AN ORGANIZATIONAL COMMUNICATION PROTOCOL BASED ON
SPEECH ACTS:
DESIGN, VERIFICATION AND FORMAL SPECIFICATIONS

By

TAO ZENG

B.Sc., PEKING UNIVERSITY, China, 1986

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(DEPARTMENT OF COMPUTER SCIENCE)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

July 1990
©Tao Zeng, 1990
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of  

The University of British Columbia  
Vancouver, Canada  

Date  

Aug. 2, 1990  

DE-6 (2/88)
Abstract

Current technologies are not sufficient to support the full spectrum of organizational communications because organizations are open systems and organizational communication is rather complex (e.g., involves negotiations). Speech Acts is a branch of Linguistics which views speaking to be the same as acting. Recently, Speech Acts theory has been introduced into the design of computer systems, like organizational information systems (OISs), that require complex interactions among themselves. By doing so, it is hoped that actions can be incorporated into man-machine and machine-machine communications. In this thesis, one tractable portion of the speech act theory was identified which can provide a basis for the automation of a class of semi-structured communications (e.g., simple negotiations) in a distributed organizational environment. This portion of rather abstract Linguistics theory was transformed into a concrete application layer communication protocol (namely, the SACT protocol), which was then validated using a protocol validation tool (i.e., VALIRA), specified in a standard formal specification language LOTOS, and simulated using a protocol development toolkit (i.e., the Ottawa University LOTOS Toolkit). This protocol can be used by computer-based organizational systems to automate simple negotiations, as well as recurring tasks of collecting information in an organizational environment. In addition, a communication scheme (called SACT network) was added to the Woo and Lochovsky's MOAP (Micro Organization Activity Processor) model to automate inter-micro-organizational communications using the SACT protocol. The usefulness of this scheme is demonstrated through an example application.
## Contents

Abstract \hspace{1cm} ii  
List of Figures \hspace{1cm} vi  
Acknowledgements \hspace{1cm} vii  

1 Introduction \hspace{1cm} 1  
1.1 The communication aspect of organizational information systems \hspace{1cm} 2  
1.2 Goal of the thesis \hspace{1cm} 4  
1.3 Approach used in the thesis \hspace{1cm} 5  
1.3.1 Why use the speech act theory? \hspace{1cm} 5  
1.3.2 Why design a protocol? \hspace{1cm} 6  
1.4 Outline of the thesis \hspace{1cm} 7  

2 Background \hspace{1cm} 8  
2.1 An analysis of organizational communication \hspace{1cm} 8  
2.1.1 Three schools of thought in OIS \hspace{1cm} 8  
2.1.2 Vision of the future \hspace{1cm} 11  
2.1.3 The "locator" problem in distributed organizational environments \hspace{1cm} 12  
2.2 A survey of existing systems for the automation of organizational communications \hspace{1cm} 13  
2.2.1 The Kno system \hspace{1cm} 13  
2.2.2 The Coordinator \hspace{1cm} 14  
2.2.3 CHAOS \hspace{1cm} 15  
2.3 Some related knowledge and techniques in Data Communication \hspace{1cm} 16
2.3.1 Inspiration from the DARPA Internet project 16
2.3.2 The OSI (Open Systems Interconnection) Reference Model 17
2.3.3 Formal description techniques for communication protocols 18

3 Design of the Protocol 21
3.1 A scheme for supporting organizational communications 22
3.1.1 Background: The MOAP model and agent objects 22
3.1.2 Architecture: SACT-gateways and SACT-objects 25
3.2 Overview of the SACT Protocol 30
3.3 The client protocol 39
3.3.1 Implementing services provided to Object Manager 39
3.3.2 The semantics of the client protocol 40
3.4 The server side protocol 42
3.4.1 Implementing services provided to Gateway Manager 42
3.4.2 The semantics of the server protocol 43

4 Validation, Specification and Simulation of the Protocol 45
4.1 Using VALIRA to validate the SACT protocol 47
4.2 LOTOS specification and simulation 49
4.2.1 Why LOTOS? 49
4.2.2 The Ottawa University LOTOS toolkit 51
4.2.3 Overview of the LOTOS specification 52
4.2.4 Simulation 54

5 Example Application 57
5.1 A journal editing example 57
5.2 Simulation of the journal editing example 62

6 Conclusion 64
6.1 Thesis Contribution 64
6.2 Reflection and Future Work 65

A BNF definition for the syntax of SACT language 72
B CFSM diagrams of the SACT protocol 74
C  Partial validation result

D  Major paths in the simulation of the SACT protocol
List of Figures

2.1 The OSI Model ............................................. 20
3.1 A SACT Network ........................................... 26
3.2 SACT-object and SACT-gateway .......................... 28
4.1 The top behaviour level of the LOTOS specification .... 55
4.2 The block diagram for the top level of the LOTOS specification 56
5.1 The p-sact of the journal editing application .......... 58
5.2 Part of the data structures in the workspace. .......... 59
5.3 Sequence of sentences in the case of broken up negotiation 63
B.1 Active FSM .................................................... 75
B.2 Passive FSM ................................................... 76
Acknowledgements

This thesis is dedicated to my dear grandmother. I deeply regret that I was unable to be at her side when she passed away in June 1989. Her love is the sweetest part of my childhood memories.

I am indebted to my supervisor. I'm indeed very lucky to have Dr.Carson Woo as my supervisor. He has been extremely generous with his time and energy. He always gives me the best he can. From him I have seen what it is like to be a real intellectual.

I would like to take this opportunity to thank Dr.Bob Goldenstein, my instructor in the Database course and the second reader of my thesis, for his superb lectures and clear thoughts.

It is impossible to list all the people who have given me support regarding my thesis in particular and my life in graduate school in general, but among them I especially wish to extend my gratitude to the following: Professors John McCarthy and Son Vuong for their inspiration in my research; my colleagues in the Computer Science Department: Andrew Csinger, Ann Lo, Jie Jiang, Hilde Larsen, Ola Sikskik, Sijian Zhang, Norm Goldsten; and my floormates in the residence: Robert Boissy, Jason Thorn, Chi Li and John Mahon.

Last but never least: my mom and dad. Without their support, I cannot imagine myself be able to sit in front of a SparcStation, finishing the final touches on my thesis. Their love will always be my source of strength.
Chapter 1

Introduction

This thesis is concerned with the replacement of human-to-human communications by computer-to-computer interactions, within a prescribed organizational environment. Organizations, as one component of the society (family is one of the other components of the society), have their own characteristics (e.g., they are open systems). In an organizations, communications range from casual coffee break chat to the flow of formal forms (e.g., bills). Not all of organizational communications can be replaced by automatic computers, of course. However, in order to maximize the productivity benefits from computers and computer communication systems in organizations, organizational communications should be computerized as much as possible. This is a very challenging problem, both because some organizational communications are complicated (e.g., require negotiations), and also because quite often there exist independently developed application systems which are not compatible with each other in terms of their communication methods.
CHAPTER 1. INTRODUCTION

This thesis proposes a solution to this problem. A major component of the solution is a Speech Acts based application layer protocol. Speech Acts is a branch of Linguistics theory which has recently been introduced into the design of computer systems by some OIS researchers [WiFl86]. It views speaking to be the same as acting. This thesis views the speech act theory as a theory for integrating actions into communications. In this sense, the speech act theory sheds light on the design of computer systems that require complex interactions with each other or with human users [McCa90], especially in the setting of organizational information systems. In this thesis, a useful and tractable part of Searle's Speech Acts taxonomy was identified and transformed into the format of a communication protocol for application in organizational communications. Formal techniques in the Data Communication area were used in this thesis to validate, specify and simulate the protocol.

In this introductory chapter, the important role of organizational communication and its complexity are discussed in Section 1, the goal of the thesis is discussed in Section 2, and the approach taken is introduced in Section 3, followed by an outline of the thesis in Section 4.

1.1 The communication aspect of organizational information systems

Organizational Information Systems, also called Office Information Systems and Office Automation, is the study of the application of computer technolo-
CHAPTER 1. INTRODUCTION

There are different definitions given to the concept of "organization" (or "office"), depending on different perspectives regarding the nature of organizations [FlGr88] (also see Section 2.1). For example, one definition (not necessarily the best one) is given by Carl Hewitt [Hewi86]:

"By office we mean the place where office work is done, and office work, is the information processing done to coordinate all the work that an organization does with the exception of direct manipulation of physical objects."

Organizations are inherently open systems – systems that consist of subsystems that are independently developed at disparate geographical locations. These subsystems are open-ended, undergoing continual evolution and capable of communicating with each other [Hewi85].

To cope with this highly specialized environment, organizational workers specialize in different areas of knowledge and techniques. Each one of these areas of specialization can be considered as a "micro-organization" [LoWW90]. An organization consists of many micro-organizations, which in turn may consist of other micro-organizations. In other words, "micro-organization" is a recursive concept. For example, a university is an organization that consists of various departments, one of which is the Computer Science Department. In the Computer Science Department, there are various labs, plus graduate student offices and undergraduate student offices.

Work done in the organization is mostly the collaborative effort of many of its micro-organizations. Without productive interactions among micro-
organizations, hardly anything can be achieved in an organization. Statistics show that office workers typically spend more than 70% of their working time engaged in communication tasks such as meetings and phone calls [TeGe83]. These statistics reveal the important role played by communications in organizations.

As a result of the open system nature of organizations, organizational communication is difficult to computerize. On one hand, since the components of an organizational information system are developed independently, they may use incompatible methods to represent and manipulate objects in the organizations (for example, they may use different programming languages), which makes it difficult for them to communicate with each other. On the other hand, in open systems, interactions among the system components are rather complex (e.g., cooperative and competitive negotiations, resolution of inconsistent knowledge, etc.). There is a particular need for negotiations among system components [Hewi85].

