline
\forall\, delivering : Delivering \\
\bullet\ \exists\, accept : Accept\ | \\
\quad accept.accepting \in Add \land \\
\quad accept.accepting.commitment = isCommitmentToDeliver(delivering) \\
\quad \bullet\ accept = isAcceptToAdoptDelivering(delivering)
\end{array}
$$

5.6.7 Handing Out a Parcel

Once that the delivery of the books is accepted by the carrier, he is committed to hand out the parcel to the receiver. This communication is supported by the axioms *isProposeToDischargeHandingOutParcel* and (the previously described) *isInformBooks*.

The axiom *isProposeToDischargeHandingOutParcel* defines a function that receives a *HandingOutParcel* action and an expected time of reply, and returns a proposal to discharge the action of handing out a parcel.[47]

$$
\begin{array}{|l}
isProposeToDischargeHandingOutParcel : \\
\qquad\qquad\qquad\qquad HandingOutParcel \times Interval \rightarrow \downarrow Propose \\
\hline
\forall\, handingOut : HandingOutParcel;\ reply : Interval \\
\bullet\ \exists\, propose : \downarrow Propose \mid \\
\quad propose.proposing \in Delete \land \\
\quad propose.reply = reply \land \\
\quad propose.proposing.commitment = isCommitmentToHandOutParcel(handingOut) \\
\quad \bullet\ propose = isProposeToDischargeHandingOutParcel(handingOut, reply)
\end{array}
$$

5.6.8 Proposing to Pay

Once an agent has proposed the adoption of a payable action, the payee of the action (if it happens to be one of the interacting agents) adopts the obligation to propose to the payer to commit paying for the payable action. The axioms *isProposeToAdoptPaying* and *isInformInvoice* specify the illocutionary points for proposing to adopt a commitment to pay, and for informing the amount that this payment must cover.

The axiom *isProposeToAdoptPaying* is a function that receives an instance of type *Paying* and an interval indicating an expected time of reply, and returns a proposal to adopt the specified paying action.

[47] It is worth commenting that this example models the interaction between the carrier and the receiver based on the discharge of the action in which the carrier hands out a parcel to the receiver, that is, without any preceding interaction between them to adopt it. As it is explained later in this chapter, this specification effectively minimizes the interactions between the carrier and receiver and allows both to communicate the parcel being delivered, and to discharge the obligations in which the parcel is delivered.

$$
\begin{array}{|l}
isProposeToAdoptPaying : Paying \times Interval \rightarrow \downarrow Propose \\
\hline
\forall paying : Paying;\ reply : Interval \\
\bullet \exists propose : \downarrow Propose \mid \\
\quad propose.proposing \in Add \land \\
\quad propose.reply = reply \land \\
\quad propose.proposing.commitment = isCommitmentToPay(paying) \\
\quad \bullet propose = isProposeToAdoptPaying(paying, reply)
\end{array}
$$

The axiom *isInformInvoice* is a function that informs a given invoice.

$$
\begin{array}{|l}
isInformInvoice : Invoice \rightarrow Inform \\
\hline
\forall amount : Invoice \\
\bullet \exists inform : Inform \mid \\
\quad inform.informing = \{amount\} \\
\quad \bullet inform = isInformInvoice(amount)
\end{array}
$$

5.6.9 Accepting to Pay

After the proposal to adopt a paying action has been issued, one could expect that this proposal is accepted. This acceptance is modelled by the axiom *isAcceptToAdoptPaying*, which specifies a function that takes a *Paying* action as input and returns an illocutionary point accepting to adopt a commitment to pay.

$$
\begin{array}{|l}
isAcceptToAdoptPaying : Paying \rightarrow Accept \\
\hline
\forall paying : Paying \\
\bullet \exists accept : Accept \mid \\
\quad accept.accepting \in Add \land \\
\quad accept.accepting.commitment = isCommitmentToPay(paying) \\
\quad \bullet accept = isAcceptToAdoptPaying(paying)
\end{array}
$$

5.6.10 Submitting a Payment

Accepting a commitment to pay makes the payer responsible for paying to the payee. To that end, the axioms *isInformPayment* and *isProposeToDischargePaying* define the illocutionary points for submitting a payment, and for proposing to discharge a commitment to pay, respectively.

The axiom *isProposeToDischargePaying* is a function that receives an instance of type *Paying* and interval indicating an expected time of reply, and returns a proposal to discharge a commitment to pay.

$$
\begin{array}{|l}
isProposeToDischargePaying : Paying \times Interval \rightarrow \downarrow Propose \\
\hline
\forall paying : Paying;\ reply : Interval \\
\bullet \exists propose : \downarrow Propose \mid \\
\quad propose.proposing \in Delete \land \\
\quad propose.reply = reply \land \\
\quad propose.proposing.commitment = isCommitmentToPay(paying) \\
\quad \bullet propose = isProposeToDischargePaying(paying, reply)
\end{array}
$$

The axiom *isInformPayment* is a function that informs a given payment.

$$
\begin{array}{|l}
isInformPayment : Payment \rightarrow Inform \\
\hline
\forall payment : Payment \\
\bullet \exists inform : Inform \mid \\
\quad inform.informing = \{payment\} \\
\quad \bullet inform = isInformPayment(payment)
\end{array}
$$

5.6.11 Accepting Payment

Lastly, once a payment is produced, it is expected that the payment will be accepted by accepting the proposal to discharge the commitment to pay. This acceptance is specified by the axiom *isAcceptToDischargePaying*, which is a function that takes as input an instance of type *Paying*, and returns an acceptance to discharge the paying action.

$$
\begin{array}{|l}
isAcceptToDischargePaying : Paying \rightarrow Accept \\
\hline
\forall paying : Paying \\
\bullet \exists accept : Accept \mid \\
\quad accept.accepting \in Delete \land \\
\quad accept.accepting.commitment = isCommitmentToPay(paying) \\
\quad \bullet accept = isAcceptToDischargePaying(paying)
\end{array}
$$

After introducing in this section various illocutionary point definitions, the next section will show how agents participating in the shopping conversations use these definitions.

5.7 Participants

This example involves three types of participants: a *buyer*, a *seller* and a *carrier*. Subsections below describe the communicational participations of these agents based on the illocutionary points described above.

5.7.1 Buyer

The class *Buyer* (which is shown in Figure 32 and Figure 33) is a subclass of *Agent* that defines operations for requesting to buy books, for paying books, for receiving a proof of purchase and for accepting the delivery of books.

Requesting to Buy Books

The operation *RequestingToBuyBooks* defines the behaviour for requesting the sale of books. This operation is defined as the sequential composition of the operations *ProposeToAdoptSellingBooks* (which is described below) and *SendUtterance* (which was described earlier as an operation that communicates a speech act from a speaker to an addressee).

The operation *ProposeToAdoptSellingBooks* produces a speech act where the speaker and the addressee are the buyer (who is also the current buyer instance) and seller of the provided selling action. In addition, the resulting speech act contains a proposal to adopt selling and an inform indicating the books requested for the sale.

Accepting to Pay Books

The operation *AcceptingToPayBooks* defines the behaviour for accepting to pay for the sale of books. This operation receives a *Paying* action for adoption and evaluates whether or not the buyer (i.e., the current instance) holds obligations to reply to a proposal to adopt the *Paying* action (as specified by the axiom *existsReplyToProposeToAdoptPaying*, which is defined next). The fulfilment of this condition leads to the operations *AcceptToAdoptPayingBooks* (which define a speech act accepting to adopt paying) and *SendUtterance*.

$Buyer$

$\upharpoonright(RequestingToBuyBooks, SubmittingBooksPayment,$
$PayingBooks, AcceptingToEndSale, AcceptingToReceiveBooks)$

$Agent$

$ProposeToAdoptSellingBooks$

$selling? : Selling$
$speechAct! : ToSpeak$

$speechAct!.speaker = selling?.buyer = self \wedge$
$speechAct!.addressee = selling?.seller \wedge$
$isProposeToAdoptSelling(selling?, presently) \in speechAct!.points \wedge$
$isInformBookDescriptions(selling?.items) \in speechAct!.points$

$AcceptToAdoptPayingBooks$

$payingBooks? : Paying$
$speechAct! : ToSpeak$

$speechAct!.speaker = payingBooks?.payer = self \wedge$
$speechAct!.addressee = payingBooks?.payee \wedge$
$isAcceptToAdoptPaying(payingBooks?) \in speechAct!.points$

$ProposeToDischargePayingBooks$

$payingBooks? : Paying$
$payment : Payment$
$speechAct! : ToSpeak$

$payment.currency.value = payingBooks?.invoice.amount$

$speechAct!.speaker = payingBooks?.payer = self \wedge$
$speechAct!.addressee = payingBooks?.payee \wedge$
$isProposeToDischargePaying(payingBooks?, presently) \in speechAct!.points \wedge$
$isInformPayment(payment) \in speechAct!.points$

$AcceptToDischargeSellingBooks$

$selling? : Selling$
$speechAct! : ToSpeak$

$speechAct!.speaker = selling?.buyer = self \wedge$
$speechAct!.addressee = selling?.seller \wedge$
$isAcceptToDischargeSelling(selling?) \in speechAct!.points$

Figure 32. Definition of the class *Buyer* (part 1 of 2).

The axiom *existsReplyToProposeToAdoptPaying* (shown below) assesses whether or not a provided set of obligations contains a *Speaking* action in which the payer is able to reply now to a proposal to adopt a given *Paying* action.

$$
\begin{array}{|l}
\mathit{existsReplyToProposeToAdoptPaying} : \mathbb{P}\ \mathit{Obligation} \times \mathit{Paying} \rightarrow \mathbb{B} \\
\hline
\forall\ \mathit{obligations} : \mathbb{P}\ \mathit{Obligation};\ \mathit{paying} : \mathit{Paying} \\
\bullet\ \mathit{existsReplyToProposeToAdoptPaying}(\mathit{obligations}, \mathit{paying}) \Leftrightarrow \\
\quad (\exists\ \mathit{spoken}, \mathit{replied} : \mathit{Interval} \mid \\
\qquad \mathit{spoken.from} \leq \mathit{now} \leq \mathit{spoken.until} \land \\
\qquad \mathit{replied.from} \leq \mathit{now} \leq \mathit{replied.until} \\
\quad\ \bullet\ \exists\ \mathit{propose} : \downarrow\!\mathit{Propose} \mid \\
\qquad\ \ \mathit{propose} = \mathit{isProposeToAdoptPaying}(\mathit{paying}, \mathit{replied}) \\
\qquad \bullet\ \exists\ o : \mathit{obligations} \\
\qquad\ \ \bullet\ \exists\ \mathit{speaking} : \mathit{Speaking} \mid \\
\qquad\qquad \mathit{speaking.speaker} = \mathit{paying.payer} \land \\
\qquad\qquad \mathit{speaking.addressee} = \mathit{paying.payee} \land \\
\qquad\qquad \mathit{speaking.time} = \mathit{spoken} \land \\
\qquad\qquad \mathit{speaking.points} = \mathit{isReplyTo}(\mathit{propose}) \\
\qquad\quad\ \bullet\ \mathit{speaking} = o)
\end{array}
$$

Submitting Books Payment

The operation *SubmittingBooksPayment* defines the behaviour for producing a payment for the books being bought. This operation receives a *Paying* action for discharge and evaluates whether or not the buyer holds obligations to propose discharging the *Paying* action (as specified by the axiom *existsSpeakToProposeToDischargePaying*, which is defined next). This condition leads to the sequential composition of the operations *ProposeToDischargePayingBooks* (which defines a speech act proposing to discharge a paying action, and informing a payment) and *SendUtterance*.

The axiom *existsSpeakToProposeToDischargePaying* assesses whether or not a provided set of obligations contains a *Speaking* action in which the payer is able to propose now the discharge of a given *Paying* action.

$$\begin{array}{|l}
\textit{AcceptToAdoptNoticeOfDelivery} \\
\hline
\textit{handingOut?} : \textit{HandingOutParcel} \\
\textit{speechAct!} : \textit{ToSpeak} \\
\hline
\textit{speechAct!.speaker} = \textit{handingOut?.receiver} = \textit{self} \land \\
\textit{speechAct!.addressee} = \textit{handingOut?.dispatcher} \land \\
\textit{isAcceptToAdoptHandingOutParcel}(\textit{handingOut?}) \in \textit{speechAct!.points} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\textit{AcceptToDischargeHandingOutParcel} \\
\hline
\textit{handingOut?} : \textit{HandingOutParcel} \\
\textit{speechAct!} : \textit{ToSpeak} \\
\hline
\textit{speechAct!.speaker} = \textit{handingOut?.receiver} = \textit{self} \land \\
\textit{speechAct!.addressee} = \textit{handingOut?.carrier} \land \\
\textit{isAcceptToDischargeHandingOutParcel}(\textit{handingOut?}) \in \textit{speechAct!.points} \\
\hline
\end{array}$$

$$\begin{array}{l}
\textit{RequestingToBuyBooks} \mathrel{\widehat=} \\
\quad \textit{ProposeToAdoptSellingBooks} \mathbin{⨾} \textit{SendUtterance} \\
\textit{AcceptingToPayBooks} \mathrel{\widehat=} [\, \textit{payingBooks?} : \textit{Paying} \mid \\
\quad \textit{existsReplyToProposeToAdoptPaying}(\operatorname{dom} \textit{obligations}, \textit{payingBooks?}) \,] \bullet \\
\quad \textit{AcceptToAdoptPayingBooks} \mathbin{⨾} \textit{SendUtterance} \\
\textit{SubmittingBooksPayment} \mathrel{\widehat=} [\, \textit{payingBooks?} : \textit{Paying} \mid \\
\quad \textit{existsSpeakToProposeToDischargePaying}(\operatorname{dom} \textit{obligations}, \textit{payingBooks?}) \,] \bullet \\
\quad \textit{ProposeToDischargePayingBooks} \mathbin{⨾} \textit{SendUtterance} \\
\textit{PayingBooks} \mathrel{\widehat=} [\, \textit{payingBooks?} : \textit{Paying} \mid \\
\quad \textit{existsReplyToProposeToAdoptPaying}(\operatorname{dom} \textit{obligations}, \textit{payingBooks?}) \,] \bullet \\
\quad (\textit{AcceptToAdoptPayingBooks} \land \\
\quad\; \textit{ProposeToDischargePayingBooks}) \mathbin{⨾} \textit{SendUtterance} \\
\textit{AcceptingToEndSale} \mathrel{\widehat=} [\, \textit{selling?} : \textit{Selling};\ \textit{handingOut?} : \textit{HandingOutParcel} \mid \\
\quad \textit{selling?.seller} = \textit{handingOut?.dispatcher} \land \\
\quad \textit{selling?.items} = \textit{handingOut?.parcel} \land \\
\quad \textit{existsReplyToProposeToDischargeSelling}(\operatorname{dom} \textit{obligations}, \textit{selling?}) \land \\
\quad \textit{existsReplyToProposeToAdoptHandingOutParcel}(\operatorname{dom} \textit{obligations}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{handingOut?}) \,] \bullet \\
\quad (\textit{AcceptToDischargeSellingBooks} \land \\
\quad\; \textit{AcceptToAdoptNoticeOfDelivery}) \mathbin{⨾} \textit{SendUtterance} \\
\textit{AcceptingToReceiveBooks} \mathrel{\widehat=} [\, \textit{handingOut?} : \textit{HandingOutParcel} \mid \\
\quad \textit{existsReplyToProposeToDischargeHandingOutParcel}(\operatorname{dom} \textit{obligations}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{handingOut?}) \,] \bullet \\
\quad \textit{AcceptToDischargeHandingOutParcel} \mathbin{⨾} \textit{SendUtterance}
\end{array}$$

Figure 33. Definition of the class *Buyer* (part 2 of 2).

$$
\begin{array}{l}
\mathit{existsSpeakToProposeToDischargePaying} : \mathbb{P}\ \mathit{Obligation} \times \mathit{Paying} \rightarrow \mathbb{B} \\
\hline
\forall\ \mathit{obligations} : \mathbb{P}\ \mathit{Obligation};\ \mathit{paying} : \mathit{Paying} \\
\bullet\ \mathit{existsSpeakToProposeToDischargePaying}(\mathit{obligations}, \mathit{paying}) \Leftrightarrow \\
\quad (\exists\ \mathit{spoken}, \mathit{replied} : \mathit{Interval} \mid \\
\qquad \mathit{spoken.from} \leq \mathit{now} \leq \mathit{spoken.until} \wedge \\
\qquad \mathit{replied.from} \leq \mathit{now} \leq \mathit{replied.until} \\
\quad\ \bullet\ \exists\ o : \mathit{obligations} \\
\qquad \bullet\ \exists\ \mathit{speaking} : \mathit{Speaking} \mid \\
\qquad\quad \mathit{speaking.speaker} = \mathit{paying.payee} \wedge \\
\qquad\quad \mathit{speaking.addressee} = \mathit{paying.payer} \wedge \\
\qquad\quad \mathit{speaking.time} = \mathit{spoken} \wedge \\
\qquad\quad \mathit{isProposeToDischargePaying}(\mathit{paying}, \mathit{replied}) \in \mathit{speaking.points} \\
\qquad\ \bullet\ \mathit{speaking} = o)
\end{array}
$$

Paying Books

In some contexts, the uttering of two separate speech acts for accepting to pay and for producing a payment may be neither adequate nor practical. In such cases both the acceptance to pay and a payment could be communicated in a single utterance.

This is modelled by the operation *PayingBooks*. This operation evaluates whether or not the buyer holds an obligation to reply to a proposal to adopt paying (as indicated by the previously defined axiom *existsReplyToProposeToAdoptPaying*), which leads to the operations *AcceptToAdoptPayingBooks* and *ProposeToDischargePayingBooks* (also described earlier), which specify an acceptance to pay and the production of payment, respectively.

Accepting to End Sale

The operation *AcceptingToEndSale* specifies the behaviour for accepting to discharge a commitment to sell, and for accepting to adopt a commitment in which a carrier hands out a parcel of books. This operation receives a *Selling* action for discharge and a *HandingOutParcel* action for adoption, and evaluates whether or not these actions specify the same agent as their seller and sender, and the same data as the items being sold and delivered, respectively. This operation also evaluates whether or not the buyer holds an bligation to reply to a proposal to discharge the *Selling* action, and an obligation to propose the adoption of the *HandingOutParcel* action (as specified by the axioms *existsReplyTo-ProposeToDischargeSelling* and *existsReplyToProposeToAdoptHandingOutParcel*, which are defined next). The fulfilment of these conditions leads to the operations *AcceptToDischargeSellingBooks* and *AcceptToAdoptNoticeOfDelivery* (which define speech acts for accepting to discharge selling books, and for accepting to adopt a commitment in which the carrier hands out to the buyer the books he bought) followed by the operation *SendUtterance*.

The axiom *existsReplyToProposeToDischargeSelling* assesses whether or not a provided set of obligations contains a *Speaking* action in which the buyer is able to reply now to a proposal to discharge a *Selling* action.

$$
\begin{array}{|l}
existsReplyToProposeToDischargeSelling : \mathbb{P}\ Obligation \times Selling \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ selling : Selling \\
\bullet\ existsReplyToProposeToDischargeSelling(obligations, selling) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval \mid \\
\quad\quad spoken.from \leq now \leq spoken.until \wedge \\
\quad\quad replied.from \leq now \leq replied.until \\
\quad\ \bullet\ \exists\ propose : \downarrow Propose \mid \\
\quad\quad\ propose = isProposeToDischargeSelling(selling, replied) \\
\quad\quad \bullet\ \exists\ o : obligations \\
\quad\quad\ \bullet\ \exists\ speaking : Speaking \mid \\
\quad\quad\quad\quad speaking.speaker = selling.seller \wedge \\
\quad\quad\quad\quad speaking.addressee = selling.buyer \wedge \\
\quad\quad\quad\quad speaking.time = spoken \wedge \\
\quad\quad\quad\quad speaking.points = isReplyTo(propose) \\
\quad\quad\quad \bullet\ speaking = o)
\end{array}
$$

Likewise, the axiom *existsReplyToProposeToAdoptHandingOutParcel* (shown below) assesses whether or not a provided set of obligations contains a *Speaking* action in which the buyer is able to reply now to a proposal to adopt a *HandingOutParcel* action.

$$
\begin{array}{|l}
existsReplyToProposeToAdoptHandingOutParcel : \\
\qquad\qquad\qquad\qquad \mathbb{P}\, Obligation \times HandingOutParcel \rightarrow \mathbb{B} \\
\hline
\forall\, obligations : \mathbb{P}\, Obligation;\ handingOut : HandingOutParcel \\
\bullet\ existsReplyToProposeToAdoptHandingOutParcel(obligations, handingOut) \Leftrightarrow \\
\quad (\exists\, spoken, replied : Interval \mid \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad\ \bullet\ \exists\, propose : \downarrow Propose \mid \\
\qquad\quad propose = isProposeToAdoptHandingOutParcel(handingOut, replied) \\
\qquad \bullet\ \exists\, o : obligations \\
\qquad\ \ \bullet\ \exists\, speaking : Speaking \mid \\
\qquad\qquad speaking.speaker = handingOut.receiver \land \\
\qquad\qquad speaking.addressee = handingOut.carrier \land \\
\qquad\qquad speaking.time = spoken \land \\
\qquad\qquad speaking.points = isReplyTo(propose) \\
\qquad\quad \bullet\ speaking = o)
\end{array}
$$

Accepting to Receive Books

Lastly, the operation *AcceptingToReceiveBooks* defines the behaviour for accepting to come into possession of the books delivered by the carrier. This operation receives a *HandingOutParcel* action for discharge, and evaluates whether or not the buyer holds an obligation to reply to a proposal to discharge the *HandingOutParcel* action (as specified by the axiom *existsReplyToProposeToDischargeHandingOutParcel*, which is defined next). The fulfilment of this condition then leads to the operations *AcceptToDischargeHandingOutParcel* (which defines a speech act accepting to discharge the handing out of a parcel) and *SendUtterance*.

The axiom *existsReplyToProposeToDischargeHandingOutParcel* (shown below) is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the buyer is able to reply now to a proposal to discharge a *HandingOutParcel* action.

$$
\begin{array}{l}
\mathit{existsReplyToProposeToDischargeHandingOutParcel} : \\
\qquad\qquad\qquad \mathbb{P}\ \mathit{Obligation} \times \mathit{HandingOutParcel} \rightarrow \mathbb{B} \\
\hline
\forall\ \mathit{obligations} : \mathbb{P}\ \mathit{Obligation};\ \mathit{handingOut} : \mathit{HandingOutParcel} \\
\bullet\ \mathit{existsReplyToProposeToDischargeHandingOutParcel}(\mathit{obligations}, \mathit{handingOut}) \Leftrightarrow \\
\quad (\exists\ \mathit{spoken}, \mathit{replied} : \mathit{Interval} \mid \\
\qquad \mathit{spoken.from} \leq \mathit{now} \leq \mathit{spoken.until} \land \\
\qquad \mathit{replied.from} \leq \mathit{now} \leq \mathit{replied.until} \\
\quad\ \bullet\ \exists\ \mathit{propose} : \downarrow \mathit{Propose} \mid \\
\qquad\quad \mathit{propose} = \mathit{isProposeToDischargeHandingOutParcel}(\mathit{handingOut}, \mathit{replied}) \\
\qquad \bullet\ \exists\ o : \mathit{obligations} \\
\qquad\quad \bullet\ \exists\ \mathit{speaking} : \mathit{Speaking} \mid \\
\qquad\qquad\quad \mathit{speaking.speaker} = \mathit{handingOut.receiver} \land \\
\qquad\qquad\quad \mathit{speaking.addressee} = \mathit{handingOut.carrier} \land \\
\qquad\qquad\quad \mathit{speaking.time} = \mathit{spoken} \land \\
\qquad\qquad\quad \mathit{speaking.points} = \mathit{isReplyTo}(\mathit{propose}) \\
\qquad\qquad \bullet\ \mathit{speaking} = o)
\end{array}
$$

The next section describes the second (of the three) agents involved in the *eBookstore* interactions: the seller agent.

5.7.2 Seller

Sellers are those agents that sell books to buyers and who request the delivery of these books to carrier agents.

The class *Seller* (which is shown in Figure 34, Figure 35 and Figure 36) is a class inheriting from *Agent* which defines operations for accepting to sell books, requesting to pay books, accepting books payment, proposing to end a sale, requesting the delivery of books, and accepting to pay their delivery.

Accepting to Sell Books

The operation *AcceptingToSellBooks* specifies the behaviour for accepting a proposal to adopt a commitment to sell books. This operation receives a *Selling* action for adoption and evaluates whether or not the seller (i.e., the current instance) holds an obligation to reply to a proposal to adopt the action (as specified by the axiom *existsReplyToProposeToAdoptSelling*, which is defined next). The fulfilment of this condition leads to the operations *AcceptToAdoptSellingBooks* (which defines a speech act accepting to sell books) and *SendUtterance*.

The axiom *existsReplyToProposeToAdoptSelling* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the bidder is able to reply now to a proposal to adopt a given *Selling* action.

$$
\begin{array}{l}
existsReplyToProposeToAdoptSelling : \mathbb{P}\ Obligation \times Selling \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ selling : Selling \\
\bullet\ existsReplyToProposeToAdoptSelling(obligations, selling) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval \mid \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad\ \bullet\ \exists\ propose : \downarrow Propose \mid \\
\qquad\ propose = isProposeToAdoptSelling(selling, replied) \\
\qquad \bullet\ \exists\ o : obligations \\
\qquad\ \ \bullet\ \exists\ speaking : Speaking \mid \\
\qquad\qquad speaking.speaker = selling.seller \land \\
\qquad\qquad speaking.addressee = selling.buyer \land \\
\qquad\qquad speaking.time = spoken \land \\
\qquad\qquad speaking.points = isReplyTo(propose) \\
\qquad\quad \bullet\ speaking = o)
\end{array}
$$

Requesting Books Payment

The operation *RequestingBooksPayment* specifies the behaviour of a seller when proposing that she is paid for selling a set of requested books. This operation receives a *Paying* action and evaluates whether or not the seller holds an obligation to propose to the buyer its adoption (as specified by the axiom *existsSpeakToProposeToAdoptPaying*, which is defined next). The fulfilment of this condition leads to the operations *ProposeToAdoptPayingBooks* (which defines a speech act proposing to pay for the books requested) and *SendUtterance*.

$Seller$

$\upharpoonright(AcceptingToSellBooks, RequestingBooksPayment, AcceptingBooksPayment,$
$ProposingToEndSale, ProposingToEndTransaction, RequestingToDeliverBooks,$
$AcceptingToPayDelivery, SubmittingDeliveryPayment, PayingDelivery)$

$Agent$

$AcceptToAdoptSellingBooks$

$selling? : Selling$
$speechAct! : ToSpeak$

$speechAct!.speaker = selling?.seller = self \wedge$
$speechAct!.addressee = selling?.buyer \wedge$
$isAcceptToAdoptSelling(selling?) \in speechAct!.points$

$ProposeToAdoptPayingBooks$

$payingBooks? : Paying$
$speechAct! : ToSpeak$

$speechAct!.speaker = payingBooks?.payee = self \wedge$
$speechAct!.addressee = payingBooks?.payer \wedge$
$isProposeToAdoptPaying(payingBooks?, presently) \in speechAct!.points \wedge$
$isInformInvoice(payingBooks?.invoice) \in speechAct!.points$

$AcceptToDischargePayingBooks$

$payingBooks? : Paying$
$speechAct! : ToSpeak$

$speechAct!.speaker = payingBooks?.payee = self \wedge$
$speechAct!.addressee = payingBooks?.payer \wedge$
$isAcceptToDischargePaying(payingBooks?) \in speechAct!.points$

$ProposeToDischargeSellingBooks$

$selling? : Selling$
$receipt : Receipt$
$speechAct! : ToSpeak$

$speechAct!.speaker = selling?.seller = self \wedge$
$speechAct!.addressee = selling?.buyer \wedge$
$isProposeToDischargeSelling(selling?, presently) \in speechAct!.points \wedge$
$isInformReceipt(receipt) \in speechAct!.points$

Figure 34. Definition of the class *Seller* (part 1 of 3).

The axiom *existsSpeakToProposeToAdoptPaying* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the seller is able to propose now to discharge a given *Paying* action.

$$
\begin{array}{|l}
existsSpeakToProposeToAdoptPaying : \mathbb{P}\ Obligation \times Paying \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ paying : Paying \\
\bullet\ existsSpeakToProposeToAdoptPaying(obligations, paying) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval\ | \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad \bullet\ \exists\ o : obligations \\
\qquad \bullet\ \exists\ speaking : Speaking\ | \\
\qquad\quad speaking.speaker = paying.payee \land \\
\qquad\quad speaking.addressee = paying.payer \land \\
\qquad\quad speaking.time = spoken \land \\
\qquad\quad isProposeToAdoptPaying(paying, replied) \in speaking.points \\
\qquad\quad \bullet\ speaking = o)
\end{array}
$$

Accepting Books Payment

The operation *AcceptingBooksPayment* specifies the behaviour for accepting to discharge a commitment to pay. This operation receives a *Paying* action for discharge, and evaluates whether or not the seller holds an obligation to reply to a proposal to discharge paying the books (as specified by the axiom *existsReplyToProposeToDischargePaying*, which is defined next). The fulfilment of this condition leads to the operations *AcceptToDischargePayingBooks* (which defines a speech act accepting the discharge of paying the books) and *SendUtterance*.

The axiom *existsReplyToProposeToDischargePaying* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the seller can propose the discharge of a given *Paying* action.

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{ProposeToAdoptNoticeOfDelivery}} \\
\textit{handingOut}? : \textit{HandingOutParcel} \\
\textit{speechAct}! : \textit{ToSpeak} \\
\hline
\textit{speechAct}!.\textit{speaker} = \textit{handingOut}?.\textit{dispatcher} = \textit{self} \wedge \\
\textit{speechAct}!.\textit{addressee} = \textit{handingOut}?.\textit{receiver} \wedge \\
\textit{isProposeToAdoptHandingOutParcel}(\textit{handingOut}?, \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{presently}) \in \textit{speechAct}!.\textit{points} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{ProposeToAdoptDelivery}} \\
\textit{delivering}? : \textit{Delivering} \\
\textit{books} : \mathbb{P}_1\ \textit{Book} \\
\textit{speechAct}! : \textit{ToSpeak} \\
\hline
\forall\ \textit{description} : \downarrow\textit{BookDescription} \mid \\
\quad \textit{description} \in \textit{delivering}?.\textit{parcel} \\
\bullet\ \exists\ \textit{book} : \textit{books} \\
\quad \bullet\ \textit{book}.\textit{title} = \textit{description}.\textit{title} \\
\textit{speechAct}!.\textit{speaker} = \textit{delivering}?.\textit{sender} = \textit{self} \wedge \\
\textit{speechAct}!.\textit{addressee} = \textit{delivering}?.\textit{carrier} \wedge \\
\textit{isProposeToAdoptDelivering}(\textit{delivering}?, \textit{presently}) \in \textit{speechAct}!.\textit{points} \wedge \\
\textit{isInformBooks}(\textit{books}) \in \textit{speechAct}!.\textit{points} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{AcceptToAdoptPayingDelivery}} \\
\textit{payingDelivery}? : \textit{Paying} \\
\textit{speechAct}! : \textit{ToSpeak} \\
\hline
\textit{speechAct}!.\textit{speaker} = \textit{payingDelivery}?.\textit{payer} = \textit{self} \wedge \\
\textit{speechAct}!.\textit{addressee} = \textit{payingDelivery}?.\textit{payee} \wedge \\
\textit{isAcceptToAdoptPaying}(\textit{payingDelivery}?) \in \textit{speechAct}!.\textit{points} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{ProposeToDischargePayingDelivery}} \\
\textit{payingDelivery}? : \textit{Paying} \\
\textit{payment} : \textit{Payment} \\
\textit{speechAct}! : \textit{ToSpeak} \\
\hline
\textit{payment}.\textit{currency}.\textit{value} = \textit{payingDelivery}?.\textit{invoice}.\textit{amount} \\
\textit{speechAct}!.\textit{speaker} = \textit{payingDelivery}?.\textit{payer} = \textit{self} \wedge \\
\textit{speechAct}!.\textit{addressee} = \textit{payingDelivery}?.\textit{payee} \wedge \\
\textit{isProposeToDischargePaying}(\textit{payingDelivery}?, \textit{presently}) \in \textit{speechAct}!.\textit{points} \wedge \\
\textit{isInformPayment}(\textit{payment}) \in \textit{speechAct}!.\textit{points} \\
\hline
\end{array}$$

Figure 35. Definition of the class *Seller* (part 2 of 3).

$$
\begin{array}{|l}
existsReplyToProposeToDischargePaying : \mathbb{P}\ Obligation \times Paying \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ paying : Paying \\
\bullet\ existsReplyToProposeToDischargePaying(obligations, paying) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval \mid \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad\ \bullet\ \exists\ propose : \downarrow Propose \mid \\
\qquad\ propose = isProposeToDischargePaying(paying, replied) \\
\qquad \bullet\ \exists\ o : obligations \\
\qquad\ \ \bullet\ \exists\ speaking : Speaking \mid \\
\qquad\qquad speaking.speaker = paying.payer \land \\
\qquad\qquad speaking.addressee = paying.payee \land \\
\qquad\qquad speaking.time = spoken \land \\
\qquad\qquad speaking.points = isReplyTo(propose) \\
\qquad\quad \bullet\ speaking = o)
\end{array}
$$

Proposing to End Sale

The operation *ProposingToEndSale* specifies the behaviour for finalizing books' sales. This operation receives a *Selling* action and a *HandingOutParcel* action, and evaluates

a) whether or not these actions specify the same agent as the buyer of books and the receiver of the parcel (respectively),

b) whether or not the books bought are the same as those delivered in the parcel, and

c) whether or not the seller holds an obligation to propose discharging the *Selling* action (as specified by the axiom *existsSpeakToProposeToDischargeSelling*, which is defined next).

The fulfilment of these conditions leads to the conjunctive composition of the operations *ProposeToDischargeSellingBooks* and *ProposeToAdoptNoticeOfDelivery* (which define speech acts for proposing to discharge selling books, and for proposing to adopt a commitment in which the carrier hands out the books to the buyer) followed by the operation *SendUtterance*.

$$
\begin{array}{l}
\mathit{AcceptingToSellBooks} \mathrel{\widehat=} [\, \mathit{selling?} : \mathit{Selling} \mid \\
\quad \mathit{existsReplyToProposeToAdoptSelling}(\mathrm{dom}\ \mathit{obligations}, \mathit{selling?}) \,] \bullet \\
\quad \mathit{AcceptToAdoptSellingBooks} \mathbin{\S} \mathit{SendUtterance} \\
\mathit{RequestingBooksPayment} \mathrel{\widehat=} [\, \mathit{payingBooks?} : \mathit{Paying} \mid \\
\quad \mathit{existsSpeakToProposeToAdoptPaying}(\mathrm{dom}\ \mathit{obligations}, \mathit{payingBooks?}) \,] \bullet \\
\quad \mathit{ProposeToAdoptPayingBooks} \mathbin{\S} \mathit{SendUtterance} \\
\mathit{AcceptingBooksPayment} \mathrel{\widehat=} [\, \mathit{payingBooks?} : \mathit{Paying} \mid \\
\quad \mathit{existsReplyToProposeToDischargePaying}(\mathrm{dom}\ \mathit{obligations}, \mathit{payingBooks?}) \,] \bullet \\
\quad \mathit{AcceptToDischargePayingBooks} \mathbin{\S} \mathit{SendUtterance} \\
\mathit{ProposingToEndSale} \mathrel{\widehat=} [\, \mathit{selling?} : \mathit{Selling};\ \mathit{handingOut?} : \mathit{HandingOutParcel} \mid \\
\quad \mathit{selling?.buyer} = \mathit{handingOut?.receiver} \land \\
\quad \mathit{selling?.items} = \mathit{handingOut?.parcel} \land \\
\quad \mathit{existsSpeakToProposeToDischargeSelling}(\mathrm{dom}\ \mathit{obligations}, \mathit{selling?}) \,] \bullet \\
\quad (\mathit{ProposeToDischargeSellingBooks} \land \\
\quad\ \mathit{ProposeToAdoptNoticeOfDelivery}) \mathbin{\S} \mathit{SendUtterance} \\
\mathit{ProposingToEndTransaction} \mathrel{\widehat=} [\, \mathit{payingBooks?} : \mathit{Paying}; \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{selling?} : \mathit{Selling}; \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{handingOut?} : \mathit{HandingOutParcel} \mid \\
\quad \mathit{payingBooks?.payer} = \mathit{selling?.buyer} = \mathit{handingOut?.receiver} \land \\
\quad \mathit{selling?.items} = \mathit{handingOut?.parcel} \land \\
\quad \mathit{existsReplyToProposeToDischargePaying}(\mathrm{dom}\ \mathit{obligations}, \mathit{payingBooks?}) \land \\
\quad \mathit{existsSpeakToProposeToDischargeSelling}(\mathrm{dom}\ \mathit{obligations}, \mathit{selling?}) \,] \bullet \\
\quad (\mathit{AcceptToDischargePayingBooks} \land \\
\quad\ \mathit{ProposeToDischargeSellingBooks} \land \\
\quad\ \mathit{ProposeToAdoptNoticeOfDelivery}) \mathbin{\S} \mathit{SendUtterance} \\
\mathit{RequestingToDeliverBooks} \mathrel{\widehat=} \\
\quad \mathit{ProposeToAdoptDelivery} \mathbin{\S} \mathit{SendUtterance} \\
\mathit{AcceptingToPayDelivery} \mathrel{\widehat=} [\, \mathit{payingDelivery?} : \mathit{Paying} \mid \\
\quad \mathit{existsReplyToProposeToAdoptPaying}(\mathrm{dom}\ \mathit{obligations}, \mathit{payingDelivery?}) \,] \bullet \\
\quad \mathit{AcceptToAdoptPayingDelivery} \mathbin{\S} \mathit{SendUtterance} \\
\mathit{SubmittingDeliveryPayment} \mathrel{\widehat=} [\, \mathit{payingDelivery?} : \mathit{Paying} \mid \\
\quad \mathit{existsSpeakToProposeToDischargePaying}(\mathrm{dom}\ \mathit{obligations}, \mathit{payingDelivery?}) \,] \bullet \\
\quad \mathit{ProposeToDischargePayingDelivery} \mathbin{\S} \mathit{SendUtterance} \\
\mathit{PayingDelivery} \mathrel{\widehat=} [\, \mathit{payingDelivery?} : \mathit{Paying} \mid \\
\quad \mathit{existsReplyToProposeToAdoptPaying}(\mathrm{dom}\ \mathit{obligations}, \mathit{payingDelivery?}) \,] \bullet \\
\quad (\mathit{AcceptToAdoptPayingDelivery} \land \\
\quad\ \mathit{ProposeToDischargePayingDelivery}) \mathbin{\S} \mathit{SendUtterance}
\end{array}
$$

Figure 36. Definition of the class *Seller* (part 3 of 3).

The axiom *existsSpeakToProposeToDischargeSelling* assesses whether or not a provided set of obligations contains a *Speaking* action in which the seller can propose now to discharge a given *Selling* action.

$$
\begin{array}{l}
existsSpeakToProposeToDischargeSelling : \mathbb{P}\ Obligation \times Selling \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ selling : Selling \\
\bullet\ existsSpeakToProposeToDischargeSelling(obligations, selling) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval \mid \\
\quad\quad spoken.from \leq now \leq spoken.until \land \\
\quad\quad replied.from \leq now \leq replied.until \\
\quad \bullet\ \exists\ o : obligations \\
\quad\quad \bullet\ \exists\ speaking : Speaking \mid \\
\quad\quad\quad speaking.speaker = selling.seller \land \\
\quad\quad\quad speaking.addressee = selling.buyer \land \\
\quad\quad\quad speaking.time = spoken \land \\
\quad\quad\quad isProposeToDischargeSelling(selling, replied) \in speaking.points \\
\quad\quad\quad \bullet\ speaking = o)
\end{array}
$$

Proposing to End Transaction

The operation *ProposingToEndTransaction* specifies in one operation (thus one communication) the acceptance of a payment and the end of a sale (behaviours that were defined separately in the two previous sections by the operations *AcceptingBooksPayment* and *ProposingToEndSale*).

As shown in Figure 36, this operation evaluates whether or not the given *Paying*, *Selling* and *HandingOutParcel* actions have the same agent as their payer, buyer and receiver (respectively), whether or not the books being sold are the same as those in the parcel to be delivered, and whether or not the seller holds an obligation in which she replies to a proposal to discharge paying, and an obligation in which she proposes to discharge the selling of books (as indicated by the previously defined axioms *existsReplyToProposeToDischargePaying* and *existsSpeakToProposeToDischargeSelling*). The fulfilment of these conditions leads to the conjunctive composition of the operations *AcceptToDischargePayingBooks*, *ProposeToDischargeSellingBooks* and

ProposeToAdoptNoticeOfDelivery (which were also described earlier) followed by the operation *SendUtterance*.

Requesting to Deliver Books

The operation *RequestingToDeliverBooks* specifies the behaviour for requesting to a carrier the delivery of books. This operation defines the sequential composition of the operations *ProposeToAdoptDelivery* (which defines a speech act for proposing to adopt a commitment to deliver a set of books) and *SendUtterance*.

Accepting to Pay Delivery

Since the action *Delivering* is a subclass of *Payable*, then proposing to adopt this action enables the carrier to propose a payment. The operation *AcceptingToPayDelivery* defines the behaviour for accepting to pay for a delivery. This operation receives a *Paying* action for adoption, and evaluates whether or not the seller holds an obligation to reply to a proposal to adopt paying (as specified by the previously defined axiom *existsReplyToProposeToAdoptPaying*). The fulfilment of this condition leads to the operations *AcceptToAdoptPayingDelivery* (which defines a speech act accepting to pay for the delivery of books) and *SendUtterance*.

Submitting Delivery Payment

The operation *SubmittingDeliveryPayment* specifies the behaviour for submitting a payment for the delivery of books. This operation receives a Paying action for discharge, and evaluates whether or not the seller holds an obligation to propose the discharge of a commitment to pay the delivery of books (as specified by the previously defined axiom *existsSpeakToProposeToDischargePaying*). The fulfilment of this condition leads to the operations *ProposeToDischargePayingDelivery* (which defines a speech act proposing to discharge paying for the delivery) and *SendUtterance*.

Paying Delivery

Lastly, the operation *PayingDelivery* specifies in one operation the behaviours for accepting to pay and producing a payment for the delivery (as indicated by the previously defined operations *AcceptingToPayDelivery* and *SubmittingDeliveryPayment*).

As shown, this operation evaluates whether or not the seller holds an obligation to reply to a proposal to pay for the requested delivery of books (as specified by the previously defined axiom *existsReplyToProposeToAdoptPaying*), which leads to the operations *AcceptToAdoptPayingDelivery* and *ProposeToDischargePayingDelivery* (which were also described earlier), followed by the operation *SendUtterance*.

To recap, this section described the behaviour of seller agents. The next section describes the operations making the behaviour of the last type of agent in the *eBookstore* scenario: that of carrier agents.

5.7.3 Carrier

Carrier agents are those that perform deliveries between the seller and the buyer.

The class *Carrier* (which is shown in Figure 37 and Figure 38) is a subclass of *Agent* that defines operations for requesting and accepting delivery payments, for accepting to do deliveries and for handing out a parcel delivered. These operations (along with other private operations that support them) are described in the subsections below.

Requesting a Delivery Payment

The operation *RequestingDeliveryPayment* defines the behaviour for requesting the payment of a delivery. This operation evaluates whether or not the carrier (i.e., the current instance) holds obligations to propose to the seller that she pay for a delivery (as specified by the previously defined axiom *existsSpeakToProposeToAdoptPaying*), which leads to the operations *ProposeToAdoptPayingDelivery* (which defines a speech act proposing the adoption of a commitment in which the seller pays a delivery) and *SendUtterance*.

Carrier

$\upharpoonright(RequestingDeliveryPayment, AcceptingDeliveryPayment,$
$AcceptingToDeliverBooks, AcceptingBooksDelivery, RequestingToReceiveBooks)$

Agent

ProposeToAdoptPayingDelivery

$payingDelivery? : Paying$
$speechAct! : ToSpeak$

$speechAct!.speaker = payingDelivery?.payee = self \wedge$
$speechAct!.addressee = payingDelivery?.payer \wedge$
$isProposeToAdoptPaying(payingDelivery?, presently) \in speechAct!.points$

AcceptToDischargePayingDelivery

$payingDelivery? : Paying$
$speechAct! : ToSpeak$

$speechAct!.speaker = payingDelivery?.payee = self \wedge$
$speechAct!.addressee = payingDelivery?.payer \wedge$
$isAcceptToDischargePaying(payingDelivery?) \in speechAct!.points$

AcceptToAdoptDelivering

$delivering? : Delivering$
$speechAct! : ToSpeak$

$speechAct!.speaker = delivering?.carrier = self \wedge$
$speechAct!.addressee = delivering?.sender \wedge$
$isAcceptToAdoptDelivering(delivering?) \in speechAct!.points$

ProposeToDischargeHandingOutParcel

$handingOut? : HandingOutParcel$
$books : \mathbb{P}_1\, Book$
$speechAct! : ToSpeak$

$\forall\, description : \downarrow BookDescription \mid$
$\quad description \in handingOut?.parcel$
$\bullet\ \exists\, book : books$
$\quad \bullet\ book.title = description.title$

$speechAct!.speaker = handingOut?.carrier = self \wedge$
$speechAct!.addressee = handingOut?.receiver \wedge$
$isProposeToDischargeHandingOutParcel(handingOut?,$
$\qquad presently) \in speechAct!.points \wedge$
$isInformBooks(books) \in speechAct!.points$

Figure 37. Definition of the class *Carrier* (part 1 of 2).

$$
\begin{array}{l}
\mathit{RequestingDeliveryPayment} \mathrel{\widehat=} [\, \mathit{payingDelivery?} : \mathit{Paying} \mid \\
\quad \mathit{existsSpeakToProposeToAdoptPaying}(\operatorname{dom} \mathit{obligations}, \mathit{payingDelivery?}) \,] \bullet \\
\quad \mathit{ProposeToAdoptPayingDelivery} \mathbin{⨾} \mathit{SendUtterance} \\
\mathit{AcceptingDeliveryPayment} \mathrel{\widehat=} [\, \mathit{payingDelivery?} : \mathit{Paying} \mid \\
\quad \mathit{existsReplyToProposeToDischargePaying}(\operatorname{dom} \mathit{obligations}, \mathit{payingDelivery?}) \,] \bullet \\
\quad \mathit{AcceptToDischargePayingDelivery} \mathbin{⨾} \mathit{SendUtterance} \\
\mathit{AcceptingToDeliverBooks} \mathrel{\widehat=} [\, \mathit{delivering?} : \mathit{Delivering} \mid \\
\quad \mathit{existsReplyToProposeToAdoptDelivering}(\operatorname{dom} \mathit{obligations}, \mathit{delivering?}) \,] \bullet \\
\quad \mathit{AcceptToAdoptDelivering} \mathbin{⨾} \mathit{SendUtterance} \\
\mathit{AcceptingBooksDelivery} \mathrel{\widehat=} [\, \mathit{payingDelivery?} : \mathit{Paying};\ \mathit{delivering?} : \mathit{Delivering} \mid \\
\quad \mathit{payingDelivery?}.\mathit{payer} = \mathit{delivering?}.\mathit{sender} \land \\
\quad \mathit{existsReplyToProposeToDischargePaying}(\operatorname{dom} \mathit{obligations}, \mathit{payingDelivery?}) \land \\
\quad \mathit{existsReplyToProposeToAdoptDelivering}(\operatorname{dom} \mathit{obligations}, \mathit{delivering?}) \,] \bullet \\
\quad (\mathit{AcceptToDischargePayingDelivery} \land \\
\quad\ \mathit{AcceptToAdoptDelivering}) \mathbin{⨾} \mathit{SendUtterance} \\
\mathit{RequestingToReceiveBooks} \mathrel{\widehat=} [\, \mathit{handingOut?} : \mathit{HandingOutParcel} \mid \\
\quad \mathit{existsSpeakToProposeToDischargeHandingOutParcel}(\operatorname{dom} \mathit{obligations}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{handingOut?}) \,] \bullet \\
\quad \mathit{ProposeToDischargeHandingOutParcel} \mathbin{⨾} \mathit{SendUtterance}
\end{array}
$$

Figure 38. Definition of the class *Carrier* (part 2 of 2).

Accepting a Delivery Payment

The operation *AcceptingDeliveryPayment* defines the behaviour for accepting a submitted payment. This operation evaluates whether or not the carrier holds obligations for replying to a proposal to discharge paying (as specified by the previously defined axiom *existsReplyToProposeToDischargePaying*), which leads to the operations *AcceptToDischargePayingDelivery* (which defines a speech act accepting to discharge the commitment to pay a delivery) and *SendUtterance*.