1.2 Goal of the thesis

We believe that work done in the area of data communications provide useful ideas and tools towards solving the "incompatibility" problem discussed above. In addition, work done in the area of Linguistics provide useful ideas and guidelines towards supporting negotiations with computer tools.

In this thesis, a solution was formulated for automating a class of communications, called *semi-structured organizational communications*. *Semi-structured organizational communications* occur in any of the following sit-
CHAPTER 1. INTRODUCTION

uations:

- One micro-organization intends to collect/disseminate some information from/to other micro-organization(s). However, it may not know (initially) the whereabouts of the information it wants to collect or the exact list of the micro-organization(s) to which it wants to disseminate the information.

- One micro-organization wants to negotiate some terms with some other micro-organization(s).

- One micro-organization intends to solicit commitment(s) from some other micro-organization(s).

The term “information” means data values or knowledge where knowledge is the procedures or rules through which some data values can be arrived at from some other data values. For example, a formula in a spreadsheet system is a piece of knowledge.

1.3 Approach used in the thesis

In order to reach the goal of supporting the automation of semi-structured organizational communications, a protocol based on the speech act theory [Sear69, BaBr81, Sear87] was designed.

1.3.1 Why use the speech act theory?

Why use Speech Act theory?
• Searle's Speech Act taxonomy [Sear87] provides a theory for the selection of a set of constructs (i.e., the so called “speech act verbs”\(^1\)) that is complete \(^2\) and small in number for automating semi-structured communications. The set of constructs is small in number because many speech act verbs have similar meanings and Searle groups them together according to their semantics [Sear87].

• Speech Act theory provides models of the interaction patterns of social and organizational conversations. Searle defines “conversations” as “ordered speech act sequences” that constitute arguments, discussions, buying and selling, exchanging letters, making jokes, etc. He argues that ([Sear87]):

"The key to understanding the structure of conversations is to see that each illocutionary act \(^3\) creates the possibility of a finite and usually quite limited set of appropriate illocutionary acts as replies".

1.3.2 Why design a protocol?

Why design a protocol?

• It’s a natural way to represent the interaction patterns and the conversation constructs proposed in the speech act theory.

---

\(^1\)Ballmer and Brennenstuhl define speech act verbs as verbs that we use in speaking [BaBr81].

\(^2\)The completeness of Searle’s speech act taxonomy will be discussed in Section 3.2.

\(^3\)This concept will be defined in Section 2.1.1. For now, consider it as a spoken sentence.
CHAPTER 1. INTRODUCTION

- The protocol is designed according to international standards (i.e., ISO standards) of communication protocols. It can make use of existing networking facilities. It is also suitable for standardization.  

- A large amount of effort has been put into the development of software tools which can be used in the design, validation, specification, simulation, implementation and testing of communication protocols [Liu89]. These tools can be used in the design of the protocol.

1.4 Outline of the thesis

In Chapter 2, related work in the areas of OIS, Linguistics and Data Communications are briefly described, with emphasis on those parts which have given inspiration to the work done in this thesis.

In Chapter 3, a general scheme for organizational communications using dynamic objects and gateways is described. Since the focus of this thesis is the protocol, most of Chapter 3 is devoted to the description of the design of the protocol. Chapter 4 describes the validation, the formal specification and the simulation of the protocol. Chapter 5 uses a journal editing example to demonstrate the application of our approach. Finally, Chapter 6 gives some concluding remarks, as well as a discussion of future research.

\*We believe that in the future there will emerge some technical standards for use by OIS vendors and users, just like the ISO standards which are currently widely used by the vendors and users of data communication networks.
Chapter 2

Background

In the previous chapter the important role played by organizational communications in organizational information systems (OIS) has been discussed. To provide additional background material for this thesis, some related work in areas such as OIS, Data Communication and Linguistics are introduced in this chapter.

2.1 An analysis of organizational communication

2.1.1 Three schools of thought in OIS

There are three major schools of thought (or theoretical orientations) underlying most of the organizational information systems in use today [FlGr88]. Each of them offers "distinctions" (or key concepts) from which users and
designers observe and participate in organizational activities [FlGr88] ¹.

The oldest school of thought is the "data processing" orientation, which is based on the fundamental concepts of "data" and "information". The key concepts it identifies are data, formatting, and algorithms for data storage and manipulation [FlGr88]. The fundamental assumption of this orientation is that OISs should provide comprehensive, accurate and up-to-minute data to the organizational workers, and the greater the quantity and accuracy of the information available, the more able are organizational workers to consider alternatives and make decisions. Systems developed under the guidance of this school of thought have largely failed in their attempt to improve organizational productivity and the quality of management, because whether or not the data available to managers (i.e., organizational workers) are sufficient and up-to-minute is not the real problem in organizations. An overwhelming quantity of information sometimes turns into a burden and distraction for organizational workers, as in the case of too many junk messages received by many electronic mail users.

The second oldest school of thought is the so called problem solving orientation [FlGr88], which takes "decision making" as the central task of organizational workers. This school of thought has resulted in the design of "decision support systems" and "expert systems" to assist organizational workers. Here the focus moves away from the data itself to the process of problem solving and decision making. This process can be roughly characterized as a series of steps, which include defining the problem space, listing alternatives

¹A theory, or a school of thought, can be viewed as a set of key distinctions for observing, participating and designing. It is the eyes with which we see what is going on [FlGr88].
within that space, assessing the consequences of each alternative, and finally selecting from among them. The weak point of this school of thought is the assumption that there always exists a relatively well-established and formalizable problem space. This assumption is theoretically in conflict with the open systems nature of organizations and not surprisingly is rarely appropriate in practice [FlGr88].

The newest school of thought is the "action through language" orientation, which views organizations as networks of commitments [FlGr88]. This school of thought has its root in the speech act theory [Sear69, Reis85]. Traditional language theories look at the way in which words convey information. Speech act theory looks at the ways in which speaking is connected to future possibilities and consequences [Wino88]. The basic idea of the speech act theory is that we bring about change by speaking [Sear69]. When one utters or writes sentences, he is performing some acts (so called speech acts) that have consequences for his own future action and for the actions of the people he is addressing.

The following paragraph was taken from [Woo90]:

According to Searle, when we utter a sentence, we perform the following acts [Sear69]:

1. Locutionary act – the speaker utters something.
2. Illocutionary act – the speaker does something in saying something.
3. Perlocutionary act – the speaker does something by saying something.
The prefix "ill" and "per" means "in" and "by" respectively. For example, by uttering the sentence [I will fix the problem] the speaker is performing the three acts in the following way:

1. Locutionary act – uttering the sentence "I will fix the problem".
2. Illocutionary act – committing to fix the problem.
3. Perlocutionary act – persuading the hearer that the problem will be fixed.

Note that these three acts are performed concurrently.

From the Speech Act viewpoint, human beings (and hence organizational workers) are fundamentally linguistic beings: Action happens in language in a world constituted through language [FlGr88]. Organizations are structures for the social coordination of action, generated in conversations based on requests and promises. The crucial distinctions offered by the "action through language" orientation are the fundamental linguistic actions: requests, promises, assertions, and declarations. Organizational information systems should improve the capacity of organizational workers to act by helping them in their "conversations for actions" [FlGr88].

The work done in this thesis is mainly based on this third school of thought.

2.1.2 Vision of the future

John McCarthy described the future of computerized organizational communications (taken from [McCa90]):
"... firms may issue electronic catalogs and advertisements that cause them to be listed automatically as suppliers of certain items. Programs looking for good prices and delivery conditions for the items might automatically negotiate with the programs representing the sellers and even issue purchase orders within their authority to do so."

The work done in this thesis is one step towards this rather exciting future of organizational communication, and represents an attempt to incorporate simple negotiations and commitments into computer-to-computer interactions in organizational environments.

2.1.3 The "locator" problem in distributed organizational environments

The work done in this thesis is an extension of [Woo88], in which the "locator" problem in distributed organizational environments was addressed ([Woo88]):

"Performing office problem-solving in a logically distributed environment creates one interesting new problem: how are we going to know where (and possibly when) to find a particular piece of information? Consider a manual office where information is scattered among different office workers. In order to get some particular information for problem solving, an office worker may have to spend a lot of time locating the appropriate office worker who has that piece of information [Popp82]. When computer technology was first introduced to the office, centralized data bases were
proposed to overcome this information gathering difficulty. Now that we are proposing a distributed office work model, the problem is reintroduced unless we provide tools to help office workers in locating the information they want.”

Although this thesis does not subscribe to the “problem solving” school of thought, the argument about the “locator” problem is still relevant.

One solution to this “locator” problem is the agent object [Woo88] (see also Section 3.1.1).

2.2 A survey of existing systems for the automation of organizational communications

Little work has been done in the area of automating organizational communications. To our knowledge, there are only three systems relevant to this thesis: the Kno system, the Coordinator system and the CHAOS system.

2.2.1 The Kno system

The Kno environment (currently under development at the University of Geneva) is an object-oriented environment that provides active, nomadic and adaptive objects (called Knos \(^2\)) to manipulate fragments of knowledge with their own rules [TFGN87]. A set of cooperating objects is grouped into

\(^2\)Knos stands for KNowledge acquisition, dissemination, and manipulation Objects.
a context. Communication inside a context is done by reading messages from or writing messages onto a blackboard. Communication between two different contexts is handled by an agent Kno for the remote Kno. In the Kno system, messages are themselves objects. A special message called negotiator can carry out a dialog and exchange information with other objects [TsGi87]. The dialog for negotiation is controlled by rules in the negotiator. However, these rules are not based on any theory such as the speech act theory [Woo90]. In fact, the Kno system can be categorized into the “problem solving” school of thought and hence has the corresponding limitations (as discussed in Section 2.1.1).