Accepting to Deliver

The operation *AcceptingToDeliverBooks* defines the behaviour for accepting to deliver a parcel of books. This operation evaluates whether or not the carrier holds obligations to reply to a proposal to adopt committing to make a delivery (as specified by the axiom *existsReplyToProposeToAdoptDelivering*, which is defined next), which leads to the operations *AcceptToAdoptDelivering* (which defines a speech act accepting to deliver) and *SendUtterance*.

The axiom *existsReplyToProposeToAdoptDelivering* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the carrier is able to reply now to a proposal to adopt a given *Delivering* action.

$$
\begin{array}{|l}
\mathit{existsReplyToProposeToAdoptDelivering} : \mathbb{P}\ \mathit{Obligation} \times \mathit{Delivering} \rightarrow \mathbb{B} \\
\hline
\forall\ \mathit{obligations} : \mathbb{P}\ \mathit{Obligation};\ \mathit{delivering} : \mathit{Delivering} \\
\bullet\ \mathit{existsReplyToProposeToAdoptDelivering}(\mathit{obligations}, \mathit{delivering}) \Leftrightarrow \\
\quad (\exists\ \mathit{spoken}, \mathit{replied} : \mathit{Interval} \mid \\
\qquad \mathit{spoken.from} \leq \mathit{now} \leq \mathit{spoken.until} \land \\
\qquad \mathit{replied.from} \leq \mathit{now} \leq \mathit{replied.until} \\
\quad\ \bullet\ \exists\ \mathit{propose} : \downarrow\mathit{Propose} \mid \\
\qquad\ \mathit{propose} = \mathit{isProposeToAdoptDelivering}(\mathit{delivering}, \mathit{replied}) \\
\qquad \bullet\ \exists\ o : \mathit{obligations} \\
\qquad\ \ \bullet\ \exists\ \mathit{speaking} : \mathit{Speaking} \mid \\
\qquad\qquad \mathit{speaking.speaker} = \mathit{delivering.carrier} \land \\
\qquad\qquad \mathit{speaking.addressee} = \mathit{delivering.sender} \land \\
\qquad\qquad \mathit{speaking.time} = \mathit{spoken} \land \\
\qquad\qquad \mathit{speaking.points} = \mathit{isReplyTo}(\mathit{propose}) \\
\qquad\quad\ \bullet\ \mathit{speaking} = o)
\end{array}
$$

Accepting Books Delivery

The operation *AcceptingBooksDelivery* defines the behaviour for accepting a payment and accepting to deliver, thus combining the behaviour previously specified by the operations *AcceptingDeliveryPayment* and *AcceptingToDeliverBooks*.

This operation receives a *Paying* action for discharge and a *Delivering* action for adoption, and evaluates whether or not these actions specify the same agent as their payer and sender (respectively), and whether or not the carrier holds obligations to reply to a proposal to discharge paying, and a proposal to deliver (as specified by the previously defined axioms *existsReplyToProposeToDischargePaying* and *existsReplyToProposeToAdoptDelivering*). The fulfilment of these conditions leads to the operations *AcceptToDischarge-*

PayingDelivery and *AcceptToAdoptDelivering* (which were also defined earlier) followed by the operation *SendUtterance*.

Handing Out a Delivery

Lastly, the operation *RequestingToReceiveBooks* indicates the behaviour followed by the carrier for handing out the parcel to the buyer. This operation receives a *HandingOutParcel* action for discharge, and evaluates whether or not the carrier holds an obligation to propose the discharge of the action (as specified by the axiom *existsSpeakToProposeToDischargeHandingOutParcel*, which is defined next), which leads to the operations *ProposeToDischargeHandingOutParcel* (which defines a speech act proposing to hand out books) and *SendUtterance*.

The axiom *existsSpeakToProposeToDischargeHandingOutParcel* is a function that assesses whether or not a provided set of obligations contains a *Speaking* action in which the carrier is able to propose now to discharge a given *HandingOutParcel* action.

$$
\begin{array}{|l}
existsSpeakToProposeToDischargeHandingOutParcel : \\
\qquad\qquad\qquad\qquad \mathbb{P}\ Obligation \times HandingOutParcel \rightarrow \mathbb{B} \\
\hline
\forall\ obligations : \mathbb{P}\ Obligation;\ handingOut : HandingOutParcel \\
\bullet\ existsSpeakToProposeToDischargeHandingOutParcel(obligations, handingOut) \Leftrightarrow \\
\quad (\exists\ spoken, replied : Interval \mid \\
\qquad spoken.from \leq now \leq spoken.until \land \\
\qquad replied.from \leq now \leq replied.until \\
\quad \bullet\ \exists\ o : obligations \\
\quad\ \bullet\ \exists\ speaking : Speaking \mid \\
\qquad\quad speaking.speaker = handingOut.carrier \land \\
\qquad\quad speaking.addressee = handingOut.receiver \land \\
\qquad\quad speaking.time = spoken \land \\
\qquad\quad isProposeToDischargeHandingOutParcel(handingOut, \\
\qquad\qquad\qquad\qquad\qquad\qquad replied) \in speaking.points \\
\qquad \bullet\ speaking = o)
\end{array}
$$

To summarize, the above sections presented the communicational behaviour of buyers, seller and carriers (behaviour that they can follow as long as their committal preconditions are met). Nevertheless, these operations are disembodied of any concrete interaction. The next section will show how these operations are assembled into coherent interactions that follow the principles of the model for conversations.

5.8 eShopping as a Joint Activity

The class *eBooksShopping* (which is shown in Figure 39) is a subclass of *JointActivity* which specifies the interactions that can occur when shopping for books at *eBookStore*. This class specifies three participants (a *buyer*, a *seller* and a *carrier*) and four actions in which they participate (*selling*, *delivering*, *paying books* and *paying delivery*). This class also specifies that the books involved in the *selling* action are equal to those in the *delivering* action.

5.8.1 Interactions

The operation *Interaction* defines the sequences of interdependent agent operations making the activity. This operation (which is illustrated as a Petri Net in Figure 40) specifies that a request from a buyer to buy books be followed by an acceptance from the seller. Given this acceptance, the specification defines the concurrent execution of two interactions: one for continuing the sale transaction (between the buyer and the seller), and another one for arranging the delivery of books (between the seller and the carrier).

On the one hand, the interaction between the buyer and the seller continues with a communication in which the seller requests to the buyer to provide a payment for the books requested. As specified, this request could be followed by the acceptance to pay and the submission of payment—either as separate communications or in a single utterance. These acceptance to pay and payment communications are followed by the seller’s acceptance of the payment and the remission of a receipt (which, as earlier, could be done through one or two utterances).

eBookStoreShopping

$$
\begin{array}{l}
JointActivity \\
\hline
buyer : \downarrow Buyer \\
seller : \downarrow Seller \\
carrier : \downarrow Carrier \\
selling : Selling \\
delivering : Delivering \\
payingBooks, payingDelivery : Paying \\
\hline
buyer = selling.buyer = payingBooks.payer = delivering.receiver \land \\
seller = selling.seller = payingBooks.payee = delivering.sender \\
\qquad = payingDelivery.payer \land \\
carrier = delivering.carrier = payingDelivery.payee \land \\
selling = payingBooks.payable \land \\
delivering = payingDelivery.payable \land \\
actions = \{selling, delivering, payingBooks, payingDelivery\} \\
\forall\, bookRequested : BookDescription \mid \\
\quad bookRequested \in selling.items \\
\bullet\ \exists\, bookDelivered : BookDescription \mid \\
\qquad bookDelivered \in delivering.parcel \\
\quad \bullet\ bookDelivered.title = bookRequested.title
\end{array}
$$

$$
\begin{array}{l}
Interaction \mathrel{\widehat=} [\, buyer? : \downarrow Buyer;\ seller? : \downarrow Seller;\ carrier? : \downarrow Carrier \mid \\
\qquad\qquad\qquad buyer? = buyer \land seller? = seller \land carrier? = carrier \,] \bullet \\
\quad buyer.RequestingToBuyBooks \fatsemi \\
\quad seller.AcceptingToSellBooks \fatsemi \\
\quad ((seller.RequestingBooksPayment \fatsemi \\
\qquad ((buyer.AcceptingToPayBooks \fatsemi buyer.SubmittingBooksPayment) \\
\qquad\ \ [] \\
\qquad\ buyer.PayingBooks) \fatsemi \\
\qquad ((seller.AcceptingBooksPayment \fatsemi seller.ProposingToEndSale) \\
\qquad\ \ [] \\
\qquad\ seller.ProposingToEndTransaction) \fatsemi \\
\qquad buyer.AcceptingToEndSale) \land \\
\quad\ (seller.RequestingToDeliverBooks \fatsemi \\
\qquad carrier.RequestingDeliveryPayment \fatsemi \\
\qquad ((seller.AcceptingToPayDelivery \fatsemi seller.SubmittingDeliveryPayment) \\
\qquad\ \ [] \\
\qquad\ seller.PayingDelivery) \fatsemi \\
\qquad ((carrier.AcceptingDeliveryPayment \fatsemi carrier.AcceptingToDeliverBooks) \\
\qquad\ \ [] \\
\qquad\ carrier.AcceptingBooksDelivery))) \fatsemi \\
\quad carrier.RequestingToReceiveBooks \fatsemi \\
\quad buyer.AcceptingToReceiveBooks
\end{array}
$$

Figure 39. Definition of the joint activity *eBookStoreShopping*.

On the other hand, the interaction between the seller and the carrier begins with a request to deliver books from seller to carrier. This is followed by a communication from the carrier in which he requests the seller to pay for the delivery. The seller replies to this request with an acceptance to pay and the submission of a payment (which could be carried out in one or two utterances), which is followed by the carrier's acknowledgement of the payment and his acceptance to deliver the books.

Once these concurrent interactions have ended, the conversation continues with a communication from the carrier to the buyer in which he (the carrier) hands out to the buyer the books he received from the seller for delivery. The interaction ends with an acknowledgement from the buyer in which he accepts the books delivered.

5.9 eBookStore Society

The class *eBookStoreSociety* is specified as a subclass of *PFPsociety* that defines the policies for payable actions as norms in the society, and also that *eBookStoreShopping* is a joint activity in the society.

```
eBookStoreSociety
PFPsociety

  INIT
  (∃₁ p₁ : PayablePolicy1 • p₁ ∈ norms)
  (∃₁ p₂ : PayablePolicy2 • p₂ ∈ norms)

eBookStoreShopping ≙ [ shop : eBookStoreShopping |
                       shop ∈ activities ] •
                       shop.Interaction
```

5.10 Example Conversation: Buying Books

Figure 41 shows a UML interaction diagram for a conversation in the *eBooksShopping* activity. The conversation begins when a buyer requests to a seller to buy books, and advances until a carrier delivers the books to the buyer.

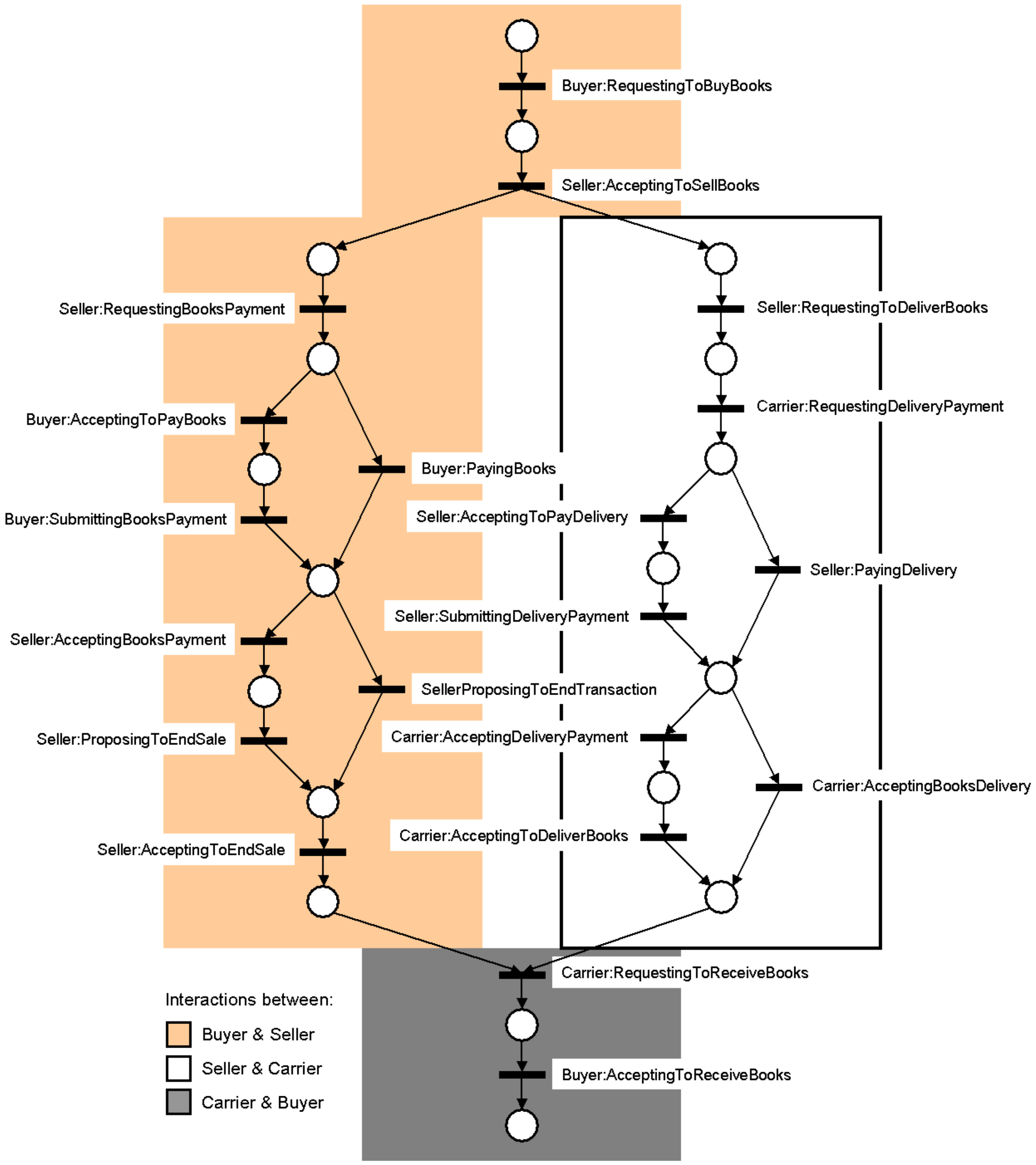

Buyer:RequestingToBuyBooks
Seller:AcceptingToSellBooks
Seller:RequestingBooksPayment
Seller:RequestingToDeliverBooks
Carrier:RequestingDeliveryPayment
Buyer:AcceptingToPayBooks
Buyer:PayingBooks
Buyer:SubmittingBooksPayment
Seller:AcceptingToPayDelivery
Seller:PayingDelivery
Seller:SubmittingDeliveryPayment
Seller:AcceptingBooksPayment
SellerProposingToEndTransaction
Seller:ProposingToEndSale
Carrier:AcceptingDeliveryPayment
Carrier:AcceptingBooksDelivery
Seller:AcceptingToEndSale
Carrier:AcceptingToDeliverBooks
Carrier:RequestingToReceiveBooks
Interactions between:
Buyer & Seller
Seller & Carrier
Carrier & Buyer
Buyer:AcceptingToReceiveBooks

Figure 40. Petri net diagram with the interactions in the joint activity *eBooksShopping*.

This conversation is specified in the *Interaction* schema by the sequence of the operations *RequestingToBuyBooks*, *AcceptingToSellBooks*, *RequestingBooksPayment*, *PayingBooks*, *ProposingToEndTransaction* and *AcceptingToEndSale* (which are the interactions between the buyer and the seller); *RequestingToDeliverBooks*, *RequestingDeliveryPayment*, *PayingDelivery* and *AcceptingBooksDelivery* (which are the interactions between the seller and the carrier); and, *RequestingToReceiveBooks* and *AcceptingToReceiveBooks* (which are the interactions between the carrier and the buyer).

The state of shared social commitments and obligations of the buyer in this example is shown in Figure 42 and Figure 43. The state changes with each interaction. Likewise, Figure 44, Figure 45 and Figure 46 show the state of the seller, and Figure 47 and Figure 48 show the state of the carrier.

5.10.1 Requesting to Buy Books

As shown in Figure 41, the interaction begins with an utterance from the buyer (identified as *b*) to the seller (identified as *s*) in which he requests that she sell him the books identified by the titles he is providing.

As specified in *b*'s operation *RequestingToBuyBooks*, this speech act contains a *Propose* illocutionary point (labelled α), proposing the adoption of a shared commitment in which *s* is responsible to *b* for an action *Selling* in which *s* performs and informs the results of the action to *b*. The uttering of this proposal triggers the following policies:

- Policy 1 (the uttering of a proposal commits the addressee to reply to the proposal): the uttering of proposal α results in the adoption of obligations in which *s* replies to *b*'s proposal α (added as obligations 1 and 2 on both the buyer (Figure 42) and the seller (Figure 44)).
- Policy 6 (proposing a payable action commits the payee to request a payment): the uttering of proposal α (which contains the payable action *Selling*) results in the adoption of obligations in which *s* (the payee) is to propose to *b* (the payer) to pay for the sale. These obligations are added as obligations 3 and 4 on both the buyer and the seller.

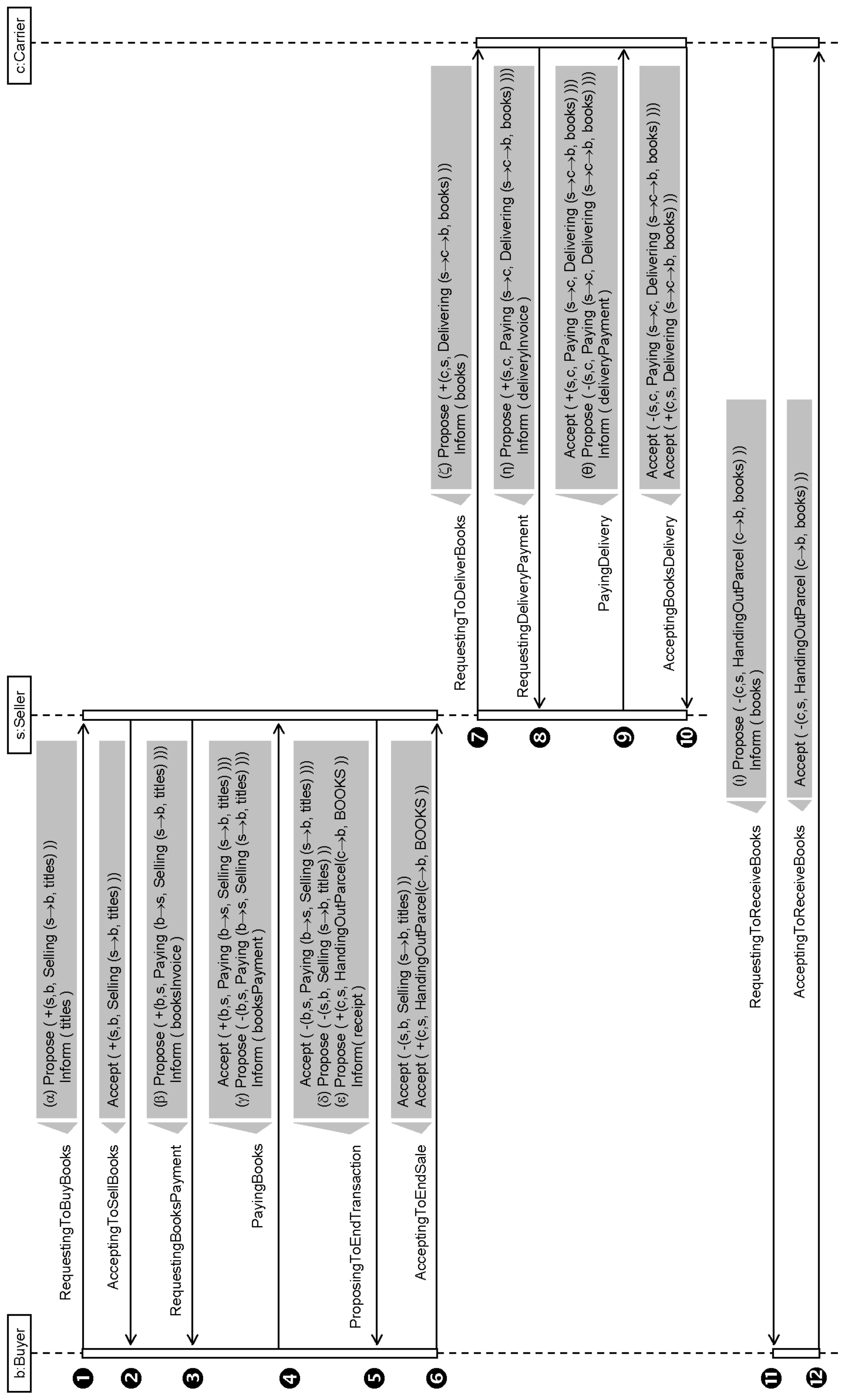

b:Buyer
s:Seller
c:Carrier
1 RequestingToBuyBooks
(α) Propose (+(s,b, Selling (s→b, titles)))
Inform (titles)
2 AcceptingToSellBooks
Accept (+(s,b, Selling (s→b, titles)))
3 RequestingBooksPayment
(β) Propose (+(b,s, Paying (b→s, Selling (s→b, titles))))
Inform (booksInvoice)
4 PayingBooks
Accept (+(b,s, Paying (b→s, Selling (s→b, titles))))
(γ) Propose (-(b,s, Paying (b→s, Selling (s→b, titles))))
Inform (booksPayment)
5 ProposingToEndTransaction
Accept (-(b,s, Paying (b→s, Selling (s→b, titles))))
(δ) Propose (-(s,b, Selling (s→b, titles)))
(ε) Propose (+(c,s, HandingOutParcel(c→b, BOOKS))
Inform(receipt)
6 AcceptingToEndSale
Accept (-(s,b, Selling (s→b, titles)))
Accept (+(c,s, HandingOutParcel(c→b, BOOKS))
7 RequestingToDeliverBooks
(ζ) Propose (+(c,s, Delivering (s→c→b, books)))
Inform (books)
8 RequestingDeliveryPayment
(η) Propose (+(s,c, Paying (s→c, Delivering (s→c→b, books))))
Inform (deliveryInvoice)
9 PayingDelivery
Accept (+(s,c, Paying (s→c, Delivering (s→c→b, books))))
(θ) Propose (-(s,c, Paying (s→c, Delivering (s→c→b, books))))
Inform (deliveryPayment)
10 AcceptingBooksDelivery
Accept (-(s,c, Paying (s→c, Delivering (s→c→b, books))))
Accept (+(c,s, Delivering (s→c→b, books)))
11 RequestingToReceiveBooks
(ι) Propose (-(c,s, HandingOutParcel (c→b, books)))
Inform (books)
12 AcceptingToReceiveBooks
Accept (-(c,s, HandingOutParcel (c→b, books)))

Figure 41. UML interaction diagram for an *eBooksShopping* conversation.

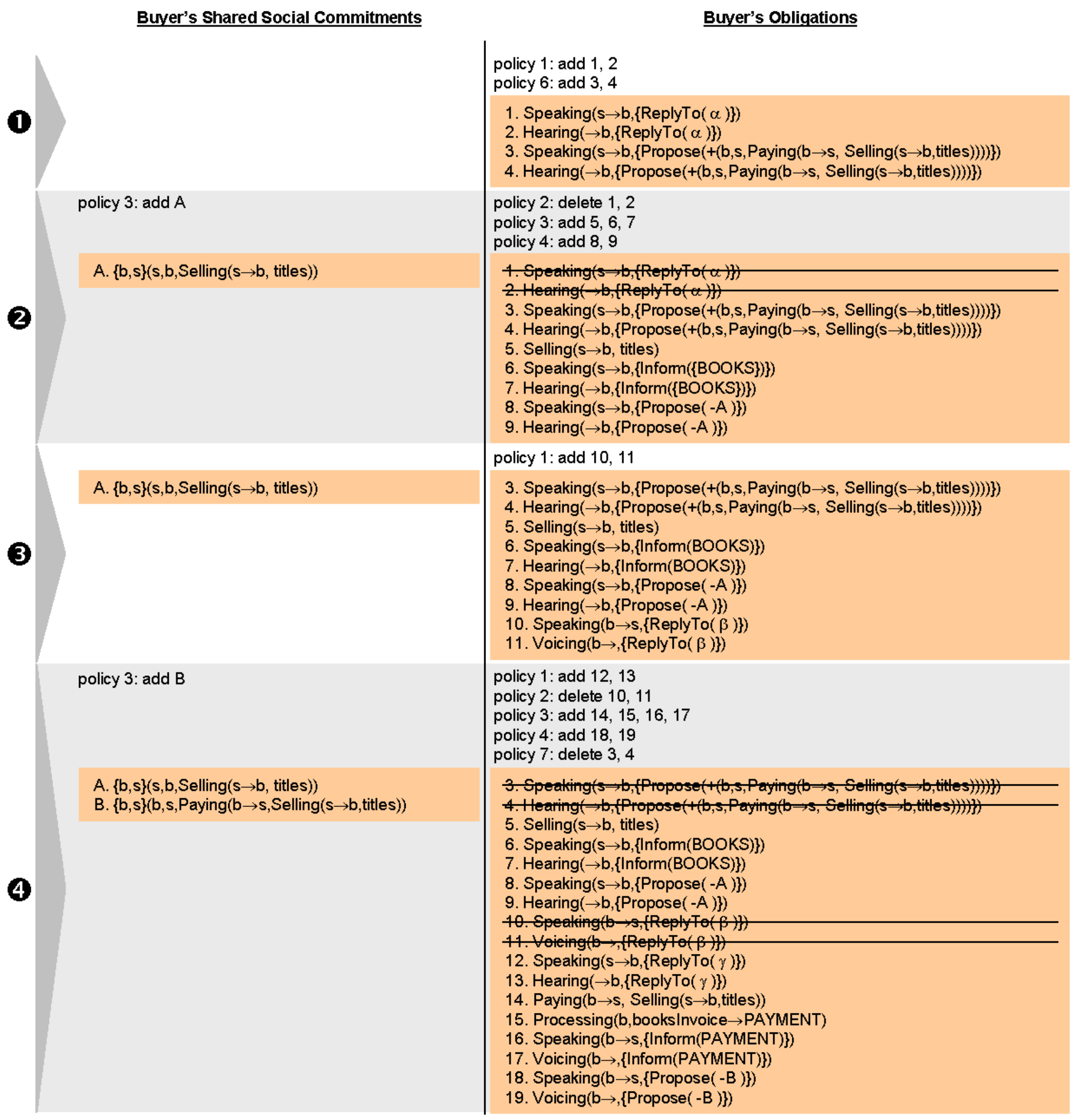
5.10.2 Accepting to Sell Books

The next interaction (labelled as interaction 2 in Figure 41) specifies the execution of *s*'s operation *AcceptingToSellBooks*, in which she accepts committing to sell the requested books. This acceptance is uttered if obligations exist in which *s* replies to a proposal to sell books (which exist as obligations 1 and 2). This acceptance results in the application of the following policies:

- Policy 2 (replying to a proposal discharges the obligation to reply): the acceptance to uptake the operation proposed in α discharges the obligation to reply to α (which deletes obligations 1 and 2 on both the buyer (Figure 42) and the seller (Figure 44)).
- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in α causes the adoption of the proposed commitment, in this case to sell books (added as commitment A in Figure 42 and Figure 44). In addition, this acceptance results in the adoption of obligations to perform the joint action. As such, the seller adopts obligations to produce and communicate the books (obligations 5 to 8), and the buyer adopts obligations to receive them (obligations 5 to 7).
- Policy 4 (accepting a *ProposingToDischarge* action obligates the *discharger* to propose its discharge): the acceptance to adopt the action *Selling* (which is a subtype of *ProposingToDischarge*) results in the adoption of obligations in which *s* (the *discharger*) is to propose to *b* (the *discharged*) discharging the action. These obligations are added as obligations 8 and 9 on the buyer, and 9 and 10 on the seller.

5.10.3 Requesting to Pay Books

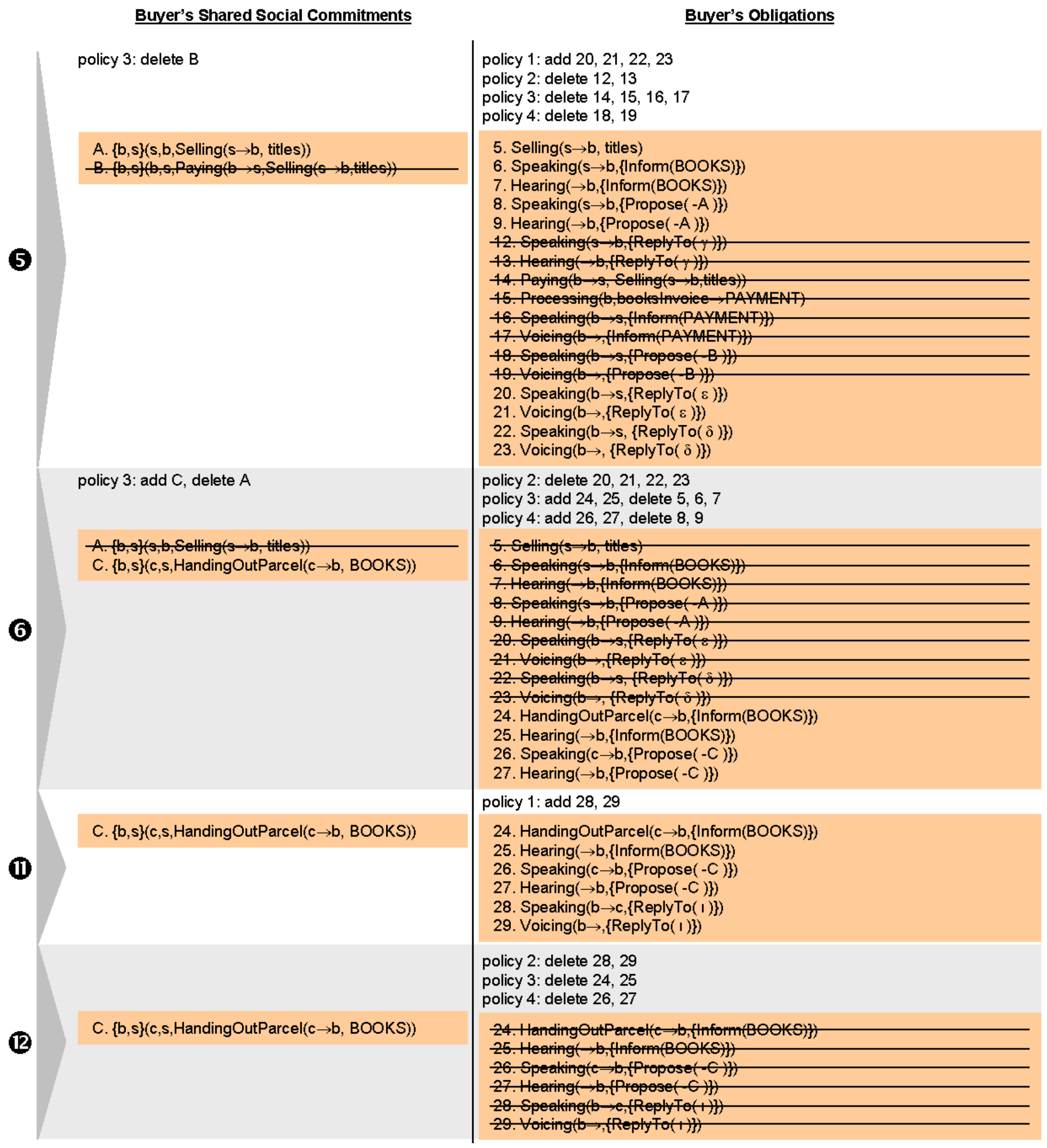
The next interaction (labelled as interaction 3 in Figure 41) specifies the execution of *s*'s operation *RequestingBooksPayment*, in which she proposes that *b* pays for the books (only if there is an obligation to request a payment—which exists as obligations 3 and 4). The uttering of this proposal (labelled β) triggers the following policy:



	Buyer's Shared Social Commitments	Buyer's Obligations
❶		policy 1: add 1, 2 policy 6: add 3, 4 1. Speaking(s→b,{ReplyTo(α)}) 2. Hearing(→b,{ReplyTo(α)}) 3. Speaking(s→b,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))}) 4. Hearing(→b,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))})
❷	policy 3: add A A. {b,s}(s,b,Selling(s→b, titles))	policy 2: delete 1, 2 policy 3: add 5, 6, 7 policy 4: add 8, 9 ~~1. Speaking(s→b,{ReplyTo(α)})~~ ~~2. Hearing(→b,{ReplyTo(α)})~~ 3. Speaking(s→b,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))}) 4. Hearing(→b,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))}) 5. Selling(s→b, titles) 6. Speaking(s→b,{Inform({BOOKS})}) 7. Hearing(→b,{Inform({BOOKS})}) 8. Speaking(s→b,{Propose(-A)}) 9. Hearing(→b,{Propose(-A)})
❸	A. {b,s}(s,b,Selling(s→b, titles))	policy 1: add 10, 11 3. Speaking(s→b,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))}) 4. Hearing(→b,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))}) 5. Selling(s→b, titles) 6. Speaking(s→b,{Inform(BOOKS)}) 7. Hearing(→b,{Inform(BOOKS)}) 8. Speaking(s→b,{Propose(-A)}) 9. Hearing(→b,{Propose(-A)}) 10. Speaking(b→s,{ReplyTo(β)}) 11. Voicing(b→,{ReplyTo(β)})
❹	policy 3: add B A. {b,s}(s,b,Selling(s→b, titles)) B. {b,s}(b,s,Paying(b→s,Selling(s→b,titles))	policy 1: add 12, 13 policy 2: delete 10, 11 policy 3: add 14, 15, 16, 17 policy 4: add 18, 19 policy 7: delete 3, 4 ~~3. Speaking(s→b,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))})~~ ~~4. Hearing(→b,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))})~~ 5. Selling(s→b, titles) 6. Speaking(s→b,{Inform(BOOKS)}) 7. Hearing(→b,{Inform(BOOKS)}) 8. Speaking(s→b,{Propose(-A)}) 9. Hearing(→b,{Propose(-A)}) ~~10. Speaking(b→s,{ReplyTo(β)})~~ ~~11. Voicing(b→,{ReplyTo(β)})~~ 12. Speaking(s→b,{ReplyTo(γ)}) 13. Hearing(→b,{ReplyTo(γ)}) 14. Paying(b→s, Selling(s→b,titles)) 15. Processing(b,booksInvoice→PAYMENT) 16. Speaking(b→s,{Inform(PAYMENT)}) 17. Voicing(b→,{Inform(PAYMENT)}) 18. Speaking(b→s,{Propose(-B)}) 19. Voicing(b→,{Propose(-B)})

Figure 42. State of shared social commitments and obligations of the buyer in the *eBooksShopping* conversation example (part 1 of 2).

- Policy 1 (the uttering of a proposal commits the addressee to reply to the proposal): the uttering of proposal β results in the adoption of obligations in which *b* replies to β (added as obligations 10 and 11 on the buyer, and 11 and 12 on the seller).



	Buyer's Shared Social Commitments	Buyer's Obligations
❺	policy 3: delete B A. {b,s}(s,b,Selling(s→b, titles)) ~~B. {b,s}(b,s,Paying(b→s,Selling(s→b,titles))~~	policy 1: add 20, 21, 22, 23 policy 2: delete 12, 13 policy 3: delete 14, 15, 16, 17 policy 4: delete 18, 19 5. Selling(s→b, titles) 6. Speaking(s→b,{Inform(BOOKS)}) 7. Hearing(→b,{Inform(BOOKS)}) 8. Speaking(s→b,{Propose(-A)}) 9. Hearing(→b,{Propose(-A)}) ~~12. Speaking(s→b,{ReplyTo(γ)})~~ ~~13. Hearing(→b,{ReplyTo(γ)})~~ ~~14. Paying(b→s, Selling(s→b,titles))~~ ~~15. Processing(b,booksInvoice→PAYMENT)~~ ~~16. Speaking(b→s,{Inform(PAYMENT)})~~ ~~17. Voicing(b→,{Inform(PAYMENT)})~~ ~~18. Speaking(b→s,{Propose(-B)})~~ ~~19. Voicing(b→,{Propose(-B)})~~ 20. Speaking(b→s,{ReplyTo(ε)}) 21. Voicing(b→,{ReplyTo(ε)}) 22. Speaking(b→s, {ReplyTo(δ)}) 23. Voicing(b→, {ReplyTo(δ)})
❻	policy 3: add C, delete A ~~A. {b,s}(s,b,Selling(s→b, titles))~~ C. {b,s}(c,s,HandingOutParcel(c→b, BOOKS))	policy 2: delete 20, 21, 22, 23 policy 3: add 24, 25, delete 5, 6, 7 policy 4: add 26, 27, delete 8, 9 ~~5. Selling(s→b, titles)~~ ~~6. Speaking(s→b,{Inform(BOOKS)})~~ ~~7. Hearing(→b,{Inform(BOOKS)})~~ ~~8. Speaking(s→b,{Propose(-A)})~~ ~~9. Hearing(→b,{Propose(-A)})~~ ~~20. Speaking(b→s,{ReplyTo(ε)})~~ ~~21. Voicing(b→,{ReplyTo(ε)})~~ ~~22. Speaking(b→s, {ReplyTo(δ)})~~ ~~23. Voicing(b→, {ReplyTo(δ)})~~ 24. HandingOutParcel(c→b,{Inform(BOOKS)}) 25. Hearing(→b,{Inform(BOOKS)}) 26. Speaking(c→b,{Propose(-C)}) 27. Hearing(→b,{Propose(-C)})
⓫	C. {b,s}(c,s,HandingOutParcel(c→b, BOOKS))	policy 1: add 28, 29 24. HandingOutParcel(c→b,{Inform(BOOKS)}) 25. Hearing(→b,{Inform(BOOKS)}) 26. Speaking(c→b,{Propose(-C)}) 27. Hearing(→b,{Propose(-C)}) 28. Speaking(b→c,{ReplyTo(ι)}) 29. Voicing(b→,{ReplyTo(ι)})
⓬	C. {b,s}(c,s,HandingOutParcel(c→b, BOOKS))	policy 2: delete 28, 29 policy 3: delete 24, 25 policy 4: delete 26, 27 ~~24. HandingOutParcel(c→b,{Inform(BOOKS)})~~ ~~25. Hearing(→b,{Inform(BOOKS)})~~ ~~26. Speaking(c→b,{Propose(-C)})~~ ~~27. Hearing(→b,{Propose(-C)})~~ ~~28. Speaking(b→c,{ReplyTo(ι)})~~ ~~29. Voicing(b→,{ReplyTo(ι)})~~

Figure 43. State of shared social commitments and obligations of the buyer in the *eBooksShopping* conversation example (part 2 of 2).

5.10.4 *Paying Books*

The next interaction (labelled as interaction 4 in Figure 41) specifies the execution of *b*'s operation *PayingBooks*, in which a) he accepts to pay the books, b) he proposes to discharge paying the books (a proposal that is labelled γ) and c) he informs a payment. The committal precondition for this utterance is an obligation in which *b* replies to a request for payment from *c* (which exists as obligations 10 and 11). The uttering of this speech act results in the following conversation policies:

- Policy 1 (the uttering of a proposal commits the addressee to reply to the proposal): the uttering of proposal γ results in the adoption of obligations in which *s* replies to γ (added as obligations 12 and 13 on the buyer, and 13 and 14 on the seller).
- Policy 2 (replying to a proposal discharges the obligations to reply): the acceptance to uptake the operation proposed in β discharges the obligations to reply to β (thus deleting obligations 10 and 11 on the buyer, and 11 and 12 on the seller).
- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in β causes the adoption of the commitment to pay (labelled as commitment B in Figure 42 and Figure 44), as well as its corresponding obligations (which are obligations 14 to 17 on the buyer, and 15 to 17 on the seller).
- Policy 4 (accepting to adopt a *ProposingToDischarge* action obligates the *discharger* of the action to propose its discharge): the acceptance to adopt the action to pay the books results in the adoption of obligations in which *b* (the *discharger*) is to propose *s* (the *discharged*) to discharge this action (which adds obligations 18 and 19 on both the buyer and the seller).
- Policy 7 (adopting a commitment to pay discharges the commitment to request a payment): the acceptance to commit to paying the books results in the discharge of obligations in which *s* (the payee) is to propose to *b* (the payer) to pay for the books (which deletes obligations 3 and 4 on both the buyer and the seller).

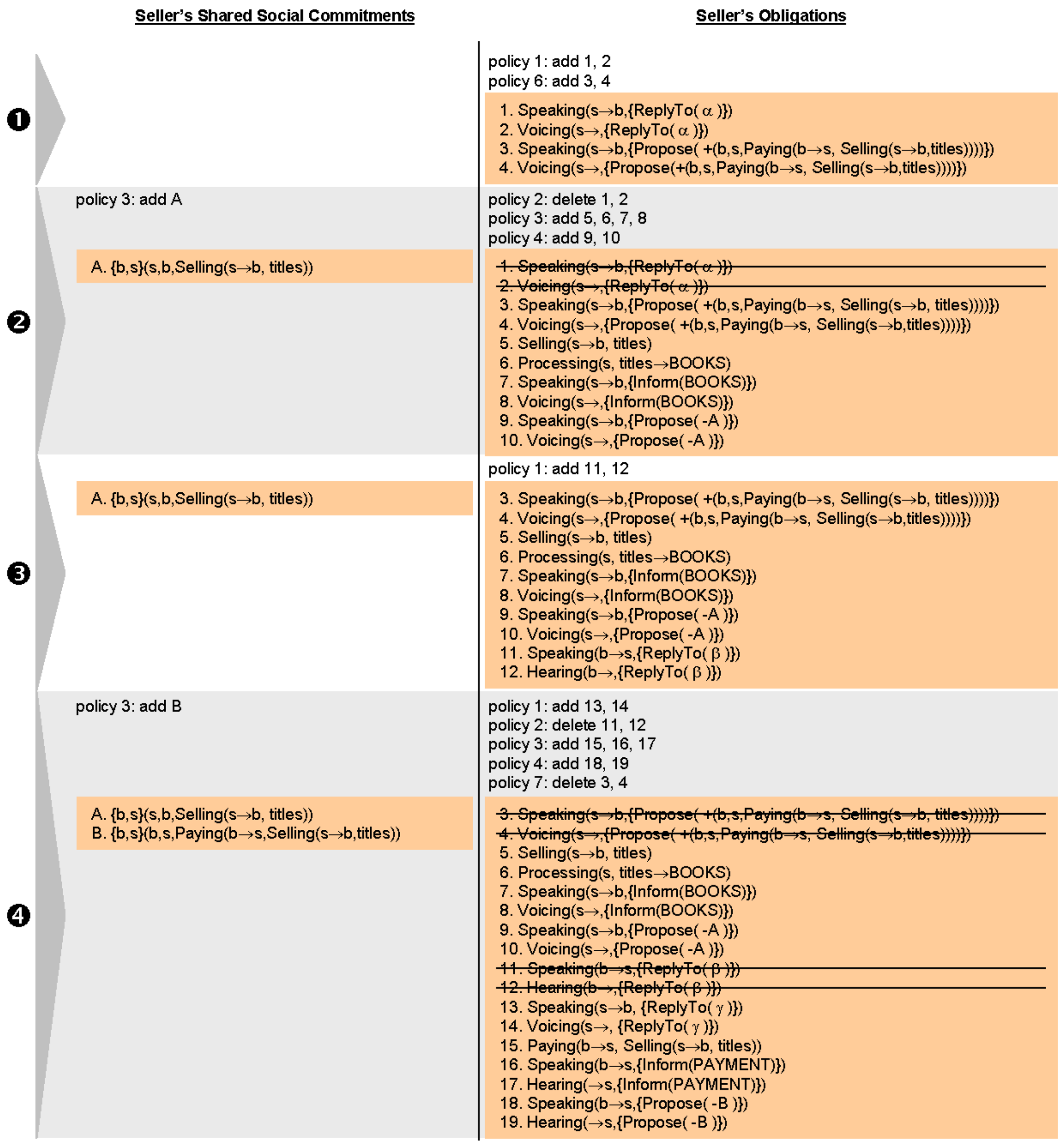
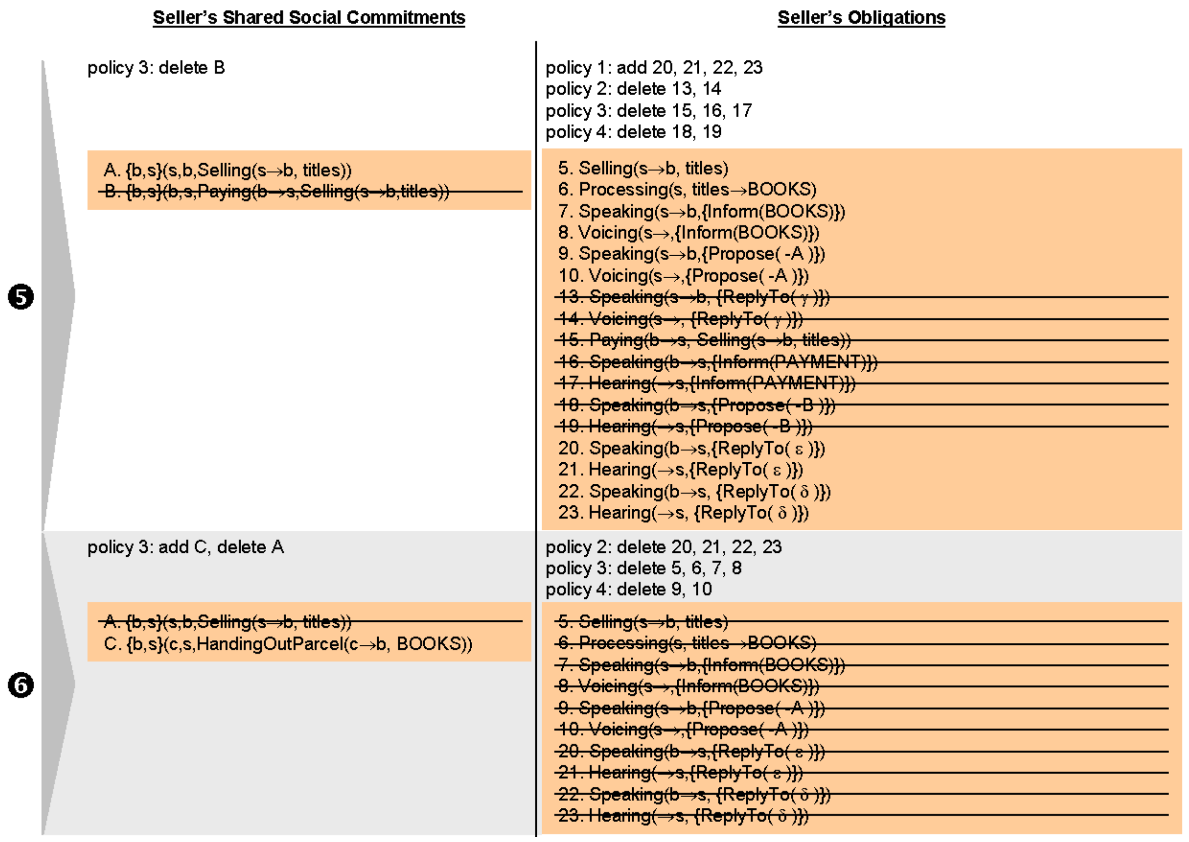

Seller's Shared Social Commitments
Seller's Obligations
❶
policy 1: add 1, 2
policy 6: add 3, 4
1. Speaking(s→b,{ReplyTo(α)})
2. Voicing(s→,{ReplyTo(α)})
3. Speaking(s→b,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))})
4. Voicing(s→,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))})
❷
policy 3: add A
A. {b,s}(s,b,Selling(s→b, titles))
policy 2: delete 1, 2
policy 3: add 5, 6, 7, 8
policy 4: add 9, 10
~~1. Speaking(s→b,{ReplyTo(α)})~~
~~2. Voicing(s→,{ReplyTo(α)})~~
3. Speaking(s→b,{Propose(+(b,s,Paying(b→s, Selling(s→b, titles))))})
4. Voicing(s→,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))})
5. Selling(s→b, titles)
6. Processing(s, titles→BOOKS)
7. Speaking(s→b,{Inform(BOOKS)})
8. Voicing(s→,{Inform(BOOKS)})
9. Speaking(s→b,{Propose(-A)})
10. Voicing(s→,{Propose(-A)})
❸
A. {b,s}(s,b,Selling(s→b, titles))
policy 1: add 11, 12
3. Speaking(s→b,{Propose(+(b,s,Paying(b→s, Selling(s→b, titles))))})
4. Voicing(s→,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))})
5. Selling(s→b, titles)
6. Processing(s, titles→BOOKS)
7. Speaking(s→b,{Inform(BOOKS)})
8. Voicing(s→,{Inform(BOOKS)})
9. Speaking(s→b,{Propose(-A)})
10. Voicing(s→,{Propose(-A)})
11. Speaking(b→s,{ReplyTo(β)})
12. Hearing(b→,{ReplyTo(β)})
❹
policy 3: add B
A. {b,s}(s,b,Selling(s→b, titles))
B. {b,s}(b,s,Paying(b→s,Selling(s→b,titles))
policy 1: add 13, 14
policy 2: delete 11, 12
policy 3: add 15, 16, 17
policy 4: add 18, 19
policy 7: delete 3, 4
~~3. Speaking(s→b,{Propose(+(b,s,Paying(b→s, Selling(s→b, titles))))})~~
~~4. Voicing(s→,{Propose(+(b,s,Paying(b→s, Selling(s→b,titles))))})~~
5. Selling(s→b, titles)
6. Processing(s, titles→BOOKS)
7. Speaking(s→b,{Inform(BOOKS)})
8. Voicing(s→,{Inform(BOOKS)})
9. Speaking(s→b,{Propose(-A)})
10. Voicing(s→,{Propose(-A)})
~~11. Speaking(b→s,{ReplyTo(β)})~~
~~12. Hearing(b→,{ReplyTo(β)})~~
13. Speaking(s→b, {ReplyTo(γ)})
14. Voicing(s→, {ReplyTo(γ)})
15. Paying(b→s, Selling(s→b, titles))
16. Speaking(b→s,{Inform(PAYMENT)})
17. Hearing(→s,{Inform(PAYMENT)})
18. Speaking(b→s,{Propose(-B)})
19. Hearing(→s,{Propose(-B)})

Figure 44. State of shared social commitments and obligations of the seller in the *eBooksShopping* conversation example (part 1 of 3).