2.2.2 The Coordinator

The Coordinator is a commercially available communication system which is based on the “action through language” school of thought [FlGr88]. It is a kind of advanced electronic mail system in that it provides a certain structure to the contents of a message (using the speech act taxonomy) so that appropriate actions (e.g., remind the user to do certain things) can be taken by the system [FlGr88]. A message in the Coordinator has some structured parts (e.g., type of message, domain of concern, etc.) and an unstructured part (i.e., the body of the message). Different types of messages are used for different purposes (e.g., “request”, “commit” or “counter-offer”, etc.) [FlGr88]. The Coordinator also differs from traditional electronic mail systems in that the basic unit in the Coordinator is conversation, not message. It provides menus for different types of conversations (e.g., conversations for action, or
conversations for possibilities). The Coordinator system differs from the “problem-solving” school of thought in that while the “problem-solving” approach tries to simulate the user’s thinking process, the Coordinator system focuses on providing an appropriately tailored set of actions to choose from.

However, the Coordinator system requires the two (or more) parties involved in the conversation to perform all the interactions manually. It does not interpret the body (i.e., the unstructured part) of a message at all, and interacting with existing information systems is not part of its goal. For example, if some data from a database needs to be sent to another person, the user has to manually export the data from the database, and then import it to the Coordinator [Woo90]. For these reasons, it is not designed to automate machine-machine communication and hence it cannot “get the full productivity benefit out of computers” [McC90].

2.2.3 CHAOS

The CHAOS system can be considered as an extension to the Coordinator system [CiSV88]. The improvement over the Coordinator is that CHAOS uses organizational knowledge (e.g., roles and experiences of each organizational worker) to help in the support of communication. In this sense, CHAOS is based both on the “problem solving” school of thought and the “action through language” school of thought. However, just like the Coordinator, communications still cannot be carried out automatically.
2.3 Some related knowledge and techniques in Data Communication

2.3.1 Inspiration from the DARPA Internet project

The "incompatibility" problem of independently developed application systems in organizations has much in common with the problem encountered in the 70's when there were many computer communication network architectures within the DARPA project and they were unable to exchange information [Tane88]. DARPA engineers eventually got around this problem by developing network gateways, all of which implement a common protocol called the Internet Protocol.

Networks developed and used by DARPA (e.g., long haul networks, packet radio networks, local area networks, etc.) differ in geographic scope, types of services to be provided, and transmission technology. This has led to the use of a variety of specific communication protocols and interfaces [Suns90]. There are good technical and marketing reasons for these different solutions, so diversity in network technologies is likely to persist both in and out of the administrative scope of DARPA. Therefore, for a network interconnection strategy to succeed, it must accommodate the autonomy and differences of individual networks to the greatest extent possible. The DARPA internetworking strategy is to adopt an Internet architecture which has been described as ([Clar88]):

"a packet switched communications facility in which a number of distinguishable networks are connected together using packet
communications processors called gateways which implement a
store and forward packet forwarding algorithm”.

A “gateway” is a “conversion” device. When different networks and pro-
tocols are involved, interconnection involves a conversion process between
the services provided for comparable functions in each network [Suns90], and
gateways are devices to implement this “conversion process”. In DARPA
Internet, all the gateways implement a standard protocol called the IP (In-
ternet Protocol). That is, the conversion is always between IP and whatever
protocol the local network implements.

This solution of the DARPA Internet problem has been shown to be very
successful in integrating a wide range of independently developed communi-
cation networks into Internet. In fact, Internet is now operating over long
haul networks (e.g., X.25), local area networks (e.g., Ethernet, ringnet, etc.),
broadcasting satellite networks (e.g., the DARPA Experimental Wideband
Satellite Net) and packet radio networks (e.g., the DARPA packet radio net-
work), and many more [Clar88].

2.3.2 The OSI (Open Systems Interconnection) Reference Model

As a first step towards international standardization of the various protocols,
the International Standardization Organization (ISO) developed a network
model called the ISO OSI(Open Systems Interconnection) Reference Model
[Tane88]. Here the term “open systems” means “systems that are open for
communication with other systems”.
In this section, some concepts in the OSI Reference model which will be used later on in this thesis are explained. Note that only a few OSI Reference model concepts like Service Access Point and Application Layer are described.

Due to some historical reasons which are beyond the scope of this thesis, the OSI model has seven layers (see figure 2.1). The active elements in each layer are called entities. Entities in the same layer on different machines are called peer entities. The entities in layer $N$ implement a service used by layer $N + 1$. Services are available at SAPs (Service Access Points). The layer $N$ SAPs are the places where layer $N + 1$ can access the services offered.

2.3.3 Formal description techniques for communication protocols

A communication protocol is a set of rules defining how a class of distributed communicating entities should behave, step by step, in order to exchange data, voice or any other format of information. It prescribes the manner in which communication takes place, the meaning of information exchanged and the appropriateness of communication under prescribed conditions [Liu89].

The informal techniques (e.g., narrative descriptions) traditionally used to design and implement communication protocols have been largely successful, but have also yielded a disturbing number of errors or untrustworthy implementations [Liu89]. Consequently, formal description techniques (e.g., LOTOS) have emerged in the last decade. Using the formal approach, a protocol is represented by a formal model. Analytic techniques are then used to
examine logical correctness and performance of the protocol before it is actually implemented. This methodology has been proven to be very effective in identifying many protocol design errors [Liu89].

Another desirable consequence of using formal description techniques to specify protocols is that the design, validation, implementation, testing and performance analysis can be fully or partially automated.

In fact, software tools have been used throughout the process of designing, testing and revising the protocol proposed in this thesis to support organizational communication tasks. Using a formal technique has helped to clarify and concretize the ideas implied in the speech act theory.
Figure 2.1: The OSI Model
Chapter 3

Design of the Protocol

In this chapter, the design of the “SACT” (Speech Act) protocol is described. This protocol is intended to be used to support semi-structured organizational communications (see Section 1.2 for the definition of semi-structured organizational communication), although “structured” organizational communications are certainly within the range of its capabilities.

This chapter is organized in the following way: First, a new approach to support organizational communication is discussed in Section 3.1, which also gives one of the environments within which the SACT protocol is expected to work. Second, an overview of the SACT protocol is given in Section 3.2, followed by descriptions of both the client side (i.e., the SACT object) of the protocol (in Section 3.3) and the server side (i.e., the SACT gateway) of the protocol (in Section 3.4).
CHAPTER 3. DESIGN OF THE PROTOCOL

3.1 A scheme for supporting organizational communications

3.1.1 Background: The MOAP model and agent objects

Following the conceptual model (called MOAP) proposed by Woo and Lochovsky [WoLo90, LoWW90], a functional unit in an organization is considered as a micro-organization and the computer-based system supporting the micro-organization is referred to as the Micro-Organizational Activity Processor (or simply MOAP). "Micro-organization" is a recursive concept in that one micro-organization may consist of other micro-organizations, and consequently "MOAP" is also a recursive concept.

Work done in this thesis is a continuation of that described in [Woo88] and [Woo90], in which migratable, intelligent objects called Agent Objects are proposed to collect information needed for their masters (i.e., the MOAPs which initiate the Agent Objects) to perform some tasks. There is the so called "locator" problem associated with the collection of information in organizations (see Section 2.1.3): sometimes the locations of relevant information might be initially unclear. The agent object handles this problem by traveling to other MOAPs, learning from them about where to find the information, and finally bringing the information back to its master (or reporting failure if no MOAP knows the whereabouts of the specific information).

Communications between the agent object and the MOAP is done via a shared workspace, through the following process of exchanging sentences
CHAPTER 3. DESIGN OF THE PROTOCOL

([Woo90]):

The agent object puts its request in the workplace, and the MOAP will then pick up the request and try to satisfy it using the knowledge available to it. When the MOAP satisfies the request, it will put the response back into the workspace for the agent object to pick up. If the MOAP cannot satisfy the request, it can appeal to its corresponding organizational workers. Alternatively, the agent object can simply time out on waiting for a response and move on to communicate with the next MOAP.

In addition to being used as a short term memory for communication, the workspace is also an area to store the collected information or other information that is needed to perform the communication. For this reason, the agent object carries a workspace with it when it travels to other MOAPs.

In previous work [Woo90], a subset of Searle's Speech Act Taxonomy was selected and operationalized into the agent object and the result was the so called SACT tool. An application dependent program call p-sact could be written to control the agent object in its dialogue with the micro-organizational activity processors (MOAPs).

This was the first work that appeared in the literature that used Speech Act theory to develop communication tools to automate computer-to-computer interactions in a distributed organizational environment. The advantage of using Speech Act theory is that certain types of semi-structured organizational communications can be automated. For example, simple negotiations can be automated using the tool proposed in [Woo90]. The same kind of
job cannot be done by other communication tools like email systems or the Coordinator system (see Section 2.2.2).

There are significant differences between the work done in this thesis and previous research, particularly the work described in [Woo90] and [Woo88]:

- In [Woo90] and [Woo88], there were no proposals as to how the communicating process at the opposite side of the agent object (i.e., the MOAP side) would be automated, and in this sense the results in [Woo88] and [Woo90] were incomplete. In this thesis not only has the structure of the agent object been improved by using a protocol entity, but also an interacting partner of the agent object (i.e., the SACT-gateway) which contains a peer protocol entity has been designed as the front end of the MOAP to handle conversations with SACT-objects.

- In this thesis the selected subset of Searle's Speech Act taxonomy is enriched and re-organized into an application layer communication protocol (the SACT protocol). This protocol fits into the OSI Reference model and hence is suitable for implementation and standardization.

- Communication was done via the workspace in the approach proposed in [Woo88], but in this thesis this has been changed in order to be more compatible with the standard communication model – the OSI (Open Systems Interconnection) Reference model (see section 3.2).
CHAPTER 3. DESIGN OF THE PROTOCOL

3.1.2 Architecture: SACT-gateways and SACT-objects

Because the agent object is redesigned for the implementation of a Speech Act based protocol (i.e., SACT protocol), it is herein re-named as SACT-object in this thesis. In order to facilitate the automation of the MOAP side of the communication process, there was a need to design the interacting partner (or peer) of the SACT-object. This interacting partner is named SACT-gateway, since part of its function is to act as a translator between the MOAP to which it is attached and the SACT-objects which are “visiting” (communicating with) the MOAP.

As illustrated in figure 3.1, each MOAP is attached to a SACT-gateway, which is dedicated to communication with SACT-objects. These MOAPs and SACT-gateways form a conceptual network (the SACT network) in the Application Layer of the OSI model. Note that in figure 3.1 ordinary data communication networks (e.g., Ethernet) are assumed to connect the various micro-organizational activity processors (MOAPs).

Communication in the SACT Network

The following steps describe how two micro-organizational activity processors (MOAPs) in a SACT Network communicate (suppose they are MOAP-α and MOAP-β):

1. MOAP-α requests gateway-α to invoke a SACT-object by sending a message containing information needed for gateway-α to invoke a SACT-object Ω.
2. After being invoked by gateway-\(\alpha\), the SACT-object will choose the first MOAP to visit (the selection of the MOAPs to visit is based on a data structure called "reference list", which lists all the candidate MOAPs according to their priorities [KeWo88]). Suppose that MOAP-\(\beta\) is chosen. A communication channel to MOAP-\(\beta\) will be established. This can either be a local channel or a remote channel. A local channel means the two communicating entities reside in the same machine. A remote channel spans different machines.

3. The SACT-object \(\Omega\) then conducts a dialogue with gateway-\(\beta\). This dialogue is guided by the policies set by the SACT protocol and the application dependent program \(p\text{-sact}\). The result is that the SACT-gateway either provides the needed information to the SACT-object
CHAPTER 3. DESIGN OF THE PROTOCOL

Ω or else it may give some suggestions as to which other MOAPs are likely to be able to satisfy what Ω wants.

When this process terminates, Ω can return the collected knowledge and data to MOAP-α. Alternatively, it may go on to communicate with some other MOAPs using a similar process. In these steps, step 3 (i.e., the dialogue between a SACT-object and a SACT-gateway) is the focus of this thesis.

Note that because the SACT-object always initiates a dialogue and is the active partner of the communication (correspondingly, the SACT-gateway is the passive partner of the communication), we sometimes also call the SACT-object the “client” side of the SACT protocol and the SACT-gateway the “server” side of the SACT-protocol.

The SACT-object

The structures of the SACT-object and the SACT-gateway are shown in figure 3.2. Note that both the SACT-object and the SACT-gateway in the figure have a component called protocol entity which is the module implementing the SACT protocol. This pair of protocol entities are also known as peer entities in Data Communication jargon.

Figure 3.2 also shows that both the SACT-gateway and the SACT-object have access to the workspace. During conversations they can therefore refer to any data items (e.g., tables and variables) stored in the workspace. There are three ways to implement the workspace:

- If MOAP-β resides in the same computer as MOAP-α, the SACT-object’s workspace will consist of a shared memory between the SACT-
to MOAP

Figure 3.2: SACT-object and SACT-gateway

object and the gateway-\( \beta \), and no physical migration is necessary.

- If MOAP-\( \alpha \) and MOAP-\( \beta \) reside in different computers, and the cost of passing messages between them is small compared to the cost of moving a SACT-object from one site to another, then there will be no physical migration either. The shared memory (i.e., the workspace for the SACT-object) will be implemented (or emulated) by means of message passing.

- Otherwise, the SACT-object together with the workspace will migrate to the computer where MOAP-\( \beta \) resides.
CHAPTER 3. DESIGN OF THE PROTOCOL

The object manager (OM) uses the services provided by the protocol entity (PE) and executes an application dependent program (called p-sact) used by the sender(or master) MOAP to invoke the SACT-object. These services are defined by the SACT-protocol (see Section 3.2).

The SACT-gateway

Similar in structure to that of a SACT-object, the SACT-gateway also has two components: the gateway manager (GM) and the protocol entity (PE). Just like the object manager, GM also makes use of the services provided by the protocol entity. GM is the control mechanism in the SACT-gateway. It is similar to an expert system with an inference engine and a knowledge base. There are standard replies to some sentences spoken by the SACT-object, and they are governed by the rules and facts stored in the knowledge base. For example, GM can keep a list of acceptable topics, and if a SACT-object requests permission to open a topic which happens to be on that list, GM may grant permission for PE to open a dialogue on that particular topic.

But unlike the object manager, GM is not designed to make all the decisions and it has to consult its MOAP when it doesn’t know how to continue a dialogue. For example, if a topic is not on GM’s list, it will not reject it (unless it is on another list of unacceptable topics). In fact, it will inform the MOAP that a SACT-object from a particular MOAP requests to talk about a particular topic, and let the MOAP decide whether to give PE permission to open the dialogue or not.

Because MOAPs can speak a different language from the SACT-object,
the gateway manager has to do some translation when it consults the MOAP. This is similar to the translation between different protocols in a gateway in the ARPA Internet environment \(^1\). A lot of challenging problems are associated with this translation process (see Section 6.2), but to discuss them in detail is beyond the scope of this thesis.

3.2 Overview of the SACT Protocol

The SACT protocol addresses the following issues:

1. The syntax of the sentences used in the dialogue between a SACT-object and a SACT-gateway;

2. The services provided by the protocol entity in a SACT-object and the protocol entity in a SACT-gateway.

3. The rules guiding the conversation between the protocol entity in a SACT-object and the protocol entity in a SACT-gateway.

Syntax

The syntax of sentences used in the dialogue between a SACT-object and a SACT-gateway is as follows:

\[
<\text{FunctionId}><\text{Expression}>
\]

\(^1\)Note that all the SACT-objects speak the same protocol (SACT protocol). This intermediate “standard” protocol reduces the number of translators needed from \((n\times n)/2\) to \(n\), where \(n\) is the number of different MOAP languages.
“FunctionId” is the identifier of the various Speech Act functions. “Expression” will eventually include all the ordinary mathematical and logical expressions found in programming languages like Pascal or C, although at this stage only simple expressions containing single logical comparators (see Appendix A) are permitted.

Speech Act functions

The basic idea behind Speech Act theory is that speaking and acting are synonymous. Hence when we utter a sentence, we also conduct some acts (so called speech acts) which requires the listener(s) to carry out some actions and/or commit the speaker to perform some actions.

Searle classified speech act functions (or speech act verbs) into the following categories [Sear87, WiFl86]:

1. Assertives – commit the speaker to veracity (i.e., the truth of the expressed proposition). [WiFl86].
2. Commissives – commit the speaker to some future course of action.
3. Directives – attempt to get the hearer to do something. These include both questions (which can direct the hearer to make an assertive speech act in response) and commands (which attempt to get the hearer to carry out some linguistic or non-linguistic act).
4. Declaratives – bring about the correspondence between the propositional content of a speech act and reality. Here the propositional content of an utterance is “the fact that the utterance involves a proposi-
tion about some topic, such as the speaker's attendance at a particular meeting at a particular time” [WiFl86].

5. Expressives – express a psychological state about a state of affairs.

This classification is complete in that it covers all the “directions of fit” of language. Direction of fit is “the way in which propositional content is related to a world of utterance” [Sear87]. There are four and only four directions of fit in language [Sear87]:

1. The word-to-world direction of fit. In achieving success of fit the propositional content of the utterance fits the state of affairs in the world. Utterances using speech act verbs in the assertive category have this word-to-world direction of fit.

2. The world-to-word direction of fit. The world is altered to fit the propositional content of an utterance. Both commissives and directives have this direction of fit. In the case of a commissive utterance, the responsibility for bringing about success of fit rests with the speaker, while as in the case of a directive utterance the responsibility rests with the hearer.

3. The double direction of fit. In achieving success of fit the world is altered to fit the propositional content by representing the world as being so altered. This direction of fit corresponds to declarative utterances. By uttering a declarative sentence, the speaker makes the world match the propositional content simply by saying that the propositional content matches the world.
4. The null direction of fit. The success of fit is presupposed by the utterance, so there is no question of achieving success of fit between the propositional content and the world. This corresponds to expressive utterances, which simply express the speaker's attitude about the states of affairs represented by the propositional contents of the utterances.

The followings are some examples of speech act functions within each of the five categories [Reis85]:

1. Commissives - promise, offer, permit, refuse, accept, agree.
2. Directives - request, suggest, beg, invite, warn, advise, challenge to do.
3. Assertives - state, insist, conclude, deduce.
4. Declaratives - define, resign, nominate, declare war, bless.
5. Expressives - thank, congratulate, apologize, welcome.

For the purpose of supporting organizational communications, four of the five categories were chosen for use in the SACT protocol (the expressive category was not included because the present level of machine intelligence is inadequate to deal with this category of speech acts).

Searle's classification of these four categories is too general to be practical ([BaBr81]). To overcome this problem, each category was subdivided into a set of typical speech act functions to be used in a dialogue. "Typical" speech act functions represent the situations likely to arise in an organizational environment, and they differ from each other in that they don't have the same objectives. The following (typical) speech act functions are selected based
CHAPTER 3. DESIGN OF THE PROTOCOL

on common communication problems arising in organizational environments (e.g., the crucial role of commitments in organizational work, the typical procedure of negotiations in an organization). The implications (semantics) of each typical speech act functions are also briefly described.

- Typical directives are “question”, “suggest” and “request”. “Suggest” is meant to give the hearer a piece of advice, i.e., tell the hearer to do something. “Question” is used to ask the hearer for some data value or a piece of knowledge (a procedure or a set of rules) (Note that the hearer is allowed to ask counter questions – see Sections 3.3.2 and 3.4.2). “Request” is used to solicit a commitment from the hearer. In response to “request”, a so called counter offer (one of “request”, “suggest” or “question”) is also allowed (see Sections 3.3 and 3.4.).

- Typical commissives are “commit” and “decline”. “Commit” gives a promise to the hearer, in response to a previous request; “Decline” denotes the opposite of “commit”.