5.10.5 *Proposing to End Sale*

The next interaction (labelled as interaction 5 in Figure 41) specifies the execution of *s*'s operation *ProposingToEndTransaction*, in which a) she accepts to discharge the

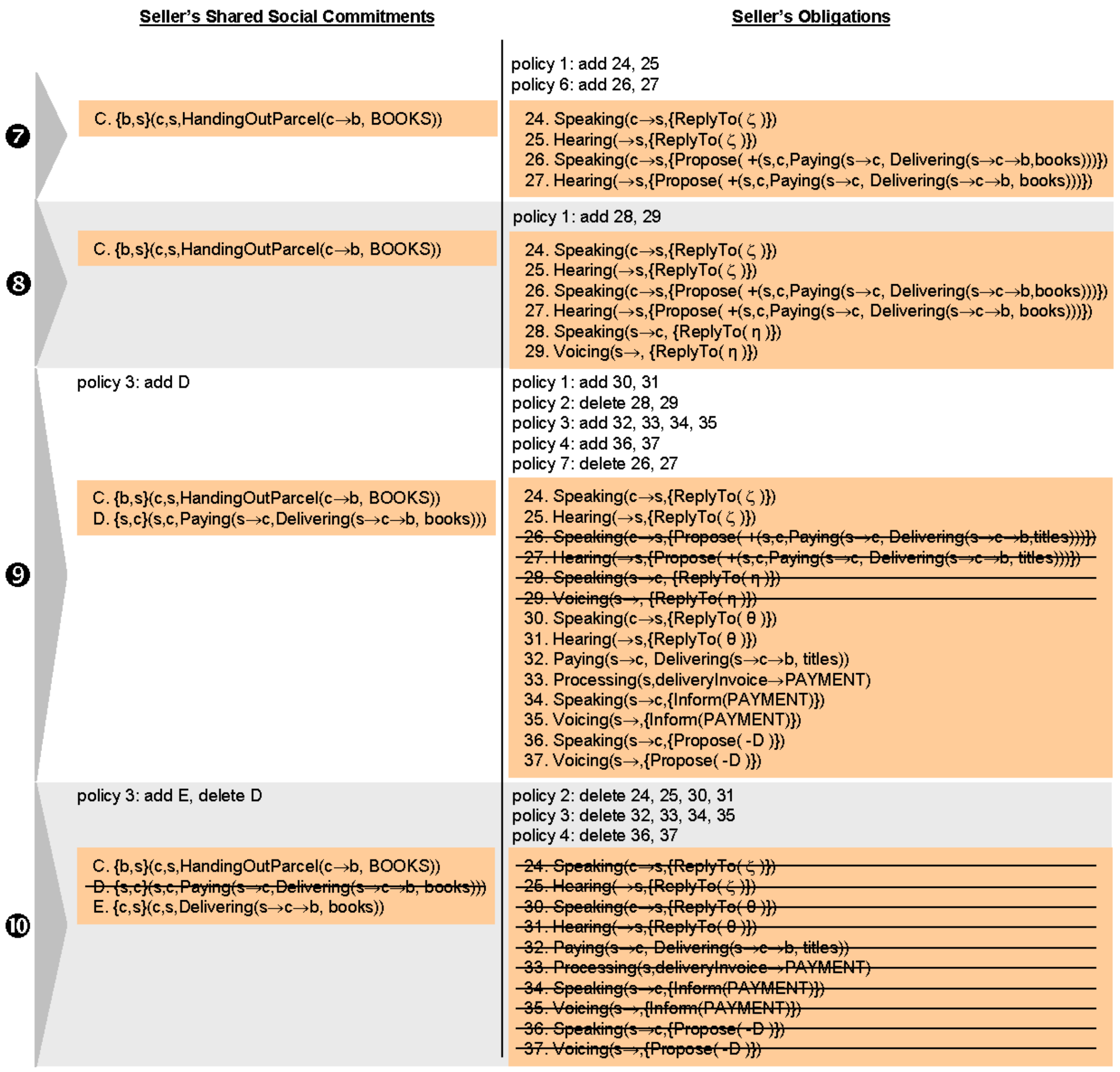


	Seller's Shared Social Commitments	Seller's Obligations
❺	policy 3: delete B A. {b,s}(s,b,Selling(s→b, titles)) ~~B. {b,s}(b,s,Paying(b→s,Selling(s→b,titles))~~	policy 1: add 20, 21, 22, 23 policy 2: delete 13, 14 policy 3: delete 15, 16, 17 policy 4: delete 18, 19 5. Selling(s→b, titles) 6. Processing(s, titles→BOOKS) 7. Speaking(s→b,{Inform(BOOKS)}) 8. Voicing(s→,{Inform(BOOKS)}) 9. Speaking(s→b,{Propose(-A)}) 10. Voicing(s→,{Propose(-A)}) ~~13. Speaking(s→b, {ReplyTo(γ)})~~ ~~14. Voicing(s→, {ReplyTo(γ)})~~ ~~15. Paying(b→s, Selling(s→b, titles))~~ ~~16. Speaking(b→s,{Inform(PAYMENT)})~~ ~~17. Hearing(→s,{Inform(PAYMENT)})~~ ~~18. Speaking(b→s,{Propose(-B)})~~ ~~19. Hearing(→s,{Propose(-B)})~~ 20. Speaking(b→s,{ReplyTo(ε)}) 21. Hearing(→s,{ReplyTo(ε)}) 22. Speaking(b→s, {ReplyTo(δ)}) 23. Hearing(→s, {ReplyTo(δ)})
❻	policy 3: add C, delete A ~~A. {b,s}(s,b,Selling(s→b, titles))~~ C. {b,s}(c,s,HandingOutParcel(c→b, BOOKS))	policy 2: delete 20, 21, 22, 23 policy 3: delete 5, 6, 7, 8 policy 4: delete 9, 10 ~~5. Selling(s→b, titles)~~ ~~6. Processing(s, titles→BOOKS)~~ ~~7. Speaking(s→b,{Inform(BOOKS)})~~ ~~8. Voicing(s→,{Inform(BOOKS)})~~ ~~9. Speaking(s→b,{Propose(-A)})~~ ~~10. Voicing(s→,{Propose(-A)})~~ ~~20. Speaking(b→s,{ReplyTo(ε)})~~ ~~21. Hearing(→s,{ReplyTo(ε)})~~ ~~22. Speaking(b→s, {ReplyTo(δ)})~~ ~~23. Hearing(→s, {ReplyTo(δ)})~~

Figure 45. State of shared social commitments and obligations of the seller in the *eBooksShopping* conversation example (part 2 of 3).

commitment to be paid (which she does if there is an obligation to reply to a proposal to discharge paying—which exists as obligations 13 and 14); b) she proposes to discharge selling the books (labelled as δ) (only if there is an obligation to propose its discharge—which exists as obligations 18 and 19); c) she informs a receipt; and d) she proposes to adopt a commitment in which the carrier hands out the books to the buyer (which is labelled as ε). The uttering of this speech act triggers the following conversational policies:

- Policy 1 (the uttering of a proposal commits the addressee to reply to the proposal): the uttering of proposal ε results in the adoption of obligations in which *b* replies to ε (added as obligations 20 and 21 on both the buyer (Figure 43) and the seller (Figure 45)).



	Seller's Shared Social Commitments	Seller's Obligations
❼	C. {b,s}(c,s,HandingOutParcel(c→b, BOOKS))	policy 1: add 24, 25 policy 6: add 26, 27 24. Speaking(c→s,{ReplyTo(ζ)}) 25. Hearing(→s,{ReplyTo(ζ)}) 26. Speaking(c→s,{Propose(+(s,c,Paying(s→c, Delivering(s→c→b,books)))}) 27. Hearing(→s,{Propose(+(s,c,Paying(s→c, Delivering(s→c→b, books)))})
❽	C. {b,s}(c,s,HandingOutParcel(c→b, BOOKS))	policy 1: add 28, 29 24. Speaking(c→s,{ReplyTo(ζ)}) 25. Hearing(→s,{ReplyTo(ζ)}) 26. Speaking(c→s,{Propose(+(s,c,Paying(s→c, Delivering(s→c→b,books)))}) 27. Hearing(→s,{Propose(+(s,c,Paying(s→c, Delivering(s→c→b, books)))}) 28. Speaking(s→c, {ReplyTo(η)}) 29. Voicing(s→, {ReplyTo(η)})
❾	policy 3: add D C. {b,s}(c,s,HandingOutParcel(c→b, BOOKS)) D. {s,c}(s,c,Paying(s→c,Delivering(s→c→b, books)))	policy 1: add 30, 31 policy 2: delete 28, 29 policy 3: add 32, 33, 34, 35 policy 4: add 36, 37 policy 7: delete 26, 27 24. Speaking(c→s,{ReplyTo(ζ)}) 25. Hearing(→s,{ReplyTo(ζ)}) ~~26. Speaking(c→s,{Propose(+(s,c,Paying(s→c, Delivering(s→c→b,titles)))})~~ ~~27. Hearing(→s,{Propose(+(s,c,Paying(s→c, Delivering(s→c→b, titles)))})~~ ~~28. Speaking(s→c, {ReplyTo(η)})~~ ~~29. Voicing(s→, {ReplyTo(η)})~~ 30. Speaking(c→s,{ReplyTo(θ)}) 31. Hearing(→s,{ReplyTo(θ)}) 32. Paying(s→c, Delivering(s→c→b, titles)) 33. Processing(s,deliveryInvoice→PAYMENT) 34. Speaking(s→c,{Inform(PAYMENT)}) 35. Voicing(s→,{Inform(PAYMENT)}) 36. Speaking(s→c,{Propose(-D)}) 37. Voicing(s→,{Propose(-D)})
❿	policy 3: add E, delete D C. {b,s}(c,s,HandingOutParcel(c→b, BOOKS)) ~~D. {s,c}(s,c,Paying(s→c,Delivering(s→c→b, books)))~~ E. {c,s}(c,s,Delivering(s→c→b, books))	policy 2: delete 24, 25, 30, 31 policy 3: delete 32, 33, 34, 35 policy 4: delete 36, 37 ~~24. Speaking(c→s,{ReplyTo(ζ)})~~ ~~25. Hearing(→s,{ReplyTo(ζ)})~~ ~~30. Speaking(c→s,{ReplyTo(θ)})~~ ~~31. Hearing(→s,{ReplyTo(θ)})~~ ~~32. Paying(s→c, Delivering(s→c→b, titles))~~ ~~33. Processing(s,deliveryInvoice→PAYMENT)~~ ~~34. Speaking(s→c,{Inform(PAYMENT)})~~ ~~35. Voicing(s→,{Inform(PAYMENT)})~~ ~~36. Speaking(s→c,{Propose(-D)})~~ ~~37. Voicing(s→,{Propose(-D)})~~

Figure 46. State of shared social commitments and obligations of the seller in the *eBooksShopping* conversation example (part 3 of 3).

- Policy 1 (*ditto*): the uttering of proposal δ results in the adoption of obligations in which *b* replies to δ (added as obligations 22 and 23 on both the buyer and the seller).

- Policy 2 (replying to a proposal discharges the obligations to reply): the acceptance to uptake the operation proposed in γ discharges the obligations to reply to γ (thus deleting obligations 12 and 13 on the buyer, and 13 and 14 on the seller).
- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in γ causes the discharge of the commitment to pay (labelled as commitment B in Figure 43 and Figure 45) and all corresponding obligations (which are obligations 14 to 17 on the buyer, and 15 to 17 on the seller), and lastly
- Policy 4 (accepting to discharge a *ProposingToDischarge* action discards the obligations in which the *discharger* of the action is to propose its discharge): the acceptance to discharge the commitment to pay results in the discharge of the obligations in which *b* is to propose discharging the commitment that he pays the books (which deletes obligations 18 and 19 on both the buyer and the seller).

5.10.6 Accepting to End Sale

The last interaction between the buyer and the seller (labelled as interaction 6 in Figure 41) specifies the execution of *b*'s operation *AcceptingToEndSale*, in which *b* accepts both the discharge of the commitment that *s* sells the books to *b*, and the adoption of the commitment that *c* (the carrier) will deliver the books to *b*. The committal preconditions for this utterance are that there exist obligations to reply to a proposal to discharge selling (which exist as obligations 22 and 23), and obligations to reply to a proposal to adopt the delivery (which exist as obligations 20 and 21). The uttering of these acceptances results in the application of the following policies:

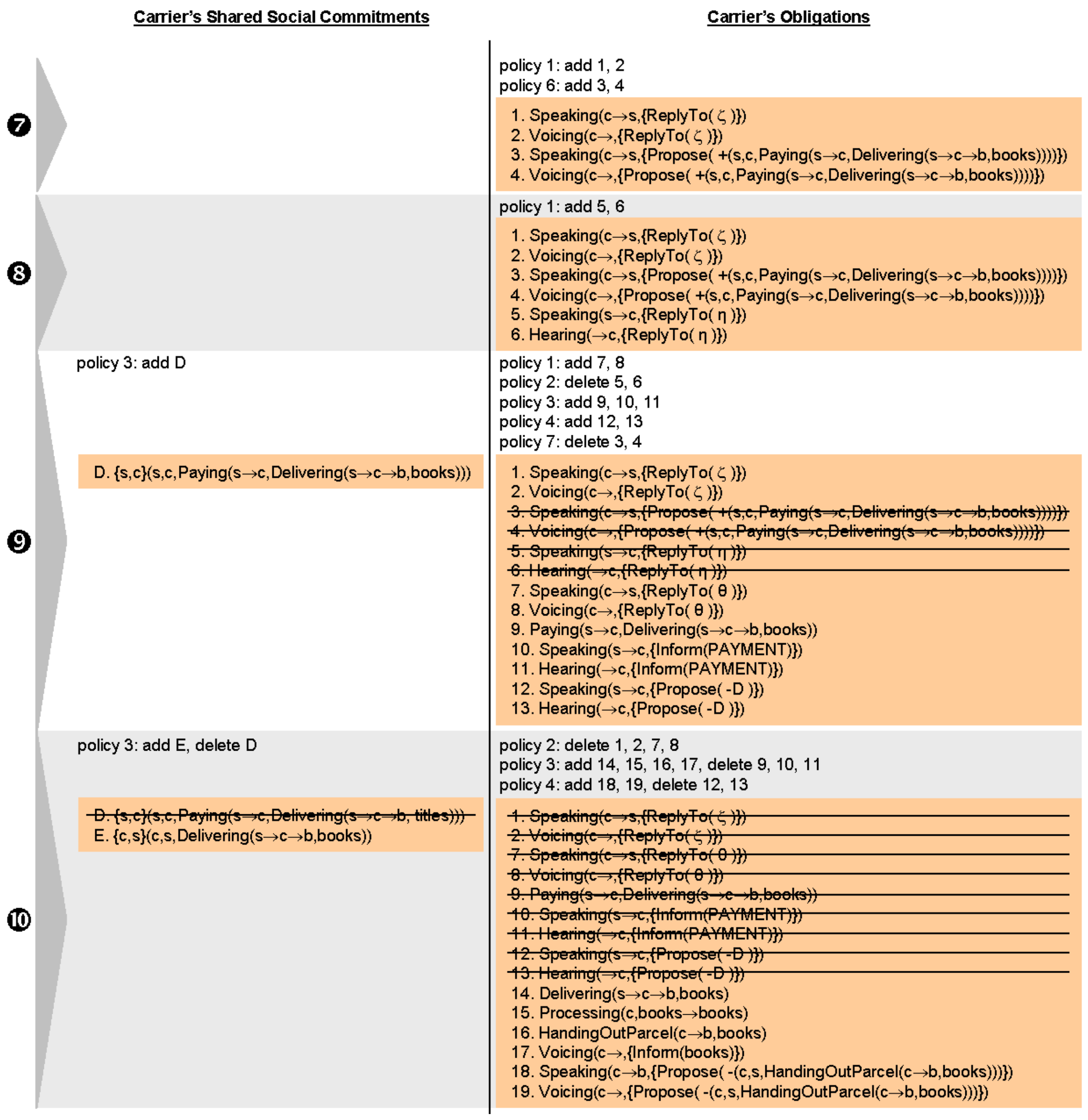
- Policy 2 (replying to a proposal discharges the obligations to reply): the acceptance to uptake the operation proposed in ε discharges the obligations to reply to ε (thus deleting obligations 20 and 21 on both the buyer and the seller).

- Policy 2 (*ditto*): the acceptance to uptake the operation proposed in δ discharges the obligations to reply to δ (thus deleting obligations 22 and 23 on both the buyer and the seller).
- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in ε causes the adoption of the commitment to hand out the books (labelled as commitment C in Figure 43 and Figure 45) as well as its corresponding obligations (which are obligations 24 and 25 on the buyer, and none on the seller).
- Policy 3 (*ditto*): the acceptance to uptake the operation proposed in δ causes the discharge of the commitment to pay the books (labelled as commitment A) and all corresponding obligations (which are obligations 5 to 7 on the buyer, and 5 to 8 on the seller).

5.10.7 Requesting to Deliver Books

The request for delivery is the first interaction between the seller and the carrier (who is identified as *c*). This interaction (labelled as interaction 7 in Figure 41) specifies the execution of *s*'s operation *RequestingToDeliverBooks*, in which *s* proposes to *c* to commit to deliver a given set of books. The uttering of this proposal (labelled ζ) triggers the following conversation policies:

- Policy 1 (the uttering of a proposal obligates the addressee to reply to the proposal): the uttering of proposal ζ results in the obligations in which *c* replies to ζ (which adds obligations 24 and 25 on the seller (Figure 46), and 1 and 2 on the carrier (Figure 47)).
- Policy 6 (proposing a payable action commits the payee to request a payment): the uttering of proposal ζ (which contains the payable action *Delivering*) results in the adoption of obligations in which *c* (the payee) is to propose to *s* (the payer) to pay for the delivery. These obligations are added as obligations 26 and 27 on the seller and 3 and 4 on the carrier.



Carrier's Shared Social Commitments | Carrier's Obligations

❼

policy 1: add 1, 2
policy 6: add 3, 4

1. Speaking(c→s,{ReplyTo(ζ)})
2. Voicing(c→,{ReplyTo(ζ)})
3. Speaking(c→s,{Propose(+(s,c,Paying(s→c,Delivering(s→c→b,books))))})
4. Voicing(c→,{Propose(+(s,c,Paying(s→c,Delivering(s→c→b,books))))})

❽

policy 1: add 5, 6

1. Speaking(c→s,{ReplyTo(ζ)})
2. Voicing(c→,{ReplyTo(ζ)})
3. Speaking(c→s,{Propose(+(s,c,Paying(s→c,Delivering(s→c→b,books))))})
4. Voicing(c→,{Propose(+(s,c,Paying(s→c,Delivering(s→c→b,books))))})
5. Speaking(s→c,{ReplyTo(η)})
6. Hearing(→c,{ReplyTo(η)})

❾

policy 3: add D

D. {s,c}(s,c,Paying(s→c,Delivering(s→c→b,books)))

policy 1: add 7, 8
policy 2: delete 5, 6
policy 3: add 9, 10, 11
policy 4: add 12, 13
policy 7: delete 3, 4

1. Speaking(c→s,{ReplyTo(ζ)})
2. Voicing(c→,{ReplyTo(ζ)})
~~3. Speaking(c→s,{Propose(+(s,c,Paying(s→c,Delivering(s→c→b,books))))})~~
~~4. Voicing(c→,{Propose(+(s,c,Paying(s→c,Delivering(s→c→b,books))))})~~
~~5. Speaking(s→c,{ReplyTo(η)})~~
~~6. Hearing(→c,{ReplyTo(η)})~~
7. Speaking(c→s,{ReplyTo(θ)})
8. Voicing(c→,{ReplyTo(θ)})
9. Paying(s→c,Delivering(s→c→b,books))
10. Speaking(s→c,{Inform(PAYMENT)})
11. Hearing(→c,{Inform(PAYMENT)})
12. Speaking(s→c,{Propose(-D)})
13. Hearing(→c,{Propose(-D)})

❿

policy 3: add E, delete D

~~D. {s,c}(s,c,Paying(s→c,Delivering(s→c→b, titles)))~~
E. {c,s}(c,s,Delivering(s→c→b,books))

policy 2: delete 1, 2, 7, 8
policy 3: add 14, 15, 16, 17, delete 9, 10, 11
policy 4: add 18, 19, delete 12, 13

~~1. Speaking(c→s,{ReplyTo(ζ)})~~
~~2. Voicing(c→,{ReplyTo(ζ)})~~
~~7. Speaking(c→s,{ReplyTo(θ)})~~
~~8. Voicing(c→,{ReplyTo(θ)})~~
~~9. Paying(s→c,Delivering(s→c→b,books))~~
~~10. Speaking(s→c,{Inform(PAYMENT)})~~
~~11. Hearing(→c,{Inform(PAYMENT)})~~
~~12. Speaking(s→c,{Propose(-D)})~~
~~13. Hearing(→c,{Propose(-D)})~~
14. Delivering(s→c→b,books)
15. Processing(c,books→books)
16. HandingOutParcel(c→b,books)
17. Voicing(c→,{Inform(books)})
18. Speaking(c→b,{Propose(-(c,s,HandingOutParcel(c→b,books)))})
19. Voicing(c→,{Propose(-(c,s,HandingOutParcel(c→b,books)))})

Figure 47. State of shared social commitments and obligations of the carrier in the *eBooksShopping* conversation example (part 1 of 2).