- Typical assertives are “assert” and “acknowledge”. “Assert” is used to tell the hearer a fact (e.g., an equation). “Acknowledge” is used to notify the hearer that the previous “assert” or “suggest” statement has been received. When a role-change (i.e., one party changes from the active role to the passive role, or vice versa) occurs, “acknowledge” is the means for the passing of ”turn to speak” (like passing a token in token ring LANs). (Note that a conversation is conducted in a strict take turn fashion.) This idea is illustrated in the Communicating Finite State Diagrams in Appendix B.
CHAPTER 3. DESIGN OF THE PROTOCOL

• Typical *declaratives* are “declareTopic” and “declareEnd”. Their semantics are self-explanatory.

The typical speech act functions listed above are considered to be adequate for the purpose of supporting semi-structured organizational communications. (Other functions like “declare war” or “resign” are not relevant to our purpose of modeling machine-to-machine communication in organizations.)

The syntax of *FunctionId* and *Expression* (in Bacus Normal Form) can be found in Appendix A.

**Rules guiding Speech Act dialogues**

A dialogue between a SACT-object and a SACT-gateway is conducted in the following way:

1. SACT-objects and SACT-gateways exchange sentences in a “strict take-turn” fashion. Each side speaks one sentence at a time.

2. A SACT-object always initiates a dialogue and is the active partner of the conversation. Correspondingly, a SACT-gateway is the passive partner of the conversation (i.e., it can never declare a new topic). In the rest of this thesis, a SACT-object may also be referred to as the “client” side of the SACT protocol and a SACT-gateway the “server” side of the SACT-protocol.

3. A set of rules is used to maintain the syntax and the semantics of the dialogues between SACT-objects and SACT-gateways. These rules guide the conversations between SACT-objects and SACT-gateways. They
are shown in a pair of communicating finite state machine diagrams for the SACT Protocol in Appendix B.

Service primitives of the SACT-protocol

On the client side (i.e., the SACT-object side), the protocol entity (PE) provides the following services to the object manager (OM):

- **Visit-request(m)** where “m” is the address of a MOAP. OM uses this primitive to inform the PE that it intends to visit the MOAP “m”.

- **Visit-confirm(m, flag)** where “m” is the address of the target MOAP, and flag is of type Boolean. The PE uses this primitive to inform the OM whether the request to visit MOAP “m” is permitted or not. If yes, “flag” is true. If no, “flag” is false.

- **Speak(s)** where “s” is a speech act sentence (see section 3.2).

- **Listen(s)** where “s” is a speech act sentence.

- **Error** – a primitive used by PE to report to OM that the sentence in a previous speak(s) is invalid.

On the server side (i.e., the SACT-gateway side), the protocol entity provides the following services to the gateway manager (GM):

- **Visit-indication(o)** where “o” identifies a SACT-object. The PE uses this primitive to inform the GM that some a SACT-object intends to visit. The identification of a SACT-object consists of the address of its master MOAP and a serial number.
• **Visit-confirm**(*o*, **flag**) where "o" again identifies a SACT-object, and **flag** is of type **Boolean**. The **GM** uses this primitive to inform the **PE** whether the SACT-object "o" is welcome. If yes, "flag" is **true**. If no, "flag" is **false**.

• **Speak**(*s*) where "s" is a speech act sentence (see section 3.2).

• **Listen**(*s*) where "s" is a speech act sentence.

• **Error** – a primitive used by **PE** to report to **GM** that the sentence in a previous **speak**(*s*) is invalid.

Note that "speak" and "listen" are the primitives used to exchange sentences. Other primitives are used to establish and terminate the connections between SACT-objects and SACT-gateways. When one protocol entity (PE) gets a sentence from a peer protocol entity, it will analyze the sentence to see whether it is valid. This is to protect the protocol entity from the harmful effects of bad (i.e., incorrect or malicious) messages, since in general a protocol entity should not fully trust its communication partners. A valid sentence is one that conforms to the syntax (section 3.2) and semantics (i.e., the rules derived from the Speech Act taxonomy) of the SACT protocol. If the sentence is valid, it will be passed to the "upper layer" (i.e., **OM** or **GM**) when the "listen(*s*)" primitive is called. Otherwise, the **PE** will send an error message to its peer, and then waits for a response from the peer entity.
Services required by the SACT protocol

In order to offer the services described above, the protocol entities in both the SACT-object side and the SACT-gateway side will use the standard services defined in the application layer and the presentation layer of the OSI model:

- The application layer services are the Associate Control Service Element (ACSE) (pp.532, [Tane88]):
  1. A-Associate: Establishes an association with the designated peer entity. Note that the application layer connections are called associations, according to the OSI documentation [Tane88].
  4. A-P-Abort: Provider initiated abort. "Provider" denotes the implementor (the lower layer protocol entity) of the service primitive.

- There are no data transfer primitives defined in ACSE, so a presentation primitive must be used. The presentation service is (pp.476, [Tane88]):
  - P-Data: data transfer in the Presentation Layer.

\[^2\]Sometimes it is more efficient and/or more convenient to bypass some higher layers, and these services may be substituted by the services in some lower (e.g., transport) layer.
3.3 The client protocol

Recall that the protocol of the SACT-object side is also called "the client protocol", and the protocol of the SACT-gateway side "the server protocol".

3.3.1 Implementing services provided to Object Manager

The client protocol provides services to Object Manager (OM) via a service access point called o-sap. These services are implemented by the protocol entity (PE) in a SACT-object. Some issues concerning the implementation of the services are in order:

- **Visit-request(m)** where "m" is the address of a MOAP. OM uses this primitive to inform PE that it intends to visit the MOAP "m".

Suppose the SACT-gateway of MOAP "m" is called G. The PE in a SACT-object (OPE hereafter) will do the following:

1. Find out whether MOAP "m" is local or remote, and establish a connection with its SACT-gateway, G, accordingly, by calling the service primitive provided by a lower layer protocol (i.e., the "A-Associate" primitive).

2. After a communication channel is set up with G, PE uses the Presentation Layer service primitive P-Data to send the identity of the SACT-object to G, and then waits for a response from G.
3. If the reply message from G indicates that the visit-request has been approved, PE will use "visit-confirm(m, 'true')" to inform OM. Otherwise, "visit-confirm(m, 'false')" will be used instead.

- **Speak(s)** where "s" is a sentence.

  When OM calls this primitive, PE will first analyze "s" to see whether it is a valid sentence. Recall that a valid sentence is one that conforms to the syntax (section 3.2) and the semantics (i.e., the rules derived from the Speech Act taxonomy) of the SACT protocol. Syntax analysis is straightforward. Semantics analysis is done via a finite state machine(see section 3.3.2). In any case, if "s" is valid, PE will send it to G, otherwise, it will be discarded and an error report will be given to OM using the primitive **error**.

- **Listen(s)** where "s" is a sentence.

  When OM calls this primitive, PE will not respond until it receives a valid sentence from G. That means PE will analyze all the sentences it gets from G, just as what it does with all the sentences it gets from OM.

### 3.3.2 The semantics of the client protocol

The semantics of the SACT protocol are the rules derived from the speech act theory. These rules are shown in a pair of communicating finite state machine (FSM) diagrams for the SACT Protocol which can be found in appendix B. The semantics of the client protocol correspond to the active finite state
machine of the SACT protocol (see figure B.1 in appendix B). The job of the Protocol Entity in a SACT-object (OPE here after) is mainly to implement this FSM. The following brief explanation helps readers understand the active FSM diagram:

- In a FSM diagram, an arc (i.e., a transition) is labeled with either “+” (indicating the reception of a message) or “-” (indicating the sending of a message). The FSM model is discussed in detail in section 4.1.

- Initially, the state of OPE is “idle”. At this state, the only valid sentence from OM (Object Manager) is one whose FunctionId is “declare-Topic”. OPE will first parse the sentence to detect potential syntactic error in the sentence. After that, it sends the sentence over to the peer entity (i.e., the protocol entity in the corresponding SACT-gateway), changes into “opening” state, and waits for the reply. If the topic gets rejected by the peer entity, OPE returns to “idle” state. Otherwise, OPE changes into “opened” state.

- At “opened” state, the content of a speech act conversation starts. From the FSM diagram, it’s clear that when a SACT-object speaks an “assert” or “suggest” sentence, it will only accept “acknowledge” and “decline” as the response. When a SACT-object speaks a “question” or “request’ sentence, some counter-questions or counter-requests may occur, respectively. And when a SACT-object speaks a “declareEnd”, the SACT-gateway may decline and causes a “role change” of the conversation to happen. That is, the SACT-gateway acts as the active side and the SACT-object acts as the passive side.
3.4 The server side protocol

3.4.1 Implementing services provided to Gateway Manager

The server protocol provides services to Gateway Manager via a service access point called g-sap. These services are implemented by the PE (Protocol Entity) in a SACT-gateway. The PE in a SACT-gateway is simply called GPE hereafter. Major issues of the implementation are:

- **Visit-indication(o)** where “o” identifies the SACT-object that is making the visit-request. After GPE gets the visit-request from the lower layer protocol (i.e., Presentation Layer), it informs the GM that some SACT-object intends to visit. One point worth mentioning is that there can be several SACT-objects making visit-requests at the same time. Hence, GPE is designed to handle concurrent visit-requests. Also, at this stage, various security mechanism can be enforced to protect the SACT-gateway from any “malicious visitor”.

- **Visit-confirm(o, flag)**. If “flag” is “true”, GPE invokes a process which is responsible to handle the conversation with the SACT-object “o”. In all cases, GPE will inform OPE (using P-Data) whether the visit-request is permitted or not.

- **Speak(s)**. When GM calls this primitive, GPE will first analyze “s” to see whether it is a valid sentence. If “s” is valid, GPE will send (again, using P-Data) it to OPE, otherwise, it will be discarded and an error
report will be given to GM using the primitive error.

- **Listen(s)** where “s” is a sentence. When GM calls this primitive, GPE will not respond until it receives a valid sentence from OPE.

### 3.4.2 The semantics of the server protocol

The semantics of the server protocol correspond to the passive finite state machine of the SACT protocol (see figure B.2 in appendix B). The following is an explanation about the main features of the passive FSM:

- Initially, the state of GPE is “idle”. In this state, the only valid sentence from the corresponding OPE is “declareTopic”. GPE will first parse the sentence to detect potential syntactic error in the sentence. Then, if the sentence is valid, GPE forwards it to GM, changes into “opening” state, and waits for the reply from GM. If GPE rejects the topic, GPE sends a “decline” to OPE and then returns to “idle” state. Otherwise, GPE sends a “declareTopic” to OPE and changes into “opened” state.

- At “opened” state, the content of the speech act conversation starts. From the FSM diagram, we can tell that in response to an “assert” or “suggest” sentence, only “acknowledge” and “decline” are valid.

In response to a “question” sentence, “assert”, “decline” and “question” are valid. If the response is “question”, the conversation evolves into the state of mutual question. In the FSM diagram, the assumption is that at most 5 levels of mutual questions are allowed. The choice of 5
is arbitrary other than that it is considered to be adequate for simple negotiations.

In response to a "request" sentence, both "commit" and "decline" will let the conversation remains in "opened" state. On the other hand, a response of "request" or "question" will lead the conversation into the state of "mutual-demand", in which the two parties negotiate by counter-requesting and counter-questioning each other. Again, the assumption is that at most 5 levels of "mutual-demands" are allowed.

In response to a "declareEnd" from a SACT-object, the SACT-gateway can decline it and thus causes a "role-change" of the conversation to happen. That is, the SACT-gateway acts as the active side of the conversation and the SACT-object the passive side. After a role-change, the SACT-gateway can use "assert" or "suggest" to initiate communications with the SACT-object. For instance, the SACT-gateway may tell the SACT-object which other MOAPs are likely to be able to satisfy its "mission" by using a "suggest" sentence, even though the SACT-object didn't explicitly ask for this information when the "declareEnd" was issued.
Chapter 4

Validation, Specification and Simulation of the Protocol

In the area of Data Communication, people have long recognized the necessity of developing distributed systems, particularly communication protocols, by way of a systematic methodology and of supporting this methodology by appropriate tools[DiVA89]. Currently, this is a very active research area and many very promising results have been achieved, some of which can be and should be applied to the research on the communication aspect of Organizational Information Systems(OIS). This is because both subjects(the communication aspect of OIS and the area of Data Communication) deal with the interaction of distributed computer systems.

This chapter describes the application of some formal techniques (e.g., LOTOS) and some automatic tools(e.g., VALIRA) commonly used by Data Communication researchers in the design, specification, validation and simu-
CHAPTER 4. VALIDATION, SPECIFICATION AND SIMULATION

Validation of the SACT protocol. (see section 2.3.3 for an introduction of formal techniques for communication protocols.) This is possible since in the previous chapter a set of useful constructs of the speech act theory have been transformed into the format of a communication protocol (i.e., the SACT protocol). It is also necessary for a number of reasons:

1. Although the SACT protocol have been described in natural language in the previous chapter, its correctness, completeness and consistency haven't been dealt with. Therefore formal techniques should be used to test the SACT protocol for syntactic and semantical correctness.

2. Formal techniques (e.g., LOTOS) are useful for designing the SACT protocol. It is believed that the "correct" protocol for any reasonably complicated, multi-component system can be arrived at with less effort through a process of formal description right from the beginning of protocol development, compared to an approach that culminates in an informal description [Liu89].

3. Formal techniques facilitate rapid prototyping. Although the idea of deriving an implementation directly from a LOTOS specification is still an open research topic currently under active investigation [CaVu89], the behavior of the step by step execution of the SACT protocol can be observed by applying a LOTOS toolkit directly on the LOTOS specification. This kind of rapid prototyping is not possible with traditional programming languages like Pascal because of the lack of abstractness and support to distributed computing.
In this chapter, protocol validation using VALIRA is described in Section 1, LOTOS specification and simulation is described in Section 2.

4.1 Using VALIRA to validate the SACT protocol

Recall that the dialogues between the SACT-gateway and SACT-object are guided by a set of rules derived from the speech act theory (see Section 3.2). In the process of deriving these rules from the speech act theory, it turned out to us that a very good way, if not the best way, to represent these rules is by means of communicating finite state machines (CFSMs).

CFSM is a transition model for the formal specification of communication protocols [VuCo86]. In this model, two or more finite state machines (FSMs) communicate with each other by exchanging messages through unidirectional, error-free channels. Each message transmission or reception in the CFSM model is represented by a FSM transition labeled with a '-' or a '+' sign, respectively, followed by the message type. With this CFSM model, a number of potential design errors of protocols can be detected, using a technique call reachability analysis which is based on the enumeration of all possible interactions between communicating processes. These four types of potential errors are [VuCo86]:

1. State ambiguities: A state ambiguity is the situation when a process\(^1\) state cannot determine the exact states of the other processes in the

---

\(^1\)The terms "process" and "CFSM" are used interchangeably here.
system and the channels are all empty (i.e., no outstanding messages). State ambiguities do not necessarily represent errors, but their semantical intents must be examined with caution.

2. *State deadlocks:* A *state deadlock* occurs when the processes cannot make any further move but to remain forever in their existing states.

3. *Unspecified receptions* (underspecification): An *unspecified reception* occurs when a message reception is possible but not specified in the CFSM. Unspecified receptions are harmful since subsequent interactions are unpredictable.

4. *Nonexecutable interactions* (overspecification): A *nonexecutable interaction* is a reception which is specified, but never occurs under normal operating conditions. Non executable interactions should be handled with care since they are redundant and may indicate the existence of other errors.

These syntactic errors are common in protocol design and yet they are often difficult to manually detect even in simple protocols, and that's why automated validation tools are very helpful.

The interaction patterns of semi-structured communications have been modelled in a pair of CFSM diagrams (see Appendix B). There is an automatic tool call VALIRA which checks a CFSM model against the above mentioned errors via reachability analysis [VuCo86]. This tool is applied on the CFSM model.

The FSMs given in appendix B passed the checking of VALIRA without
any error. An earlier version of the FSMs did have an unspecified reception which was detected by VALIRA and corrected afterwards. In short, VALIRA has been found helpful during the designing process of the SACT protocol.

The result of running SACT protocol through VALIRA is included in Appendix C, and it shows that the protocol has no state deadlock, no non-executable interaction, neither has it any unspecified reception.

4.2 LOTOS specification and simulation

4.2.1 Why LOTOS?

Although the interaction patterns of semi-structured communications have been modelled in a pair of communicating finite state machines (CFSMs), the SACT protocol hasn’t been fully specified yet. LOTOS is used for the specifications of the SACT protocol.

On the other hand, even if the SACT protocol has been specified in LOTOS, the specifications cannot be checked against those potential errors discussed in the previous section – at least there are no appropriate tools available right now to the author. Therefore, the effort described in the previous section is not redundant.

LOTOS (Language Of Temporal Ordering Specification) is one of the two Formal Description Techniques (call FDTs) developed within ISO (International Standardization Organization) for the formal specification of open distributed systems[BoBr89]. Underlying LOTOS, the theoretical basis is CCS – Calculus of Communicating Systems, and ACT ONE – an algebraic
specification language for abstract data types (ADTs).

As defined in the ISO documents [ISO87], FDTs possesses the following features:

1. **expressive**: an FDT should be able to define both the protocol specifications and the service definitions of the seven layers of OSI model described in ISO 7498.

2. **well-defined**: an FDT should have a formal mathematical model that is suitable for the analysis of these specifications and definitions. Because of the existence of the formal mathematical models, eventually protocols specified in any FDT can be implemented automatically or semi-automatically [BoLS90].

3. **well-structured**: an FDT should offer means for structuring the description of a specification or definition in a manner that is meaningful and intuitively pleasing.

4. **abstract**:

   (a) an FDT should be completely independent of methods of implementation, so that the technique itself does not provide any undue constraints on implementors;

   (b) an FDT should offer the means of abstraction from irrelevant details with respect to the context at any point in a description.

Some arguments about why a formal specification language LOTOS is chosen for specifying the SACT protocol are in order:
• First of all, formal specification generally helps the protocol designers clarify their thoughts, as it certainly is in the designing of the SACT protocol.

• Available tools: much R&D efforts have been put into the development of LOTOS tools, and some of them are accessable to the author. These tools can be used to do experiments with the prototypes of different versions of the SACT protocol without spending as much time on implementation as it could have, has C or other conventional language had been used.

• Finally, since the author advocates the standardization of organizational communication protocols in order to facilitate the integration of independently developed OIS systems, it is hoped that using LOTOS to formally specify the SACT protocol can help in this direction. In fact, protocols for organizational communications can be viewed as within the scope of the OSI Layer 7, the Application Layer. In this sense, it won't be a waste of effort trying to apply OSI standard specification language to organizational communication protocols.

4.2.2 The Ottawa University LOTOS toolkit

Methods for executing programs (specifications) written in LOTOS are classified into two types: simulators and compilers. Simulators interpret and execute LOTOS code step-by-step, and they are suitable to analyze dynamic behaviours of LOTOS specifications. Compilers translate LOTOS code into
executable programs which can be used in actual, real-time environment. However, LOTOS specifications are very high level and abstract, and it is not easy to compile these specifications into programming codes [NoHT90]. For the time being, no LOTOS compiler is accessible to the author.

The Ottawa University LOTOS toolkit [LoOb88] which basically belongs to the simulator class is used in specifying and simulating the SACT protocol. It consists of 3 subsystems:

- A translator which analyze the syntax and the static semantics of the LOTOS specification and produce a suitable prolog format of the specification for testing.

- ISLA: Interactive Simulator of LOTOS Applications, which deals with the step-by-step execution of the translated specification in order to analyze its dynamic semantics.