5.10.8 Requesting to Pay Delivery

The next interaction (labelled as interaction 8 in Figure 41) specifies the execution of *c*'s operation *RequestingDeliveryPayment*, in which *c* proposes to *s* the adoption of the commitment that she pay for the delivery. This proposal is uttered if there are obligations

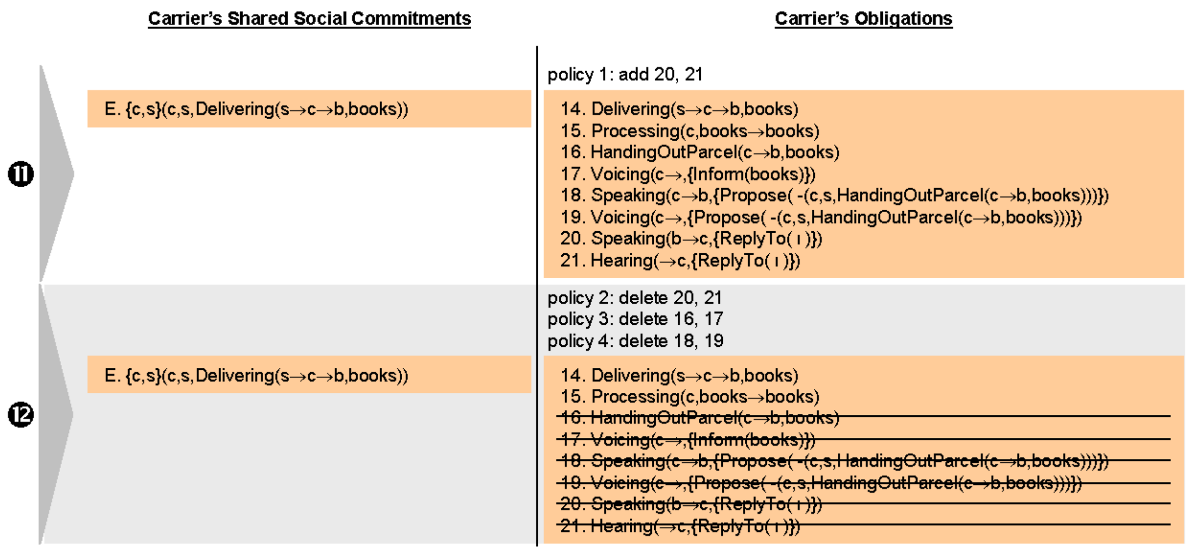

Carrier's Shared Social Commitments
Carrier's Obligations
⓫
policy 1: add 20, 21
E. {c,s}(c,s,Delivering(s→c→b,books))
14. Delivering(s→c→b,books)
15. Processing(c,books→books)
16. HandingOutParcel(c→b,books)
17. Voicing(c→,{Inform(books)})
18. Speaking(c→b,{Propose(-(c,s,HandingOutParcel(c→b,books)))})
19. Voicing(c→,{Propose(-(c,s,HandingOutParcel(c→b,books)))})
20. Speaking(b→c,{ReplyTo(ι)})
21. Hearing(→c,{ReplyTo(ι)})
⓬
policy 2: delete 20, 21
policy 3: delete 16, 17
policy 4: delete 18, 19
E. {c,s}(c,s,Delivering(s→c→b,books))
14. Delivering(s→c→b,books)
15. Processing(c,books→books)
~~16. HandingOutParcel(c→b,books)~~
~~17. Voicing(c→,{Inform(books)})~~
~~18. Speaking(c→b,{Propose(-(c,s,HandingOutParcel(c→b,books)))})~~
~~19. Voicing(c→,{Propose(-(c,s,HandingOutParcel(c→b,books)))})~~
~~20. Speaking(b→c,{ReplyTo(ι)})~~
~~21. Hearing(→c,{ReplyTo(ι)})~~

Figure 48. State of shared social commitments and obligations of the carrier in the *eBooksShopping* conversation example (part 2 of 2).

in which *c* requests a payment (which exist as obligations 3 and 4). The uttering of this proposal (labelled η) triggers the following conversational policy:

- Policy 1 (the uttering of a proposal commits the addressee to reply to the proposal): the uttering of proposal η results in the adoption of obligations in which *s* replies to η (added as obligations 28 and 29 on the seller, and 5 and 6 on the carrier).

5.10.9 *Paying Delivery*

The next interaction (labelled as interaction 9 in Figure 41) specifies the execution of *s*'s operation *PayingDelivery*, in which a) she accepts to pay the delivery, b) she proposes to discharge paying the delivery (a proposal that is labelled θ) and c) she informs a payment. The committal precondition for this utterance is an obligation in which *s* replies to a request for payment from *c* (which exists as obligations 26 and 27). The uttering of this speech act results in the following conversation policies:

- Policy 1 (the uttering of a proposal commits the addressee to reply to the proposal): the uttering of proposal θ results in the adoption of obligations in which *c* replies to θ (added as obligations 30 and 31 on the seller, and 7 and 8 on the carrier).

- Policy 2 (replying to a proposal discharges the obligations to reply): the acceptance to uptake the operation proposed in η discharges the obligations to reply to η (thus deleting obligations 28 and 29 on the seller, and 5 and 6 on the carrier).
- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in η causes the adoption of the commitment to pay (labelled as commitment D in Figure 46 and Figure 47) as well as its corresponding obligations (which are obligations 32 to 35 on the seller, and 9 to 11 on the carrier).
- Policy 4 (accepting to adopt a *ProposingToDischarge* action obligates the *discharger* of the action to propose its discharge): the acceptance to adopt the action to pay the delivery results in the adoption of obligations in which *s* (the *discharger*) is to propose to *c* (the *discharged*) to discharge this action (which adds obligations 36 and 37 on the seller, and 12 and 13 on the carrier), and lastly
- Policy 7 (adopting a commitment to pay discharges the commitment to request a payment): the acceptance to commit to paying the delivery of the books results in the discharge of obligations in which *c* (the payee) is to propose to *s* (the payer) to pay for this delivery (which deletes obligations 26 and 27 on the seller, and 3 and 4 on the carrier).

5.10.10 Accepting Books Delivery

The last interaction between the seller and the carrier (labelled as interaction 10 in Figure 41) specifies the execution of *c*'s operation *AcceptingBooksDelivery*, in which he accepts both the discharge of the commitment that *s* pays for the delivery, and the adoption of the commitment that he delivers the books. The committal preconditions for this utterance are that there exist obligations to reply to a proposal to discharge paying the delivery (which exist as obligations 7 and 8) and to reply to a proposal to adopt the delivery (which exist as obligations 1 and 2). The uttering of these acceptances results in the application of the following policies:

- Policy 2 (replying to a proposal discharges the obligations to reply): the acceptance to uptake the operation proposed in θ discharges the obligations to reply to θ (thus deleting obligations 30 and 31 on the seller, and 7 and 8 on the carrier).
- Policy 2 (*ditto*): the acceptance to uptake the operation proposed in ζ discharges the obligations to reply to ζ (thus deleting obligations 24 and 25 on the seller, and 1 and 2 on the carrier).
- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in ζ causes the adoption of the commitment to deliver the books (labelled as commitment E) as well as its corresponding obligations (which are obligations 14 to 17 on the carrier, and none on the seller).
- Policy 3 (*ditto*): the acceptance to uptake the operation proposed in θ causes the discharge of the commitment to pay (labelled as commitment D) and all corresponding obligations (which are obligations 32 to 35 on the seller, and 9 to 11 on the carrier).
- Policy 4 (accepting to discharge a *ProposingToDischarge* action discards the obligations in which the *discharger* of the action is to propose its discharge): the acceptance to discharge the commitment to pay the delivery results in the discharge of the obligations in which *s* is to propose discharging the commitment that she pays the delivery (which deletes obligations 36 and 37 on the seller, and 12 and 13 on the carrier), and lastly
- Policy 4 (accepting to adopt a *ProposingToDischarge* action obligates the *discharger* of the action to propose its discharge): the acceptance to adopt the action to hand out the delivered books results in the adoption of obligations in which *c* (the *discharger*) is to propose *b* (the *discharged*) to discharge this action (which adds obligations 18 and 19 on the carrier, and none on the seller).

5.10.11 Requesting to Receive Books

The carrier interacts with the buyer to deliver the books sold by the seller. This interaction (labelled as interaction 11 in Figure 41) specifies the execution of *c*'s operation *RequestingToReceiveBooks*, in which *c* proposes to *b* to discharge the commitment that he hand out the books to *b*. The uttering of this proposal (labelled ι) triggers the following conversation policy:

- Policy 1 (the uttering of a proposal obligates the addressee to reply to the proposal): the uttering of proposal ι results in the obligations in which *b* replies to ι (which adds obligations 28 and 29 on the buyer (Figure 43), and 20 and 21 on the carrier (Figure 48)).

5.10.12 Accepting Books

The last interaction in this conversation (labelled as interaction 12 in Figure 41) indicates the execution of *b*'s operation *AcceptingToReceiveBooks*, which specifies that *b* accepts to discharge the delivery of the books. This acceptance is uttered if obligations exist in which *b* replies to a proposal to discharge the handing out of the books (which exist as obligations 28 and 29). The uttering of this acceptance results in the application of the following policies:

- Policy 2 (replying to a proposal discharges the obligations to reply): the acceptance to uptake the operation proposed in ι discharges the obligations to reply to ι (thus deleting obligations 28 and 29 on the buyer, and 20 and 21 on the carrier).
- Policy 3 (accepting a proposal causes the uptake of the proposed operation): the acceptance to uptake the operation proposed in ι causes the discharge of a commitment to hand out the books (which is a shared commitment that does not exist between *b* and *c*) as well as obligations 24 and 25 on the buyer, and 16 and 17 on the carrier), and lastly,

- Policy 4 (accepting to discharge a *ProposingToDischarge* action discards the obligations in which the *discharger* is to propose discharging the action): the acceptance to discharge the commitment to hand out the books results in the discharge of the obligations in which *c* is to propose discarding the commitment to hand out the books (therefore deleting obligations 26 and 27 on the buyer, and 18 and 19 on the contractor).

5.10.13 Conclusion

At this point the conversation has ended. However, it left several commitments and obligations left to discharge. That is the case of

- the buyer, who keeps a shared commitment with the seller in which the carrier is committed to hand out the books (labelled as commitment C in Figure 43 and Figure 46);
- the seller, who keeps this same shared commitment with the buyer, and also a shared commitment with the carrier in which he (the carrier) is to deliver the books to the buyer (labelled as commitment E in Figure 46 and Figure 48); and lastly
- the carrier, who keeps this same shared commitment with the seller in which he delivers the books to the buyer (plus the obligations to deliver the books).

To understand why these commitments and obligations are left after the interaction has ended, it is necessary to analyse the characteristics of common types of delivery, such as normal mail and courier delivery.

In the case of normal mail, the simplest scenario is that of a sender sending a letter to a receiver. There are two steps in this scenario: first, the sender deposits the letter in the post office; and second, the post office delivers the letter to the receiver (here the reference to the post office is intended as a reference of the mail system as a whole).

Similar to the conversation example above, depositing the letter in the post office creates a commitment between the sender and the post office for the delivery of the letter. Since there is normally no further communication between the post office and the sender, this commitment is never discharged.[48] This same result prevails in the case of registered deliveries (such as courier and express services), since there is still no proactive communication from the carrier to the sender to acknowledge the delivery, thus leaving the commitment unresolved.[49] Additionally, there are deliveries where the receiver is aware that a letter is being sent (which is the case of the above example, where the buyer and the seller shared a commitment in which the carrier delivers the books). Although this commitment could be discharged with additional communications (e.g., "I received the package", "Did you receive the package?"), these communications are not routinely used.[50]

The question now is what to do with these remaining commitments (and obligations). In the case of humans, senders may assume that a letter was delivered if no exception is produced (e.g., the letter is not returned) or if there is a confirmation of delivery from the receiver or the carrier (e.g., a replying letter, an acknowledgement). These events could justify the sender's discharge of the commitment that the carrier delivers the letter. Likewise, that a carrier has delivered a letter could justify him in discharging the commitment shared with the sender in which he delivers the letter. And lastly, that a receiver receives the letter she was expecting allows her to discharge the commitment with the informer agent (e.g., the seller in the above example) in which she is to receive a delivery.

---

[48] Compare this case with that in which the letter is sent back to the sender. This could be interpreted as a proposal to discharge the commitment to deliver given that an exception has occurred (e.g., the address does not exist, the receiver no longer resides in that address).

[49] Recently, Internet-based tracking systems (which report the processing centres through which a package has passed) have made it possible to follow the development of a delivery. Although such systems could be seen as a possible mechanism for justifying the discharge of commitments to deliver, they are not considered at this point.

[50] Interestingly, joint activity specifications support the spontaneous occurrence of conversations dealing with the state of the activity. For example, knowing that sellers deliver books through carriers allows buyers to ask sellers whether or not their requested books have been given to the carrier, or sellers to enquire carriers the status of their deliveries, or (as indicated above) sellers to ask buyers whether or not they have received their books from the carrier. Although such conversations can account for richer interactions, they are not explored in this thesis.

Interestingly, these assumptions leading to the discharge of commitments and obligations are not part of the model for conversations since they are not derived by the explicit negotiation of shared social commitments. Rather these rules depend on the cognitive ability of agents and the norms and traditions prevailing in the society of interaction, which is a topic that is discussed in a later chapter on future research.

5.11 Summary

This chapter presented an example of how the model for conversations can be applied to a practical domain, in this case, the selling and delivering of books. This example involved three agents: a buyer (who requests to buy books to a seller), a seller (who sells the books to a buyer, and requests their delivery to a carrier), and a carrier (who delivers the books to the buyer). In addition, the example involved three joint actions: selling, delivering and paying. Selling and delivering were defined as payable actions, that is, actions that entail a paying action leading to the production of a payment.

The interaction in the example was initiated with a buyer's request to a seller for buying books. This request was followed by the seller's acceptance to sell and a request for payment. This led to the buyer's payment, and the seller's acceptance of the payment and notification of the future delivery of the books, which signalled the end of the sale. At this point, the seller contacted the carrier to arrange delivery of the books. Once a payment was produced the carrier delivered the books to the buyer, thus ending the interaction.

It was pointed out that these interactions did not discharge all commitments and obligations adopted by the buyer, seller and carrier. Although additional communications could have discharged them it was noted that in human settings these commitments and obligations are usually discarded through reasoning (rather than communications). As will be explained in a later chapter, the integration of the observable rules in the model for conversations and the cognitive rules that affect the commitments and obligations established by these observable rules is left as an exercise for future research.

Chapter 6
Evaluation and Conclusion

6.1 Overview

This chapter revisits the requirements for a model for conversations set in Chapter 2, and evaluates whether or not the presented model for conversations satisfies these requirements. This analysis is followed by a brief comparison of this model with other approaches to agent communication languages and conversations (which were introduced in Chapter 2). This is followed by a brief account of future work in the areas of rational social action and deliberate normative agency. Lastly, this chapter concludes by revisiting the research objectives set at the beginning of the thesis (Chapter 1), and demonstrates how the material presented in this thesis satisfies those objectives

6.2 Revisiting the Requirements

The goal of this thesis was to define a model for the specification and analysis of software agent conversations. Two requirements were set in Chapter 2 for this model:

1. *To support agent conversations in open environments, a model for conversations must specify its message semantics using publicly verifiable principles.*
2. *To support the form of conversations, a model for conversations must define policies governing conversational composition.*

In the preceding chapters, this thesis presented a model for conversations (Chapter 3) and related examples (Chapters 4 and 5) that support these requirements.

6.2.1 Publicly Verifiable Semantics

One of the main contributions of this research is to have defined a model for conversations that is based exclusively on observable behaviour.

In this thesis it is assumed that agents bind their autonomy to the conversational norms in a society where they seek to collaborate with other agents. This same assumption can be made about the ACL surveyed in an earlier chapter in this thesis. In the case of ACL based on mental attributes, agent communications are not only intentional but are also assumed to be sincere. That means that agents must *assume* that others always say the truth (unless they reason otherwise) since the intended meaning of an utterance is not verifiable (without inspecting other agents' minds). In contrast, the model for conversations presented only assumes that communications have a perceivable aim.[51]

In this model, the meaning of a speech act is given by the emergent product of the identity of the elements in the speech act, the rules indicating how these elements are used in conversations, and the rules indicating their consequences, that is:

- The identity of elements is given by the definition of actions, obligations, social commitments, and the operations and various illocutionary points used toward the shared uptake of social commitments. Although at this level the model uses names that have a recognized meaning in human discourse, they are only intended as nametags (e.g., the illocutionary point subtype labelled *Propose*).
- The use of elements is regulated by the *Protocol for Proposals*, specifically by conversation policies 1 and 2. These policies reflect the conversational properties of sequencing, turn-taking and choice towards the shared uptake of social

[51] Just as in the case of human interactions, this assumption makes agents responsible for the messages they utter (and also it does not take into account agents with defective communication mechanisms).

commitments, that is, they specify that a proposal by Alice must be followed (sequencing) by either (choice) an acceptance or a rejection from Bob (turn-taking).

- Lastly, the consequences of using these elements is defined by conversation policy 3, which indicates that a proposal followed by an acceptance causes the uptake of the negotiated operation on a social commitment, thus affecting the state of shared social commitments and obligations of the interacting agents.

In this view, the meaning of uttering a proposal (for example) can be interpreted as a communication in which the speaker is putting forth for consideration an operation on a social commitment that is intended to affect the state of shared social commitments and obligations of agents.[52]

In addition, speech acts are compositional and can encompass several illocutionary points. This allows the interpretation of the meaning of a speech act as the combined meaning of its enclosed illocutionary points. Moreover, meaning can emerge at the level of the activity, where a speech act can be interpreted according to the interactions specified in the activity; and meaning can also emerge at a higher level, where a speech act could be interpreted according to the past interactions between agents.[53]

From these it is possible to conclude that the model for conversations specifies the principles to support a publicly verifiable message semantics (thus fulfilling the first requirement).

[52] Compare this definition with that of a proposal found in a common dictionary: "1: an act of putting forward or stating something for consideration." (Merriam-Webster, 2002).

[53] To exemplify these different levels of interpretation, imagine the case in which an agent is proposing to a travel agent to buy an airplane ticket to a resort destination abroad. By analysing this utterance outside of any context, one could imply just that: that the agent attempts to buy an airplane ticket. If analysed under the scope of an activity, let us say a vacations package activity, one could infer that the agent is planning to go on holidays (rather than on a business trip, for example). Lastly, that one analyses the utterance under a higher level of abstraction that includes the observed past interactions of the agent, one could conclude that, given that the agent was recently released on bail, the agent is avoiding the law and fleeing the country.

6.2.2 Conversational Composition

Joint activities specify the constraints that conversations should abide by. In particular they specify the sequencing of agent participations (i.e., the conversational moves), defining the order in which commitments to action are negotiated.

In addition to the fine-grained conversational composition supported by the PFP (i.e., that proposals must be followed by acceptances or rejections), the model for conversations defines policies that help to advance the state of joint activities. That is the case of policies 4 and 5 (which commit agents to propose the discharge or adoption of shared commitments), and policies 6 and 7 (which commit agents to request a payment for an action). Together, these policies support structured conversations for action, and are an improvement over traditional approaches to representing conversations protocols, which do not specify any formal semantics beyond those of their corresponding graphical representations.

In addition, the model for conversations supports the requirements for a conversation model set forth by Greaves, et al. (1999):

- *It must be independent from specific implementation techniques*: The presented model emphasizes observable behaviour, thus focusing on what lies between rather than within agents. This approach allows the interoperation of heterogeneous types of agents.
- *It must be flexible enough to allow dynamic context-dependent composition*: Conversational flexibility is supported by the PFP and is regulated by the interaction specified in joint activities. In the interaction examples presented in the preceding chapters, flexibility was sacrificed to highlight the systematic composition of conversations (it can be noticed that most PFP instances in these examples were mere pairs of proposals and acceptances). To highlight the flexibility of interactions within a single PFP instance, Figure 49 shows an example diagram containing all communications allowed in the PFP. To make this example

more engaging, imagine the interaction as part of a larger conversation in which a buyer is buying items from a seller, and where this PFP instance shows the communications to adopt a commitment in which the buyer pays for the items being bought (the representation of the speech acts in this example was greatly simplified to emphasize the illocutionary point and action being communicated). The interaction begins with the seller's proposal to pay. As shown in the figure, the allowed responses include the buyer's acceptance, the buyer or the seller's rejection, the buyer's counterproposal (thus allowing bargaining) and the seller's counterproposal to its own proposal. This latter case is similar to the initial proposal in the sense that any of the just mentioned replies could follow. This PFP instance ends if the buyer accepts, or if the buyer or the seller rejects. Lastly, in the case that the buyer counterproposes then the following could occur: the seller could accept, or both agents could reject (thus ending the PFP instance), or both agents could counterpropose (thus continuing the bargaining).[54] As a result, this example and the examples in previous chapters show that the PFP supports flexible conversations both within individual PFP instances and by composing sequences of PFP instances.

- *It must support conversations among agents of different levels of sophistication*: To be able to converse, agents do not need to implement all possible message sequences in a joint activity interaction; for example, less able agents may not allow counterproposals, which they could immediately reject upon occurrence. In this view, agent implementations could stretch from those less able agents that follow the most straightforward sequence of messages (and which may not even have any internal representation of the commitments negotiated), to agents with rational

[54] As implied, this interaction could continue indefinitely as long as agents keep counterproposing. At this point, the model does not consider the implications of deadlocks in conversations, since there might be different rational, context-dependent strategies to avoid and recover from such states (which is an analysis beyond the scope of this thesis).

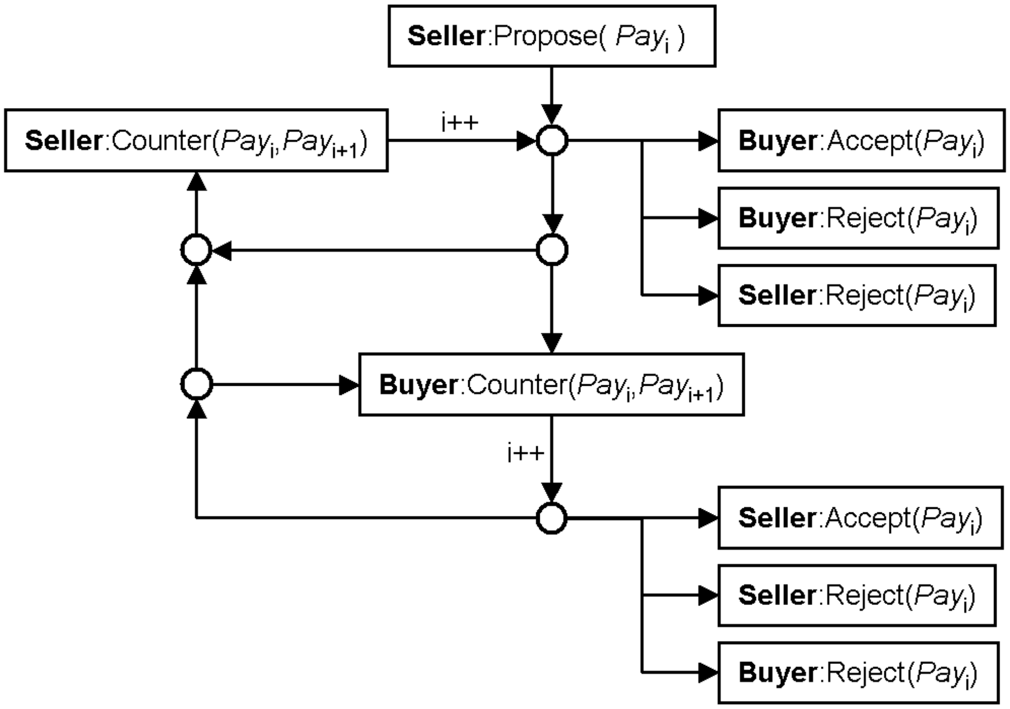


Figure 49. Example PFP instance interaction between a buyer and a seller.

engines allowing them to extend and refine their conversations to account for the context of occurrence. In any event, these agent types should be able to seamlessly interact as long as the conversation abides by the interaction specification in the activity.