- GLOTOS, a graphical LOTOS interface.

4.2.3 Overview of the LOTOS specification

The SACT protocol has been specified in LOTOS, based on the following assumptions:

1. In the whole system, there is one SACT-gateway which is attached to MOAP "moap3", and there are two SACT-objects whose identifications are "moap1.sol" and "moap2.sol" respectively. This assumption simplifies the specification, yet it conserves the major features of the SACT protocol. Note that one SACT-gateway is designed to deal with
multiple SACT-objects at the same time. That's why there are two SACT-objects but only one SACT-gateway in the LOTOS specification.

2. The procedures to set up the dialogue connections between the protocol entities of SACT-objects and SACT-gateways, or the procedure to properly implement the workspaces were not specified, because these procedures are very common in the OSI protocol stack and can be copied from say, the Session layer or the Transport layer. In the specification, it is assumed that connections and workspaces have already been set up properly.

3. It is also assumed that the maximum number of nested counter offers is 5.

The specification contains about 800 lines of LOTOS code. The source for the topmost behaviour description of the specification is given in figure 4.1. A corresponding block diagram is given in figure 4.2. In the block diagram.

In this system, SACT-object "moap1.sol" uses connection Cn1 to talk to the SACT-gateway, and SACT-object "moap2.sol" uses connection Cn2 to talk to the SACT-gateway. The SACT-gateway shares the workspaces WSpl and WSp2 with SACT-object "moap1.sol" and SACT-object "maop2.sol", respectively.

In SACT-gateway, the interaction between the gateway manager(GM) and the protocol entity (GPE) is via the service access point (SAP) call "Gsap". In SACT-object "moap1.sol" and SACT-object "moap2.sol", the
interaction points between the corresponding object managers (OMs) and the corresponding protocol entities (OPEs) are “Osap1” and “Osap2”, respectively.

Note that O_C1, O_C2, G_C1, G_C2, O_WS1, O_WS2, G_WS1 and G_WS2 denote interaction points between SACT-objects, SACT-gateway and workspaces, dialogue connections.

4.2.4 Simulation

The LOTOS specification (sact.l) was first run through a LOTOS-to-Prolog translator in the Ottawa University LOTOS toolkit, resulting in a executable Quintus-Prolog file (sact.pl). This procedure discovered many syntactic and static semantical errors in the early versions of the specification of the SACT protocol. These errors were then corrected.

Next, ISLA was used to execute the protocol step-by-step, walking through all the important paths of the SACT protocol (see Appendix D). The purpose of this simulation is to discover semantic errors with the help of the toolkit.

The outcome of this simulation process was the discovery of some errors in the early versions of the specifications. These errors have all been corrected subsequently.
CHAPTER 4. VALIDATION, SPECIFICATION AND SIMULATION  55

specification SACT_Protocol : noexit
(* here are the abstract data types definitions *)

behaviour

hide 0_WS1,G_WS1,0_WS2,G_WS2,0_C1,G_C1,0_C2,G_C2 in

(* SACT-object "moapl.so1" *)
( hide Osapl in (* OM for SACT-object "moapl.so1" *)
  OM[Osapl, 0_WS1]("moapl.so1")
  |Osapl|
  OPE[Osapl, 0_WS1, 0_C1](d)(*PE for SACT-object "moapl.so1")
 )

(* SACT-object "moap2.so1" *)
( hide Osap2 in (* OM for SACT-object "moap2.so1" *)
  OM[Osap2, 0_WS2]("moap2.so1")
  |Osap2|
  OPE[Osap2, 0_WS2, 0_C2](d)(*PE for SACT-object "moap2.so1")
 )

[[0_WS1,G_WS1,0_WS2,G_WS2]]

(* workspace 1 *)

WSp1[0_WS1, G_WS1]

(* workspace 2 *)

WSp2[0_WS2, G_WS2]

[[0_C1,G_C1,0_C2,G_C2]]

(* connection 1 *)

Cn1[0_C1,G_C1]

(* connection 2 *)

Cn2[0_C2,G_C2]

(* SACT-gateway *)

hide Gsap in (* Gsap: Gateway SAP *)
  GM[Gsap, G_WS1, G_WS2]("moap3")
  |Gsap|
  GPE[Gsap, G_WS1, G_WS2, G_C1, G_C2](d)

where

(* process definitions *)

endspec (* SACT_Protocol *)

Figure 4.1: The top behaviour level of the LOTOS specification
Figure 4.2: The block diagram for the top level of the LOTOS specification
Chapter 5

Example Application

In this chapter, an example application is discussed. This example demonstrates the use of the SACT protocol, SACT-object and SACT-gateway in an organizational environment. This is an example adapted from [Woo90]. In Section 1, the background of the example application is described, along with one possible design of the p-sact program for the SACT-object used in this example. In Section 2, the simulation of this example based on the LOTOS specification of the SACT protocol is described.

5.1 A journal editing example

When an article is submitted for publication, the assigned associated editor is responsible to find a certain number (usually 3) of referees to review the paper.

It is assumed that the associated editor and all the potential referees in
the world are supported by a MOAP. Each of these MOAPs is connected to a SACT-gateway and together they form a SACT network, like the one shown in figure 3.1.

Suppose that the associate editor is not sure who are the appropriate people to be the referees for some of the papers. (For example, the associate editor does not always know who are both knowledgeable enough and willing to review these papers.)

The task of finding referees for journal papers can be considered as one example of semi-structured organizational communications defined in Section 1.2. In order to automate this communication task, the associate editor can construct a SACT-object. To do this, he has to first specify the data structures in the workspace and give the initial data values in the workspace of the SACT-object, then he has to write the p-sact program for the SACT-object.

```plaintext
negotiable("deadline", 30, "0..60");
speak "declareTopic", "review paper";
for each paper
    paper.refereeFound < 3
    speak "assert", "paper.paperNumber", ";=", paper.paperNumber;
speak "request", "evaluation";
{ moapReply = "decline";
    speak "assert", "paper.paperNumber", ";=", paper.paperNumber;
speak "question", "referee";
}
end for
speak "declareEnd";
```

Figure 5.1: The p-sact of the journal editing application

The data structures in the workspace(see figure 5.2) can be specified as
Figure 5.2: Part of the data structures in the workspace.
The following:

- The initial list of potential referees for the papers. Note that the final referees may not all be on this list – this is the result of operating the SACT-object in an open systems.

- A table call “paper”. Each paper has one entry in this table, and the fields of this table are: the paper title, the serial number, and a brief description of the paper. There is also a field call “refereeFound” which is used to record the number of referees that have already been found. Initially, this field is 0.

- Another table call “referee” used to record the results when there is someone agrees to review some papers. For each referee, there is one entry in this table corresponding to each paper he/she has agreed to review. The fields of this table are: the name of the referee, the number of the papers he/she agrees to review, and the deadline settled down during the negotiation.

The p-sact program describes:

- The topic of the speech act dialogue (i.e., “review paper”).

- The sentences that the SACT-object should speak to the SACT-gateway.

- The negotiable values. For example, the deadline for submitting the referee report can be extended from 30 days to 60 days inclusively.

---

1 For simplicity, it is assumed that a potential referee can make the decision as to whether he/she has enough knowledge to review a particular paper based on the title and a brief description of the paper content.
CHAPTER 5. EXAMPLE APPLICATION

Note that data communicated using “assert” in the p-sact are different from data in the workspace in that:

1. Some data values are private to the SACT-object (e.g., the range of negotiable values). Using “assert” statements in p-sact gives the SACT-object control as to when and what part of the private data values ought to be communicated to the SACT-gateway in the negotiation process.

2. There can be a number of data items in the workspace (e.g., entries for different papers). Which data item should be considered by the SACT-gateway at one given moment is instructed by the SACT-object using the “assert” statement.

In the journal editing example, one sample p-sact program (adapted from the p-sact program discussed in [Woo90]) is given in figure 5.1, and the following is a brief explanation: The first line specifies that the variable “deadline” can contain any value between 0 and 60 inclusively, but the ideal value (the value that should be negotiated at first) is 30. The for each statement is used to loop through all the entries in the table “paper” in the workspace. The first line inside the for each block are the conditions for identifying the paper for the current MOAP’s owner to review. This data will be communicated to the MOAP using the “speak” command. If the MOAP declined to review the paper, then the program will ask for referee suggestions. This is done in the last three lines inside the for each block.

Note that the p-sact program is not implemented in this thesis. Issues related to the design and implementation of p-sact are discussed in [Woo90].
5.2 Simulation of the journal editing example

Recall that with the Ottawa University LOTOS Toolkit, a LOTOS specification of a protocol can be executed step by step (section 4.2.2), with events (i.e., interactions happened at the gates) manually given by the user.

The LOTOS specifications of the SACT protocol can be used to simulate (i.e., execute the protocol step by step) the conversations between the SACT-object and the SACT-gateway in the journal editing example.

During the step-by-step execution of the SACT protocol, the events for both the object manager (OM) and the gateway manager were manually supplied.

For OM, the p-sact program in Figure 5.1 was exactly followed (e.g., which event should happen next, which sentence should be spoken next).

The following cases of GM’s behaviour have been simulated:

1. GM refuses to talk on the topic “review paper”.
2. GM refuses to review a paper.
3. GM agrees to review a paper, without negotiation on the deadline (i.e., it accepts the first deadline proposed by the SACT-object).
4. GM negotiates with OM on the deadline by counter offer a deadline which “happens” to be acceptable to OM, and the negotiation ended in a deal.
5. GM negotiates with OM on the deadline by counter offer a deadline which "happens" to be un-acceptable to OM, and the negotiation was broken up.

As an example, the simulation of the case of a broken up negotiation (case 5 in the above list) is described. The sequence of events happened can be represented by the sequence of sentences exchanged between OM and GM. This sequence of sentences is given in figure 5.3.