6.3 Implementation

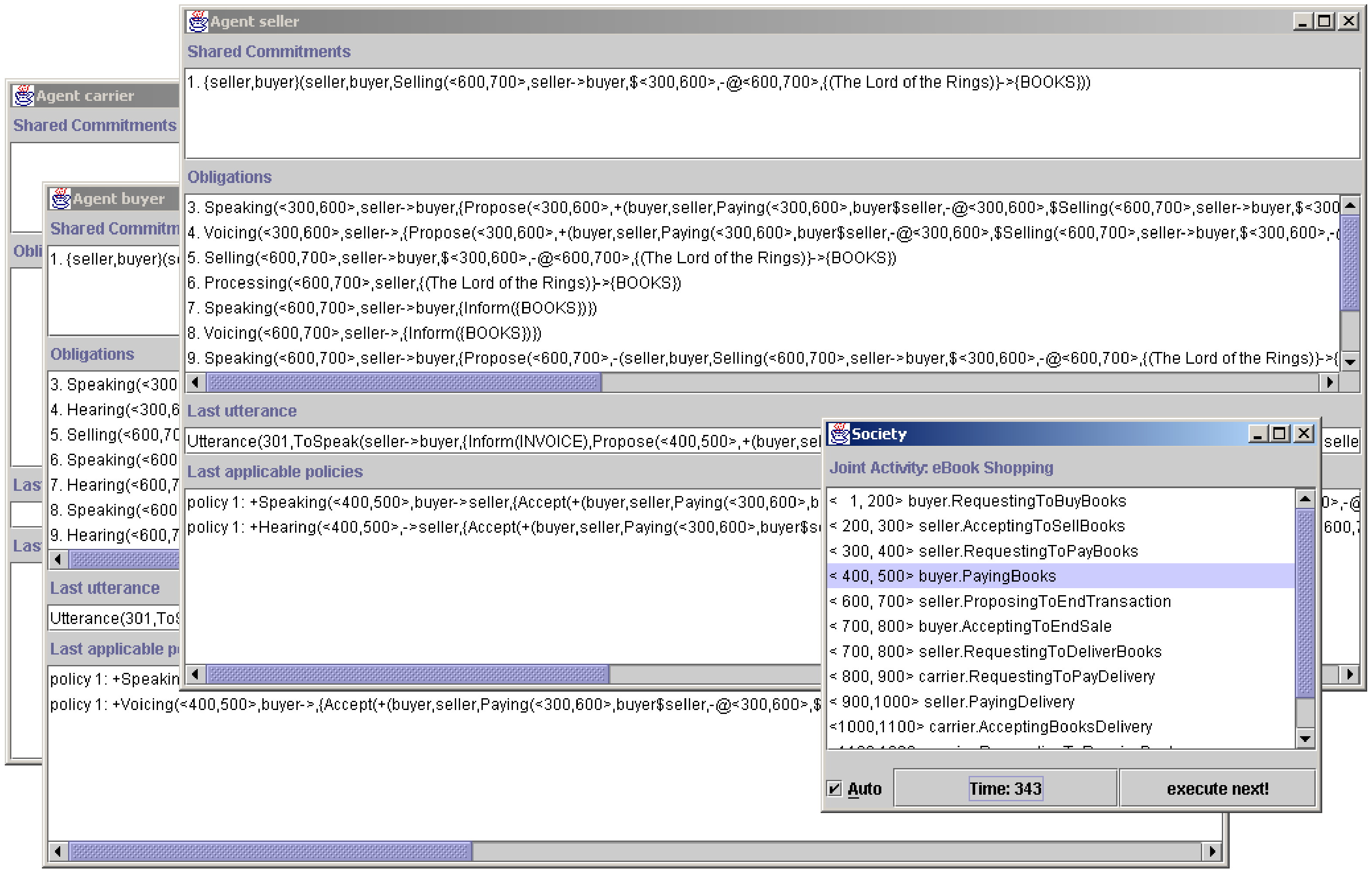
The model for conversations is designed as a framework for keeping track of the shared social commitments and obligations that are negotiated by agents through their communications. As seen throughout this thesis, the model was formalized using the Object-Z specification language. This language allows for a forthright translation from a formal specification to the object-oriented programming language of choice, which in this case was the Java programming language (Sun, 2002).

Figure 28 and Figure 50 show snapshots of the resulting test bed implementation. As shown in these figures, each interacting agent has a monitor window that displays the shared commitments and obligations it has adopted, as well as the conversation policies applied during the last communication it received. In accordance with the model for conversations, these agents are only representations of agents in the environment, and all the information maintained about them is based on their observed communications. As such, the agent representations implemented for the test bed (i.e., those depicted in the figures) do not send messages to each other, but rather are given the messages that are perceived to occur by the entity maintaining these representations. In the case of the test bed, these messages were predefined in the window object labelled "Society", which passes these messages to the agent representations for processing through the conversation policies of the model for conversations.

It is foreseen that future implementations of the model will have a more a practical application than just that of a static simulator of conversations. Presently, work is being pursued to implement a test bed engine for agents in the Collaborative Agent System Architecture (Flores, Kremer and Norrie, 2000). This work could also be adapted for agents in the FIPA's multi-agent systems architecture (FIPA, 1997).[55]

---

[55] In principle, any agent interacting in the FIPA architecture can be built with an engine that follows the specifications in the model for conversations. This, however, does not imply that FIPA conversation policies are subsumed by the principles in the model. This thesis asserts that conversation protocols are made-up message sequences constructed without any formal principles. This is true of the protocols supported by FIPA. As such, it is not expected that the sequencing of messages making a FIPA protocol (which do not have any principles behind their construction) can be explained using the model for conversations. On the other hand, it is the contention of this thesis that the principles of the model are general enough as to account for non-trivial conversations for action. That is, that conversations such as the Contract Net Protocol have been successfully modelled gives a certain assurance that simpler conversations (such as queries and requests) can be modelled as well. This, however, does not justify the claim that the model for conversations could model all conversations for action; to assert such a claim, one would need to model all conversations for action that could possible exist (certainly a daunting if not impossible task); on the other hand, to negate this claim (i.e., that the model cannot be used for all conversations for action), one would need to demonstrate that there exists at least one conversation for action that cannot be specified using the model. Since both of these issues are still to be resolved, the claim of the absolute generality of the model for conversations to specify conversations for action cannot presently be justified (or denied).



```
Agent seller
Shared Commitments
1. {seller,buyer}(seller,buyer,Selling(<600,700>,seller->buyer,$<300,600>,-@<600,700>,{(The Lord of the Rings)}->{BOOKS}))
Obligations
3. Speaking(<300,600>,seller->buyer,{Propose(<300,600>,+(buyer,seller,Paying(<300,600>,buyer$seller,-@<300,600>,$Selling(<600,700>,seller->buyer,$<300
4. Voicing(<300,600>,seller->,{Propose(<300,600>,+(buyer,seller,Paying(<300,600>,buyer$seller,-@<300,600>,$Selling(<600,700>,seller->buyer,$<300,600>,-
5. Selling(<600,700>,seller->buyer,$<300,600>,-@<600,700>,{(The Lord of the Rings)}->{BOOKS})
6. Processing(<600,700>,seller,{(The Lord of the Rings)}->{BOOKS})
7. Speaking(<600,700>,seller->buyer,{Inform({BOOKS})})
8. Voicing(<600,700>,seller->,{Inform({BOOKS})})
9. Speaking(<600,700>,seller->buyer,{Propose(<600,700>,-(seller,buyer,Selling(<600,700>,seller->buyer,$<300,600>,-@<600,700>,{(The Lord of the Rings)}->{
Last utterance
Utterance(301,ToSpeak(seller->buyer,{Inform(INVOICE),Propose(<400,500>,+(buyer,sel                                        selle
Last applicable policies
policy 1: +Speaking(<400,500>,buyer->seller,{Accept(+(buyer,seller,Paying(<300,600>,b                                    0>,-@
policy 1: +Hearing(<400,500>,->seller,{Accept(+(buyer,seller,Paying(<300,600>,buyer$s                                    600,

Agent carrier
Shared Commitments
Obli
Las
Las

Agent buyer
Shared Commitm
1. {seller,buyer}(s
Obligations
3. Speaking(<300
4. Hearing(<300,6
5. Selling(<600,70
6. Speaking(<600
7. Hearing(<600,7
8. Speaking(<600
9. Hearing(<600,7
Last utterance
Utterance(301,To
Last applicable p
policy 1: +Speakin
policy 1: +Voicing(<400,500>,buyer->,{Accept(+(buyer,seller,Paying(<300,600>,buyer$seller,-@<300,600>,$

Society
Joint Activity: eBook Shopping
<   1, 200> buyer.RequestingToBuyBooks
< 200, 300> seller.AcceptingToSellBooks
< 300, 400> seller.RequestingToPayBooks
< 400, 500> buyer.PayingBooks
< 600, 700> seller.ProposingToEndTransaction
< 700, 800> buyer.AcceptingToEndSale
< 700, 800> seller.RequestingToDeliverBooks
< 800, 900> carrier.RequestingToPayDelivery
< 900,1000> seller.PayingDelivery
<1000,1100> carrier.AcceptingBooksDelivery
☑ Auto    Time: 343    execute next!
```

Figure 50. Snapshot of the simulation of the eBookstore conversation example.

In terms of performance and efficiency of implementation, it is not straightforward to derive whether *ad hoc* conversation protocols or the model for conversations better support agent conversations. It remains uncertain since both approaches can be implemented using the same techniques: the model for conversations can be implemented as rules of inference for knowledge bases or can be used for building protocols, which then can be implemented either as sequences of operations in programming languages, or also as rules of inference in knowledge bases. Arguably, the implementation of protocols using regular procedural or object-oriented programming languages may provide more efficient implementations (e.g., in terms of speed and code size, which might be especially desirable for simple agents) than those implementations based on knowledge bases.

At this point, the issue becomes whether or not *ad hoc* protocols have any advantages over those protocols constructed using the model for conversations. Empirically, one advantage is that *ad hoc* protocols use less messages for accomplishing a task than protocols based on the model for conversations. Even without an exhaustive analysis, it can be derived that this assertion is likely to be true.[56]

For example, a protocol for giving the time could be constructed by using only three messages: an initial message for requesting the time, and two other messages that could follow it: one for rejecting giving the time, and another one for communicating the time. Therefore, asking and getting the time using this protocol requires exactly two messages; this contrasts with the minimum of three messages for doing the same task using the model for conversations (as illustrated by the example in Chapter 1). As such, there is a one-message (or 50%) overhead for the model for conversations over this *ad hoc* protocol.

The Contract Net Protocol example provides a further point for comparison. In the case of the FIPA contract net protocol, the minimum number of messages for successfully

[56] It is important to remark that this thesis is not concerned with the study of current protocols as to understand the assumptions made by designers when constructing these protocols. The contention is that designers do make assumptions (e.g., whether or not an acknowledgement is required after an inform), and those assumptions are likely to result in simpler protocols.

delegating and executing an action is four.[57] In contrast, the minimum number of messages in the Contract Net example presented in this thesis is five.[58] Although this represents still has a one-message overhead, the proportional difference is now reduced to 25%. It is worthwhile noticing that, even though these two examples are not enough for generalizing, they do hint that the proportional difference in the number of messages between *ad hoc* protocols and protocols based on the model for conversations may get smaller as the number of messages in an activity increases.

Lastly, probably the main disadvantage of the model for conversations over *ad hoc* protocols is the complexity associated with the construction of protocols using the principles in the model. That is, although the model for conversations is based on principles and *ad hoc* protocols are not, having such principles plays to the model's disadvantage, since the structuring of conversations must abide to strict rules of inference (the reader may recall the level of detail required to infer small transitions in the proof shown in Chapter 4). It will not be until tools that reduce the load associated with this inference process are made available that the construction of protocols will become more accessible to the every day designer of multi-agent system conversations. The development of such tools is still an issue open to explore.

On the other hand, the conversations based on *ad hoc* protocols are normally less flexible than conversations in the model for conversations. To illustrate this flexibility, we calculate the number of different conversations that could exist in an activity that uses the protocol, in particular the CNP activity presented in Chapter 4. For that, the following variables and formulae are defined

[57] This number is obtained from the sequence of messages “cfp”, “propose”, “accept-proposal”, and “inform” specified in the FIPA contract net protocol (FIPA, 1997).

[58] This number is obtained from the sequence of messages “RequestingBid”, “AcceptingToBid + SubmittingBid”, “EvaluatingBid + AwardingContract”, “AcceptingAward + SubmittingResult” and “AcceptingResults”, which are specified in the interaction protocol in Figure 24.

- the depth of a PFP instance (denoted as κ, where $\kappa \in \mathbb{N}_1$) is the maximum number of consecutive counterproposal illocutions in a PFP instance. For example, that a PFP instance has a $\kappa=1$ indicates that any counterproposal is automatically rejected (thus restricting the depth of the PFP); if $\kappa=2$, then the second consecutive counterproposal is rejected; if $\kappa=3$, then the third consecutive counterproposal is rejected; and so on.

- *A* is specified as the number of possible accepting sequences (i.e., sequences terminating with an *Accept* illocutionary point) for any PFP instance of depth κ. This variable is formulated as

$$A = \sum_{i=\kappa}^{i=1} 2^{i-1}$$

- *R* is defined as the number of possible rejecting sequences for any PFP instance of depth κ (including the counterproposals that are rejected at depth κ). This variable is formulated as

$$R = 2^{\kappa} + \sum_{i=\kappa}^{i=1} 2^{i}$$

- η (where $\eta \in \mathbb{N}_1$) is defined as the number of a series of consecutive PFP instances of depth κ advancing the state of an activity, and *N* is specified as the number of all accepting sequences in a series of consecutive PFP instances. *N* is formulated in terms of *A* and η as

$$N = A^{\eta}$$

- Lastly, *T* is defined as the total number of illocution sequences (either accepting or rejecting) that could exist in a sequence of η consecutive PFP instances of level κ.[59] This variable is calculated through the following series

[59] The definition of *N* and *T* are only intended to reflect certain characteristics of CNP conversations. Some of the assumptions made for these definitions are: they are intended for conversations where PFP instances do

$$T = R_1 + A_1(R_2 + A_2(\ldots(R_{\eta\text{-}1} + A_{\eta\text{-}1}(R_\eta + A_\eta))\ldots))$$

Given the above, one could calculate *N* (the number of possible sequences that could successfully carry out the activity) and *T* (the total number of sequences in a series of PFP instances) for the CNP conversation presented in Chapter 4, where the number of consecutive PFP instances required to successfully carry out the execution of a contract was four (i.e., η=4). If the depth of these instances is arbitrarily set to three (i.e., κ=3), then it is possible to conclude that there could exist at least 2.401 different conversations (given by *N*) that could successfully carry out a contract in the CNP example. Moreover, the total number of different conversations that could exist (given by *T*) would be at least 11.201. This contrasts with the one conversation sequence that can possibly exist for doing the same task in the FIPA contract net protocol, thus showing that the model for conversations can be more flexible than *ad hoc* conversation protocols.

6.4 Research Contributions

Prevalent ACL were briefly surveyed earlier in Chapter 2. These ACL were classified according to the principles they used for defining the meaning of their messages as either based on mental attributes or based on social commitments. ACL based on mental attributes (i.e., JIT, FIPA ACL and KQML) have the advantage of simplifying the inference of speakers' intentions from utterances by directly defining such intentions (or a minimal subset of them) as the meaning of their utterances. However, this approach is only useful under the assumption that agents are always sincere, that is, that their utterances undoubtedly reflect their mental states. Since this sincerity condition has been widely

not occur concurrently (as in the case of the eBookstore example in Chapter 5); for conversations where there is only one sequence of PFP instances such as the acceptance in one leads to the proposal in the next (e.g., the contractor's acceptance to submit a bid leads to a proposal to have the bid evaluated); and for conversations where there are no exceptions or abnormal terminations (e.g., a contractor that has accepted to submit a bid proposes to discharge this commitment when he realizes that he cannot fulfill it). As such, any outcome of using these definitions to compute the number of different conversation instances is less than the possible number that could exist.

criticized as a viable option for open environments, this thesis set out to explore utterance meaning under a different light: that of meaning based on observable behaviour, particularly, through the use of social commitments to action.

This use of social commitments to define utterance meaning is the approach also taken by the social ACL surveyed in Chapter 2 (i.e., Singh and Colombetti's models).[60] In general, there exist conceptual differences between these models and the model for conversations. In the case of Singh's model, the meaning of utterances is defined through a three-layered classification of their objective, subjective and practical meaning (where the objective layer defines their public meaning, the subjective layer defines their meaning under a state of sincerity, and the practical layer defines the contextual justifications for their occurrence). In this view, Singh's model covers a wider range of meaning than the model for conversations (which focuses on the objective layer), and the various ACL based on mental states (which mainly address the subjective layer).[61] In the case of Colombetti's Albatross, speech acts are defined as actions, which is an approach that is conceptually similar to the definition of speech acts in the model for conversations. More important, however, is Colombetti's recognition that the isolated utterance of speech acts cannot result in the establishment of social commitments (which is an issue still to be adopted by Singh's model). Rather, he argues, utterances create pre-commitments that must be ratified (through subsequent speech acts) to become social commitments. This explicit agreement to establish commitments is similar in spirit to the negotiation of commitments in the model

[60] One other approach to conversations based on commitments (which was recently brought to the attention of the author) is the model presented by P. McBurney and S. Parsons (2002), which is based on the typology of human dialogues for argumentation described by D.N. Walton and E.C.W. Krabbe (Walton & Krabbe, 1995). Although the model for conversations may be seen as more applied than this model, there are conceptual parallels between them, such as the notions of dialogic and semantic commitments (where the former are commitments that are only meaningful within dialogues, and the latter are commitments that have consequences outside the dialogue). These commitments could be paralleled to those generated by policies 1 and 2 (which are intended only for advancing conversations), and policy 3 (which affects the state of shared commitments and obligations of the agents), for example.

[61] Congruently to his stance on heterogeneity, Singh dismisses subjective meaning as a viable semantic element for utterances in open environments.

for conversations; nevertheless, the notion of pre-commitments is not yet developed in Albatross (as it is done in the model for conversations) to account for the systematic sequencing and turn-taking in conversations.

It is probably for this issue (the form of conversations) that the model for conversations stands out over current ACL. Presently, most of these ACL completely rely on *ad hoc* conversation protocols to guide their conversations. The exception is Singh's commitment machines model (Yolum & Singh, 2001), which (in the same vein as in the model for conversations) defines that the uttering of speech acts declares social commitments that are adopted and discharged as conversations evolve. Although their objectives are similar, the model for conversations and Singh's model have important differences. Particularly, the model for conversations explicitly accounts for the sequencing and turn-taking in conversations, and it does so through the explicit negotiation of fine-grained conversational commitments (such as, to speak, to voice and to hear). That is, Singh's model makes use of social commitments for describing protocols, but these commitments do not define the structure of these protocols (in other words, sequencing and turn-taking is still an implicit property of the protocols used). An additional advantage of the model for conversations is that it provides more for the autonomy of agents by negotiating the uptake of commitments (rather than just assuming that a commitment exists once an utterance has occurred), which is especially important when utterances are intended to commit agents other than the speaker. This is an issue that is still to be address by Singh's model.

To recap, the main contribution of this thesis is to have defined a model for conversations that supports the systematic composition of conversations (i.e., the form of conversations) by using observable behaviour (and thus supports agents in open environments). This model is an improvement over current ACL in both its account of utterance meaning based on observable behaviour, and its systematic approach to structure conversations.

6.5 Future Research

At present, the model for conversations specifies a mechanism for mutual agreement to establish responsibilities toward the execution of actions. This model is based exclusively on observable behaviour, and it is independent (but complementary) to the cognition guiding the actions of goal-directed agents.

The importance of social commitments is that they provide for a principled way to connect the external world of interactions with the internal world of individual rational action. As such, one main avenue for future research is to investigate the value of social commitments in bridging the concepts of rationality (which are inherently private) and conversations (which are public social phenomena). One area to explore is that of theories of individual social action (Conte & Castelfranchi, 1995), specifically those dealing with the modelling of deliberate normative agents, that is, agents that guide their behaviour by reasoning about the norms in their society (Castelfranchi, et al., 2000). Moreover, by being able to reason about norms, agents could infer the normative reasoning of other agents based on their observable communications, which allows predicting and influencing future behaviour. This type of agent could then be applied to open environments where the coordination of action is regulated and globally optimized by monitoring agents (e.g., Pechoucek & Norrie, 2000).

6.6 Revisiting Research Objectives

Five research objectives were defined at the beginning of this thesis. These objectives were:

1. To survey the state of the art on agent communication languages to find their adequacy to support agent conversations.
2. To define the requirements that a model for conversations should support.

3. To propose a model for conversations that fulfills the requirements previously obtained.
4. To evaluate the feasibility of the model for conversation in a range of practical domains.
5. To propose further research based on the experiences obtained.

6.6.1 Related Work

The first objective was:

To survey the state of the art on agent communication languages to find their adequacy to support agent conversations.

This objective was met in Chapter 2, where current agent communicational languages were surveyed. These ACL were analysed in two separate groups: those based on mental attributes (Cohen and Levesque's Joint Intention Theory, FIPA ACL and KQML) and those based on social commitments (Collombetti's and Singh's models).

6.6.2 Requirements

The second objective was:

To define the requirements that a model for conversations should support.

This objective was also met in Chapter 2, where two requirements were set for a model for conversations. First, it was identified that ACL based on mental states are not adequate to support the interaction of heterogeneous agents, since speech acts in these ACL are defined using rational conditions that cannot be verified in open environments. As a result, it was specified that ACL in open environments must have a verifiable semantics. Second, it was identified that all of the surveyed ACL relied on conversation protocols to structure conversations. As a result, it was specified that a model for conversations must define rules that allow the composing of conversations.

6.6.3 A Model for Conversations

The third objective was:

To propose a model for conversations that fulfills the requirements previously obtained.

This objective was met in Chapter 3, where a model for conversations was defined. This model lays down the principles upon which a verifiable semantics for conversations can be built (namely identity, use and consequences), and specifies a model for structuring conversations based on the negotiation of shared social commitments and obligations.

6.6.4 Application

The fourth objective was:

To evaluate the feasibility of the model for conversations in a range of practical domains.

This objective was met in Chapters 4 and 5, where examples of the application of the model for conversations were presented. These examples specify the interaction of agents in an implementation of the Contract Net Protocol (which is a task allocation mechanism frequently used in multi-agent systems), and an e-commerce application (in which a buyer buys books from a seller, who then contracts with a carrier for delivering the books).

6.6.5 Future Work

The fifth and last objective was:

To propose further research based on the experiences obtained.

This objective was met in Chapter 6 (the current chapter), where examples were given of future development of the model. Specifically, its was noted that the model can be complemented with formal rational frameworks for speech acts and social rational action.

6.7 Fulfilling the Aim

The aim of this thesis was to define a model for the structured specification of software agent conversations for action. The objectives set for the thesis (and reviewed above) provided a disciplined approach to satisfying this aim: first, by identifying the shortcomings of current ACL, and then by proposing a model for conversations that overcomes these shortcomings. This was followed by practical examples that illustrated the applicability of the model to support the principled specification of conversations for action. In particular, these examples showed how instances of the Protocol for Proposals could be used as the basic conversational components to assemble the interactions in joint activities, and how these interactions are constrained to indicate the characteristics of actions, such as their sequencing and the agent roles involved in their performance. Once assembled, these interactions are applied to govern the evolution of conversations in the context of their corresponding joint activity.

6.8 Conclusion

This thesis presented a model for the structured specification of software agent conversations for action. This model structures conversations using conversation policies whose principle is the negotiation of shared social commitments to action.

This thesis presented the current state of the art on agent communication languages and showed some of their weaknesses in supporting the flexible communication of agents in open environments, specifically their reliance on non-verifiable rational components for speech act semantics, and on static conversation protocols for the composition of conversations. The research presented in this thesis describes a model for conversation that advances on such issues by a) presenting the principles upon which a publicly verifiable message semantics can be constructed, and b) presenting a framework for the definition of flexible conversations that follow conversation policies. The application of this model was illustrated through two conversation examples, one on the Contract Net Protocol, and the other on a generalized example of an e-commerce scenario. Lastly, this thesis presented an evaluation of this model and a brief account of future research.

References

Austin, J.L. *How to Do Things with Words*. Harvard University Press, 1962.

Bradshaw, J.M., Dutfield, S., Benoit, P. and Woolley, J.D. KAoS: Toward An Industrial-Strength Open Agent Architecture. In J.M. Bradshaw (Ed.), *Software Agents*, AAAI Press, 1997, pp. 375-418.

Bratman, M.E. What is Intention? In P.R. Cohen, J. Morgan and M.E. Pollack (Eds.), *Intentions in Communication*, MIT Press, pp. 15-31, 1990.

Castelfranchi, C., Dignum, F., Jonker, C. and Treur, J. Deliberate normative agents: principles and architectures. In *Intelligent Agents VI*, Lecture Notes in Artificial Intelligence 1757, Proceedings of the 6th International Workshop on Agent Theories, Architectures, and Languages (ATAL '99). Springer, Springer Verlag, Berlin, 2000, pp. 206-220.

Centre for Applied Formal Methods (CAFM). *The Z Notation*, 2002. http://www.afm.sbu.ac.uk/z/

Clark, H.H. *Using language*. Cambridge University Press, 1996.

Cohen, P.R. and Levesque, H.J. Rational Interaction as the Basis for Communication. In P.R. Cohen, J. Morgan and M.E. Pollack (Eds.), *Intentions in Communication*, MIT Press, pp. 221-255, 1990.

Cohen, P.R. and Levesque, H.J. Intention is Choice with Commitment. In *Artificial Intelligence*, Elsevier Science Publishers, Number 42, pp. 213-261, 1990.

Cohen, P.R., Morgan, J. and Pollack, M.E. Introduction. In P.R. Cohen, J. Morgan and M.E. Pollack (Eds.), *Intentions in Communication*, MIT Press, pp. 1-13, 1990.

Colombetti, M. A Commitment-based Approach to Agent Speech Acts and Conversations. In M. Greaves, F. Dignum, J. Bradshaw, and B. Chaib-draa (Eds.), *Proceedings of the Workshop on Agent Languages and Conversation Policies*, 4th International Conference on Autonomous Agents (Agents 2000), Barcelona, Spain, pp. 21-29, 2000.

Conte, R. and Castelfranchi, C. *Cognitive and Social Action*. University College London Press, 1995.

Conte R. and Dellarocas, C. *Social Order in Multiagent Systems*. Kluwer Academic Publishers, 2001.

Cost, R.S., Chen, Y., Finin, T., Labrou, Y. and Peng, Y. Modeling Agent Conversations with Colored Petri Nets. In M. Greaves and J. Bradshaw (Eds.), *Proceedings of the Workshop on Specifying and Implementing Conversation Policies*, 3rd International Conference in Autonomous Agents (Agents '99), Seattle, WA, pp. 59-66, 1999.

Craig, R.T. and Tracy, K. Introduction. In R.T. Craig and K. Tracy (Eds.), *Conversational Coherence: Form, Structure, and Strategy*. Sage Publications, pp. 10-22, 1983.

Diller, A. *Z: An Introduction to Formal Methods*. John Wiley and Sons, 1990.

Fagin, R., Halpern, J.Y., Moses, Y. and Vardi, M.Y. *Reasoning about Knowledge*. MIT Press, Cambridge, Massachusetts, 1995.

Ferber, J. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Publishing, Co., 1999.

Finin, T., Labrou, Y. and Mayfield, J. KQML as an Agent Communication Language. In J.M. Bradshaw (Ed.), *Software Agents*, AAAI Press / The MIT Press, 1997, pp. 291-316.

Flores, R.A., Kremer, R.C., and Norrie, D.H. An Architecture for Modeling Internet-based Collaborative Agent Systems. In T. Wagner and O.F. Rana (Eds.), *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, Lecture Notes in Computer Science, Volume 1887, Springer Verlag, 2001, pp. 56-63.

Foundation for Intelligent Physical Agents (FIPA). *FIPA Specifications*, Version 1, 1997. http://www.fipa.org

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading Mass., Addison Wesley, 1995.

Genesereth, M.R. and Ketchpel, S.P. Software Agents. In *Communications of the ACM*, Volume 37, Number 7, pp. 48-53, July 1994.

Greaves, M., Holmback, H. and Bradshaw, J. What is a Conversation Policy? In M. Greaves and J. Bradshaw (Eds.), *Proceedings of the Workshop on Specifying and Implementing Conversation Policies*, 3rd International Conference in Autonomous Agents (Agents '99), Seattle, WA, pp. 1-9, 1999.

Habermas, J. *The Theory of Communicative Action*. Beacon Press, 1984.

Jennings, N.R. and Wooldridge, M.J. *Agent Technology: Foundations, Applications and Markets*. Springer-Verlag, 1998.

Kumar, S., Huber, M., McGee, D., Cohen, P., and Levesque, H. Semantics of agent communication languages for group interaction. *Proceedings of the 17th National Conference on Artificial Intelligence*, AAAI Press/The MIT Press, pp. 42-47, Austin, Texas, 2000.

Labrou, Y. Semantics for an Agent Communication Language. Doctoral Dissertation, Department of Computer Science and Electrical Engineering, University of

Department of Computer Science and Electrical Engineering, University of Maryland, 1997.

Labrou, Y. and Finin, T. Semantics and Conversations for an Agent Communication Language. In M.N. Huhns and M.P. Singh (Eds.), *Readings in Agents*, Morgan Kaufmann Publishers, 1999, pp. 235-242.

Merriam-Webster. *Collegiate Dictionary*. Online edition, 2002. http://www.m-w.com/

Odell, J.J., Parunak, H.V.D. & Bauer, B. Representing Agent Interaction Protocols in UML. In P. Ciancarini & M.J. Wooldridge (Eds.), *Agent-Oriented Software Engineering*, LNCS 1957, Springer, 2001.

Pechoucek, M. and Norrie, D.H. *Knowledge Structures for Reflective Multi-Agent Systems: On Reasoning about other Agents*. Report 538. University of Calgary, Department of Mechanical and Manufacturing Engineering, 2000.

Perrault, C.R. An Application of Default Logic to Speech Act Theory. In P.R. Cohen, J. Morgan and M.E. Pollack (Eds.), *Intentions in Communication*, MIT Press, pp. 161-185, 1990.

Sadock, J.M. Comments on Vanderveken and on Cohen and Levesque. In P.R. Cohen, J. Morgan and M.E. Pollack (Eds.), *Intentions in Communication*, MIT Press, pp. 257-270, 1990.

Searle, J.R. *Expression and Meaning*, Cambridge University Press, 1975.

Singh, M.P. Agent Communicational Languages: Rethinking the Principles. *IEEE Computer*, Volume 31, Number 12, pp. 40-47, 1998.

Singh, M.P. A Social Semantics for Agent Communication Languages. In F. Dignum, B. Chaib-draa and H. Weigand (Eds.), *Proceedings of the Workshop on Agent Communication Languages*, International Joint Conference in Artificial Intelligence (IJCAI '99), Stockholm, Sweden, 1999.

Singh, M.P., Rao, A.S. and Georgeff, M.P. Formal Methods in DAI: Logic-Based Representation and Reasoning. In G. Weiss (Ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, pp. 331-376, 1999.

Smith, I.A., Cohen, P.R., Bradshaw, J.M., Greaves, M. and Holmback, H. Designing Conversation Policies using Joint Intention Theory. In *Proceedings of the 3rd International Conference on Multi-Agent Systems* (ICMAS), Paris, France, pp. 269-276, 1998.

Smith, G. *The Object-Z Specification Language*, Kluwer Academic Publishers, 2000.

Smith, G. *The Object-Z Home Page*. 2002. http://www.itee.uq.edu.au/~smith/objectz.html

Smith, R.G. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. In *IEEE Transactions on Computers*, Volume 29, Number 12, pp. 1104-1113, December 1980.

Software Verification Research Centre (SVRC). *Wizard: A Type-Checker for Object-Z Specifications*, 2002. http://svrc.it.uq.edu.au/Object-Z/pages/Wizard.html

Spivey, J.M. *The Z Notation: A Reference Manual*. Prentice Hall, 1992. Available on-line at http://spivey.oriel.ox.ac.uk/~mike/zrm/

Sun Microsystems (2002) *The Source for Java™ Technology*. http://java.sun.com/

Tannen, D. *That's Not What I Meant!* Ballantine, 1986.

Walton, D.N. and Krabbe, E.C.W. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of New York Press, 1995.

Winograd, T. and Flores, F. *Understanding Computers and Cognition: A New Foundation for Design*. Addison-Wesley Publishing Company, Inc., 1987.

Wooldridge, M. Verifiable Semantics for Agent Communication Languages. In Y. Demazeau (Ed.), *Proceedings of the 3rd International Conference on Multi-Agent Systems* (ICMAS '98), IEEE Press, 1998.

Yolum, P. and Singh, M.P. *Synthesizing Finite State Machines for Communication Protocols*. Technical Report TR-2001-06. North Carolina State University, Department of Computer Science, 2001.

Appendix A
Z and Object-Z Tutorial

This appendix presents a brief tutorial of the Z and Object-Z specification languages. This tutorial aims to familiarize readers with the notions used to formalize the model for conversations described in this thesis. As such, this tutorial is by no means an attempt to cover all concepts in Z and Object-Z. A comprehensive description of these languages can be found in (Spivey, 1992) and (Diller, 1990), and (Smith, 2000), respectively.[62]

A.1 Overview

Smith (2000) defines Object-Z as an extension of the Z specification language (Spivey, 1992) that facilitates the specification of systems in an object-oriented style. On the one hand, that Object-Z is an extension of Z indicates that it retains the existing familiar syntax and semantics of Z. On the other hand, that it facilitates the specification of systems in object-oriented style indicates that it incorporates object-oriented notions such as classes, inheritance and polymorphism.

[62] Additional resources on Z and Object-Z can be found in the World-Wide Web at the Object-Z Home Page (Smith, 2002) and The Z Notation (CAFM, 2002) web pages.

A.2 The Z Specification Language

Z is a formal specification language based on first-order predicate logic and set theory. This language helps to unambiguously describe the characteristics of computer systems while abstracting from implementation details (i.e., it "describe[s] *what* the system must do without saying *how* it is done." (Spivey, 1992, p. 1)).

The following sections describe the main characteristics of this specification language.

A.2.1 Types

Z defines types as sets, that is, as collections of unique elements. Some of the most common types in Z are $\mathbb{N}$ (the type of all natural numbers), $\mathbb{N}_1$ (the type of all natural numbers greater than zero), and $\mathbb{R}$ (the type of all real numbers).

The easiest way to define types in Z is by using abbreviated definition, whose symbol is ==. An abbreviated definition makes use of existing types to create a new one; for example, the definition "$Time == \mathbb{N}$" indicates that the type *Time* is the same as the type of all natural numbers.

Beside abbreviated definitions, there are two other approaches to define types: by enumeration (i.e., by listing all members of the set), and by comprehension (i.e., by specifying the properties of their elements). For example, the definition "{1, 2, 3, 4, 5}" simply enumerates a set of natural numbers. The property of this set is that it contains all natural numbers between 0 and 6. As such, this set can also be described using set comprehension, for which one possible definition could be "$\{n: \mathbb{N}_1 \mid n < 6 \bullet n\}$" (this can be read as "all positive natural numbers *n*, such that *n* is less than 6, are the elements that compose this set").

A.2.2 Objects

Objects are instances of types. They are declared using an object name followed by a colon and a type name. For example, the declaration "$x, y, z: \mathbb{N}$" indicates that the type of each of the objects x, y and z is a natural number. There are also cases in which objects do not reference only one instance (as in the previous example), but a set of instances of a certain type. These sets are defined using the power set symbol $\mathbb{P}$. For example, the declaration "x: $\mathbb{P}\ \mathbb{R}$" indicates that x is a set (which well could be empty) of real numbers. The symbol $\mathbb{P}_1$ is used to explicitly specify sets with at least one element. The empty set is represented either as $\varnothing$ or $\{\}$.

A.2.3 Propositions

Propositions are expressions that are either true or false. Simple propositions can be grouped to create more sophisticated propositions by using logical connectives such as negation ($\neg$), disjunction ($\wedge$), conjunction ($\vee$), implication ($\Rightarrow$) and bi-implication ($\Leftrightarrow$). For example, the expression "$\neg P \Rightarrow Q$" is a proposition that can be read as "not P implies Q", where P and Q are themselves propositions.

Abstract propositions can be defined by using the universal and existential quantifiers. The universal quantifier (represented by $\forall$) specifies the properties of all the objects that abide to certain characteristics. For example, to indicate that all natural numbers smaller than 50 are also smaller than 100, one could use the expression "$\forall x$: $\mathbb{N} \mid x < 50 \bullet x < 100$", which can be read as "for all natural numbers x, such as x is smaller than 50, then x is smaller than 100." On the other hand, the existential quantifier (denoted as $\exists$) is used to indicate that certain properties are true for at least one object that abides to certain characteristics. For example, to indicate that there is at least one even number smaller than 50, one could use the expression "$\exists x$: $\mathbb{N} \mid x < 50 \bullet x \bmod 2 = 0$", which can be read as "there exists at least one natural number x, such that x is smaller than 50, such that the remainder of dividing x by 2 is zero." Lastly, the unique existential quantifier (denoted as $\exists_1$) is a specialization of the existential quantifier that does not only indicate that there exists an element with the specified characteristics but also that there exists exactly one such element.

Set Propositions

Various set predicates and operations can be used to produce set propositions. Predicates include the following: set membership ($\in$), which expresses that an object is an element of a set; set equality (=), which indicates that one set is equal to another set (i.e., they both have identical elements); and set inclusion ($\subseteq$), which expresses that a set is a subset of another set. The symbols $\notin$, $\neq$, and $\not\subset$ specify the negation of the aforementioned predicates (i.e., that an object is not a member of a set, that two sets are not the same, and that a set is not a subset of another set, respectively). In addition, the symbol $\subset$ is applied to express that a set is a proper subset of another set (where proper subset means that the first set is contained in the second set, and that the sets are not equal). Examples of set predicates are the expressions "$1 \in \mathbb{N}$" (which indicates that the number 1 is an element of the set of natural numbers), "$\{1, 2, 3\} = \{3, 2, 1\}$" (which indicates that the indicated sets are equal), and "$\mathbb{N} \subset \mathbb{R}$" (which indicates that the set of natural numbers is a proper subset of the set of real numbers).

In addition to these predicates, there are three common operations for sets. These are: union ($\cup$), difference ($\setminus$) and intersection ($\cap$). To illustrate these operations, let us define the two sets "$\{1, 2, 3\}$" and "$\{2, 3, 4\}$". On the one hand, the union of these two sets results in a set encompassing all their elements, that is, the set "$\{1, 2, 3, 4\}$"; their difference results in a new set containing those elements that are in the first set that are not in the second, that is, the set "$\{1\}$"; and lastly, the intersection of the two sets results in a new set that has all elements found in both sets, that is, the set "$\{2, 3\}$".

A.2.4 Sequences

Sequences are sets of ordered elements.[63] Sequences are specified either using angle brackets ($\langle\ \rangle$) or the type keyword *seq*. For example, the definition "$\langle$a, e, i, o, u$\rangle$" specifies

[63] Specifically, sequences are sets composed of ordered pairs where the first element (a.k.a. the domain) is a natural number and the second element (a.k.a. the range) is an object of the type specified for the sequence.

a sequence of vowels, and the declaration "*x*: *seq* ℕ" indicates that *x*'s type is a sequence of natural numbers. Empty sequences are represented as ⟨ ⟩.

Some of the most common operations applied to sequences are: concatenation (⁀), which links two sequences into one, and range (represented by the keyword *ran*), which returns a set with the elements in the sequence (c.f., the indexes in the sequence). Other operations are specifically aimed to select segments in sequences, such as *head* and *last* (which select the first and last elements in a sequence, respectively), and *front* and *tail* (which selects the entire sequence but without the last and first elements, respectively). Examples of the usage of these operators are the following expressions: "⟨a, e⟩ ⁀ ⟨o, u⟩", which returns the sequence "⟨a, e, o, u⟩" (i.e., the concatenation of the two sequences); "*ran* ⟨a, e⟩", which returns the set "{a, e}" (i.e., the elements of the sequence); "*head* ⟨a, e, i⟩", which returns the element "a" (i.e., the first element in the sequence); and "*tail* ⟨a, e, i⟩", which returns the sequence "⟨e, i⟩" (i.e., the sequence without the first element).

A.2.5 Bags

Bags are sets that allow duplicated elements.[64] Bags are defined using "fat" square brackets (⟦ ⟧) or the type keyword *bag*. For example, the definition "⟦Andy, Bob, Karyn, Andy⟧" indicates a bag of names, and the declaration "*x*: bag ℝ" indicates that *x* is a bag of real numbers.

Two of the most common operations applied to bags are: domain (represented by the keyword *dom*), which returns a set with all the elements in the bag; and bag union (⊎), which combines the elements from two bags into one. The following predicates exemplify

---

An additional restriction is that all first element numbers are unique within the set, and together form a progression of consecutive numbers starting at 1 up to the number of elements found in the set.

[64] Specifically, bags are sets composed of ordered pairs where the first element (a.k.a. the domain) is an object of the type specified for the bag, and the second element (a.k.a. the range) is a natural number indicating the number of occurrences of the first element within the bag.

the usage of these operators: “*dom* ⟦Andy, Bob, Karyn, Andy⟧”, which returns the set “{Andy, Bob, Karyn}” (i.e., the domain of the bag); and “⟦Andy⟧ ⊎ ⟦Bob, Karyn⟧”, which returns the bag “⟦Andy, Bob, Karyn⟧” (i.e., the union of the two bags).

A.2.6 Schemas

Schemas are representations that specify states and their transitions. As shown in the figure below, schemas are graphically represented as a horizontally divided box where the upper half section shows the declaration of objects, and the lower half section shows the propositions that characterize the state of these objects.

$CounterSchema$

$counter : \mathbb{N}$

$counter \geq 0$

This schema is named *CounterSchema*. It defines a single object *counter* of type natural number, and a proposition indicating that this object is equal or greater than 0 (strictly speaking, this property is implicit in the definition of *counter* as a natural number; however, it is added to exemplify the usage of propositions in a schema).

By itself, this schema is not very useful, since it only declares an object. However, schemas can be included as part of other schemas to enrich their state and provide for more complex behaviour. As shown below, the schema *CounterSchema* is included as part of the state of the schema *MultiplySchema*. Note that the name of the schema *CounterSchema* in this definition is preceded by the symbol Δ. This symbol indicates that all objects in the included schema have a state before and during the execution of the schema, and a state after the execution of the schema. In this case, it indicates that there is a variable *counter* holding the value of the variable before and during the execution of the schema *MultiplySchema*, and a variable *counter'* holding the value of the variable *counter* after the execution of the schema (that is, the decorator prime (') indicates the state of a variable after the execution of a schema).

MultiplySchema

$\Delta CounterSchema$
$number1?, number2? : \mathbb{R}$
$result! : \mathbb{R}$

$result! = number1? * number2?$
$counter' = counter + 1$

The overall purpose of this schema is to multiply two numbers, and to keep count of how many multiplications it has performed. To that end, two objects are defined as the numbers that are provided for multiplication (the real numbers *number1?* and *number2?*), one object for holding the result of the multiplication (*result!*, which is also a real number), and two objects for keeping track of the number of times that this multiplication has been performed (the natural numbers *counter* and *counter'*, which were added through the definition Δ*CounterSchema*). As implied, the question mark (?) is a decorator indicating that an object is an input to a schema, and the exclamation point (!) indicates that an object is an output from the schema.

The second part of the schema defines the propositions characterizing the state of the schema. There are two propositions in the above schema: the first proposition indicates that the object *result!* equals the product of the numbers *number1?* and *number2?*, and the second proposition indicates that the number of times that a multiplication has been done is incremented by 1 (specifically, that the recorded count after this transition (given by the object *counter'*) is equal to the current count (given by *counter*) plus 1).

A.2.7 Axiomatic descriptions

Axiomatic descriptions are notations that introduce variables and their constraints. As shown below, axiomatic descriptions are divided in two halve sections, where the upper half indicates the objects being introduced (and their types), and the lower half section (which is optional) indicates the constraints of the objects declared in the upper section.

$$\begin{array}{|l}
multiply : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \\
\hline
\forall\, number1, number2 : \mathbb{R} \\
\bullet\ multiply(number1, number2) = number1 * number2
\end{array}$$

The object declared in this axiom definition is labelled *multiply*. This object is defined as a function relationship between the two real numbers and a third real number. Functions are specialized relations, since they denote the unique mapping between two objects (in this case, a pair of real numbers to another real number). As indicated in the constraints part of the axiom, the function *multiply* multiplies the numbers *number1* and *number2* and makes this multiplication the mapped value of the function (i.e., the resulting real number).

It is worth noting that one of the main differences between schemas and axiom definitions is that of state: while schemas are capable of preserving state (e.g., keeping track of how many times a multiplication has been done) axiom definitions are not.

A.3 The Object-Z Specification Language

Object-Z extends the Z specification language to incorporate object-oriented notions, such as classes, inheritance and polymorphism. The following sections describe these notions.

A.3.1 Classes

Classes are the basic building blocks in Object-Z definitions. Classes contain three types of schemas: a state schema that defines all objects making up the state of the class; an initializing schema that specifies the initial constraints of state objects; and various operation schemas (or methods) that define the state transitions available to instances of the class.

The class shown below (which is labelled *MultiplyClass*) illustrates these schemas. The first schema (which does not have a name) is the state schema; this schema defines the natural number "counter" as the only object in the class. The second schema (which is labelled *INIT*) is the initializing schema; it specifies the initial value of *counter* as equal to zero. Lastly, this class defines a single operation labelled *multiply*, which is identical in purpose to the schema *MultiplySchema* presented earlier: it takes the numbers *number1?* and *number2?* as input, specifies that the multiplication of these numbers is equal to the output object *result!*, and then increments the value of the object *counter* by one.

In addition, this class specifies a visibility list (i.e., ↾, which is shown immediately underneath the class name) indicating the methods and objects that are publicly accessible outside the class, which in this case is the method *multiply*.

$MultiplyClass$

$\upharpoonright(multiply)$

$counter : \mathbb{N}$

$\textsc{Init}$

$counter = 0$

$multiply$

$\Delta(counter)$
$number1?, number2? : \mathbb{R}$
$result! : \mathbb{R}$

$result! = number1? * number2?$
$counter' = counter + 1$

A.3.2 Inheritance

Classes can reuse and specialize the properties of other classes by using inheritance. The class *AddAndMultiplyClass* (shown below) is defined as a subclass of *MultiplyClass*, as indicated by the label underneath the visibility list.

AddAndMultiplyClass

$\upharpoonright(add, multiply, random)$

$MultiplyClass$

add

$\Delta(counter)$

$number1?, number2? : \mathbb{R}$

$result! : \mathbb{R}$

$result! = number1? + number2?$

$counter' = counter + 1$

$random \mathrel{\widehat=} add \;[]\; multiply$

This class specifies two methods, where the first is labelled *add*; this method is similar to the inherited method *multiply* except that it add rather than multiply its input numbers. The second method, which is labelled *random*, illustrates how an operation can be modelled based on other operations (in this case, *add* and *multiply*). This definition shows the use of the angelic choice operator ( ), which specifies the invocation of the method *random* results in the state transition specified by either the method *add* or the method *multiply*. In addition to angelic choice there are other operators in Object-Z, such as the conjunction operator ($\wedge$), which combines the characteristics of operations as if they were one, and the sequential composition operator ($\fatsemi$), which concatenates operations by linking the output objects of one operation to the input objects of the next.

Class inheritance can also be specified using the class union operator ($\cup$). In brief, this type of inheritance generates a new type whose identity is the union of the properties of the types being related by the operator. For example, the object definition "*tarzan*: Ape $\cup$ Man" indicates that the object *tarzan* has as its type both the types *Man* and *Ape*.

A.3.3 Polymorphism

Type polymorphism is the object-oriented characteristic that allows manipulating objects of different classes using the type of a common superclass. Object-Z defines polymorphic types using the polymorphism operator $\downarrow$. The usage of this operator is illustrated in the class *OperationsClass* (shown below), which specifies the object *operations* as a set of objects of type (or subtype) *MultiplyClass*. That is, this set can include objects of type *MultiplyClass* or any of its subclasses (e.g., *AddAndMultiplyClass*).

$OperationsClass$

$$operations : \mathbb{P} \downarrow MultiplyClass$$

$$multiplyAll \mathrel{\widehat=} \bigwedge op : operations \bullet op.multiply$$

This class also defines the operation *multiplyAll*, which shows how the conjunctive operator (and in general any of the previously shown operators) can be applied to all the elements in a set. In this case, the conjunction operator indicates that all objects in the set *operations* conjointly invoke their *multiply* operation.

A.4 Selected Z and Object-Z Notations

After showing in the previous sections the main characteristics of the Z and Object-Z, this section presents a brief glossary of the symbols from these language specifications that were used in this thesis.

Numbers

$\mathbb{N}$, $\mathbb{N}_1$	The set of natural numbers (starting at zero, at one).
$\mathbb{R}$	The set of real numbers.
$=$, $\neq$	Equality and inequality.
$<$, $\leqslant$, $>$, $\geqslant$	Numeric comparators.

Logic

$\neg$	Negation.
$\forall$	Universal qualifier.

$\exists$, $\exists_1$	Existential qualifier.
$\wedge$, $\vee$	Conjunction and disjunction.
$\Rightarrow$, $\Leftrightarrow$	Implication and bi-implication.

Sets

$\{\ldots\}$	Set definition.
$\varnothing$, $\{\}$	Empty set.
$\cup$, $\cap$	Set union and intersection.
$/$	Set difference.
$\subseteq$, $\subset$, $\not\subset$	Subset, proper subset, not a subset.
$\in$,$\notin$	Set membership and non-membership.
$\times$	Cartesian product.
$\#$	Cardinality (number of elements).
$\mathbb{P}$, $\mathbb{P}_1$	Power set (zero or more elements, at least one element).

Relations

dom, *ran*	Domain and range of a function.
$\rightarrow$, $\nrightarrow$	Function and partial function.

Sequences

$\langle\ldots\rangle$	Sequence definition.
seq, seq_1	Sequence declaration (zero or more elements, at least one element).
$\langle\,\rangle$	Empty sequence.
head, *last*	First and last element in a sequence.

front, *tail*	Sequence without the first and last elements in a sequence.
$\frown$	Concatenation.

Bags

$[\![\ldots]\!]$	Bag definition.
bag	Bag declaration.
$[\![\;]\!]$	Empty bag.
$\uplus$	Bag union.

Definitions and Declarations

==	Definition.
$\widehat{=}$	Schema definition.

Object-Z

self	Object self reference.
INIT	Initialization method name.
$\downarrow$	Polymorphism operator.
$\upharpoonright$	Visibility operator.
	Angelic choice.
$\fatsemi$	Sequential composition.
$\wedge$	Conjunction.
$\cup$	Class union.