OM: declareTopic "review paper"
GM: declareTopic "review paper"
OM: assert paperNumber = "COIS90-001"
GM: acknowledge
OM: assert deadline = 30
GM: acknowledge
OM: request evaluation
GM: request deadline = 70
OM: decline deadline = 70
GM: decline evaluation
OM: question referee
GM: assert referee = smith
OM: declareEnd
GM: declareEnd

Figure 5.3: Sequence of sentences in the case of broken up negotiation
Chapter 6

Conclusion

6.1 Thesis Contribution

The contribution of this thesis is summarized below:

- The major contribution of this thesis is a protocol for the communication among independently developed application systems in organizational environment. This protocol supports simple negotiations, the purposes of which are soliciting commitments or collecting information. The power of this protocol comes from the novel application of Searle's taxonomy of the illocutionary points of human language into machine-to-machine interactions. By dealing with the illocutionary act of organizational communications, this protocol goes one step beyond all the traditional data communication protocols which have been confined in the scope of locutionary up to now. This step is an important one, especially in organizational environments, because language is not
only for storing and transferring information, but also for actions and reasoning.

- The second contribution of this thesis is the addition of a communication scheme to the MOAP model proposed in [WoLo90] and [LoWW90]. That is, a communication scheme using the SACT protocol is designed to facilitate the automation of recurring inter-MOAP communications. This scheme doesn't require the communicating MOAPs to be designed as a group and thus has the potential to cope with the continuing evolution of organizations.

- A minor contribution of this thesis is that it is a piece of pioneer work in applying formal description techniques into OIS research. The work done in this thesis shows that formal description techniques (FDTs) and associated software tools can be very useful for OIS researchers in building and validating prototype systems. And it should be safe to say that FDTs can be useful in OIS implementations, as well.

### 6.2 Reflection and Future Work

By looking back and reflecting on the work done in this thesis, two major problems have been identified, which lead to the plan for future research following the direction of this thesis:

- One of the SACT-gateway's responsibilities is to act as a translator among visiting SACT-objects and its corresponding MOAP. This can
be a very challenging problem, partly due to the possibility of inconsistent knowledge among the MOAPs. There is a promising technique called universal attachment [Myer90] which uses some AI methods to resolve the problems arising in the interpretation of many different knowledge representing languages. Future research plan is to look closer into universal attachment and try to apply it to design translation scheme for the SACT-gateway.

- Besides Searle’s taxonomy of speech act verbs, other researchers (for example, Ballmer and Brennenstuhl in [BaBr81]) also proposed different speech act taxonomies. At first glance, Ballmer and Brennenstuhl’s taxonomy seems to be more powerful than the one proposed by Searle, and in future research, these two taxonomies should be compared in detail to see if it is appropriate to use the taxonomy proposed by Ballmer and Brennenstuhl [BaBr81] in the SACT protocol.
Bibliography


BIBLIOGRAPHY


Appendix A

BNF definition for the syntax of SACT language

\[
\begin{align*}
\text{SactProgram} & ::= \text{Preamble} \mid \text{SegmentList} \\
\text{Preamble} & ::= \text{empty} \mid \text{WorkspaceDefList} \\
\text{WorkspaceDefList} & ::= \text{WorkspaceDef} \mid \text{WorkspaceDefList} ';';' \text{WorkspaceDef} \\
\text{WorkspaceDef} & ::= \text{IdentifierList} ':' \text{DataType} \\
\text{SegmentList} & ::= \text{Segment} \mid \text{SegmentList} ';';' \text{Segment} \\
\text{Segment} & ::= \text{TopicBegin} ';';' \text{Body} ';';' \text{TopicEnd} ';';' \\
\text{TopicBegin} & ::= \text{speaking 'declareTopic' identifier} \\
\text{TopicEnd} & ::= \text{speaking 'declareEnd' identifier} \\
\text{Body} & ::= \text{StatementList} \mid \text{ConditionList} ';';' \text{StatementList} \\
\text{ConditionList} & ::= \text{Condition} \mid \text{ConditionList} ';';' \text{Condition} \\
\text{Condition} & ::= \text{ComparisonExpression} \mid \text{NegotiableCondition} \\
\text{NegotiableCondition} & ::= \text{negotiable ' (' VariableRange ' )' } \\
\text{StatementList} & ::= \text{Statement} \mid \text{StatementList} ';';' \text{Statement}
\end{align*}
\]
Statement ::= ForLoop | SpokenSentence

SpokenSentence ::= speak FunctionId Expression

Expression ::= identifier ComparativeOperator identifier

FunctionId ::= 'declareTopic' | 'declareEnd' | 'question'
            | 'suggest' | 'request' | 'assert'
            | 'commit' | 'acknowledge' | 'decline'
Appendix B

CFSM diagrams of the SACT protocol
Active Finite State Machine of SACT Protocol

LEGEND:
- s1: one of "question" and "request".
- r1: one of "assert", "commit" and "decline".
- r2: one of "commit" and "decline".
- r3: one of "ack" and "decline".
- r4: one of "assert" and "decline".
Note that "ack" stands for "acknowledge"
Passive Finite State Machine of SACT Protocol

Figure B.2: Passive FSM

LEGEND:
s1: one of "question" and "request".
r1: one of "assert", "commit" and "decline".
r2: one of "commit" and "decline".
r3: one of "ack" and "decline".
r4: one of "assert" and "decline".

Note that "ack" stands for "acknowledge"
Appendix C

Partial validation result

This appendix includes part of the output of running the state diagrams of SACT protocol through VALIRA.
Hit return for the summary

ERRORSUMMARY ================

1. No state deadlock.

2. Stable state(s) :-

P1 P2 -------- 0 0 (node 0) 2 2 (node 2) 3 3 (node 5) 4 4 (node 7) 5 5 (node 10) 6 6 (node 13) 9 9 (node 16) 10 10 (node 19) 11 11 (node 22) 12 12 (node 25) 7 7 (node 28) 13 13 (node 31) 14 14 (node 34) 15 15 (node 37) 16 16 (node 40) 8 8 (node 43) 17 17 (node 46) 18 18 (node 48) 19 19 (node 50) 20 20 (node 53) 21 21 (node 56)

3. No non-executable interaction.

4. No unspecified reception.

5. The reachability tree is bounded at level 15.
6. Max channel queue length = 2.

Choose one item from the list below: 1. Print the reachability tree 2. Print the node table 3. Print the summary 4. Edit transitions 5. Run 6. Quit

***** End of execution *****
Appendix D

Major pathes in the simulation of the SACT protocol

This appendix includes some of the results of the simulation of the SACT protocol. These results were obtained by running the ISLA package of the Ottawa University LOTOS toolkit. They represent the major pathes of the LOTOS specifications of the SACT protocol.
1 [eq(s.idle)] Osap ?f:funcId [278]
   1 [eq(f.declareTopic)] Osap ?t:dataExp [281]
      1 i (specified explicitly) [282]
         1 C !declareTopic:funcId !var:dataExp [283]
         1 i C !funcId [284] => continue
   2 [ne(f.declareTopic)] i (specified explicitly) [307] DEADLOCK
2 [eq(s.opened)] Osap ?f:funcId [313]
   1 [eq(f.ask)] Osap ?e:dataExp [317]
      1 C !assert:funcId [278]
      1 [eq(s.idle, id!e)] Osap ?f:funcId [278] => continue
      1 [eq(opened, id!e)] Osap ?f:funcId [278] => continue
      2 [eq(opened, opened)] Osap ?f:funcId [313] => continue
   2 [eq(f.declareTopic)] i (specified explicitly) [330] DEADLOCK
3 [eq(f.declareEnd)] C !declareEnd:funcId ?af:funcId [334]
   1 [eq(f.declareEnd)] Osap ?e:dataExp [317]
      1 C !assert:funcId [278]
      1 [eq(s.idle, id!e)] Osap ?f:funcId [278] => continue
      1 [eq(opened, id!e)] Osap ?f:funcId [278] => continue
      2 [eq(opened, opened)] Osap ?f:funcId [313] => continue
   2 [eq(f.declareEnd)] i (specified explicitly) [330] DEADLOCK
4 [eq(f.question)] Osap ?e:dataExp [352]
   1 [eq(s.idle, id!e)] Osap ?f:funcId [278]
   1 [eq(passiveRole, id!e)] Osap ?f:funcId [278] => continue
   1 [eq(passiveRole, opened)] Osap ?f:funcId [313] => continue
   2 [eq(passiveRole, opened)] Osap ?f:funcId [313] => continue
   3 [eq(passiveRole, passiveRole)] C !funcId [422] => continue
   3 [and(ne(f,ack), ne(f, decline))] i (specified explicitly) [346] DEADLOCK
5 [eq(f.request)] Osap ?e:dataExp [378]
   1 [eq(f.request)] Osap ?e:dataExp [378]
   1 C !assert:funcId [353]
      1 [eq(s.idle, id!e)] Osap ?f:funcId [353] => continue
      2 [eq(opened, id!e)] Osap ?f:funcId [353] => continue
   2 [eq(opened, opened)] Osap ?f:funcId [353] => continue
   3 [eq(f.idle, question)] C !question:funcId [353] => continue
   4 [eq(f.request)] Osap ?e:dataExp [378]
   1 C !request:funcId [var:dataExp ?af:funcId [379]
      1 [eq(f.request)] Osap ?e:dataExp [379] => continue
      1 [eq(s.idle, id!e)] Osap ?f:funcId [379] => continue
      2 [eq(opened, id!e)] Osap ?f:funcId [379] => continue
   2 [eq(opened, opened)] Osap ?f:funcId [379] => continue
   3 [eq(f.commit)] C !commit:funcId [var:dataExp [423] => continue
   4 [eq(f.request)] Osap ?e:dataExp [400]
   1 C !request:funcId [var:dataExp [401] => continue
   5 [eq(f.request)] Osap ?e:dataExp [407]
| | | | I Osap !request:funcId !var:DataExp [408] => continue