PROTOCOL SPECIFICATION METHOD FOR CONCURRENT SYSTEM

NOPPADOL MANEERAT

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

DOCTOR OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2006

ISBN 974-15-2462-5

หัวข้อวิทยานิพนธ์	**การออกแบบโพรโตคอลสำหรับระบบที่ทำงานร่วมกัน**
นักศึกษา	**นายนพคล มณีรัตน์**
รหัสประจำตัว	42060008
ปริญญา	**วิศวกรรมศาสตรดุษฎีบัณฑิต**
สาขาวิชา	**วิศวกรรมไฟฟ้า**
พ.ศ.	2549
อาจารย์ผู้ควบคุมวิทยานิพนธ์	**รองศาสตราจารย์ ดร. รัตติกร วรากูลศิริพันธุ์**

บทคัดย่อ

วิธีการหนึ่งที่สำคัญที่สุดในการออกแบบระบบการสื่อสาร โดยเฉพาะอย่างยิ่งในการออกแบบโพรโตคอลคือ วิธีการรวมข้อตกลงของการออกแบบบริการและการออกแบบโพรโตคอล งานวิจัยนี้นำเสนอวิธีการในการรวมข้อตกลงของการออกแบบบริการ และการรวมข้อตกลงของการออกแบบโพรโตคอลไปพร้อม ๆ กันด้วยภาษา LOTOS (Language of Temporal Ordering Specification) โดยอาศัยแนวความคิด weak bisimulation สำหรับแบบจำลองอะซิงโครนัสถูกนำมาใช้จำลองระบบการสื่อสารจริง เทคนิคการรวมข้อตกลงสามารถรักษาความสมดุลระหว่างการรวมข้อตกลงของการออกแบบบริการ และการรวมข้อตกลงของการออกแบบโพรโตคอล งานวิจัยยังนำเสนอการประยุกต์ใช้งานแบบจำลองอะซิงโครนัสและระบบสนับสนุน เพื่อใช้ในการตรวจสอบการออกแบบดังกล่าว นอกจากนี้งานวิจัยนี้ได้เสนอวิธีการออกแบบข้อตกลงโดยอาศัยแบบจำลองสเตทแมชชีน ซึ่งใกล้เคียงกับการนำมาใช้งานจริง โดยการกำหนดให้ระบบการสื่อสารเป็นกลุ่มของสเตทแมชชีน ส่วนพฤติกรรมทั้งหมดของระบบการสื่อสารถูกกำหนดให้อยู่ในรูปของการโต้ตอบกันระหว่างสเตทแมชชีน และกิจกรรมภายในของแต่ละสเตทแมชชีน งานวิจัยนี้ได้แนะนำสเตทแมชชีนที่มีข้อจำกัดหลายอย่าง ซึ่งสามารถนำไปประยุกต์ใช้งานได้ตามวัตถุประสงค์ในการออกแบบที่แตกต่างกันไป นอกจากนี้ยังได้พัฒนาเครื่องมือที่ช่วยในการออกแบบและวิเคราะห์ระบบการสื่อสารโดยอาศัยสเตทไดอะแกรม ทำให้สามารถตรวจสอบความถูกต้องของพฤติกรรมในระบบที่ออกแบบก่อนที่จะมีการนำไปใช้งานจริงได้

Thesis Title	Protocol Specification Method for Concurrent System
Student	Mr.Noppadol Maneerat
Student ID	42060008
Degree	Doctor of Engineering
Programme	Electrical Engineering
Year	2006
Thesis Advisor	Associate Professor Dr. Ruttikorn Varakulsiripunth

ABSTRACT

One of the most important methods in communication system design, especially in protocol design, is a composition method of service and protocol specification. This research proposes a method for individually and simultaneously composing service and protocol specifications written in LOTOS (Language of Temporal Ordering Specification) language based on weak bisimulation concept. An asynchronous model is adopted for practical communication networks. The composition technique can maintain the equivalency between the composed service specifications and the composed protocol specifications. The application of an asynchronous model is presented and a support system of the composition technique is developed to verify the specifications. In addition, the research proposes a specification method based on a state machine model that is close to real implementation and not so much abstract. By defining a communication system as a collection of such state machines, the whole behaviors of the communication system is given in terms of interactions among the state machines and their individual internal activities. The research introduces constrained state machines that can be adapted to various specification purposes. The research has developed a design and analysis tool, which can be used to analyze the communication system using a state diagram form. By using this tool, the behaviors of a specified system can be investigated and verified before real implementation.

ACKNOWLEDGEMENTS

I would like to take this opportunity to extend my appreciation to my advisor, Assoc. Prof. Dr. Ruttikorn Varakulsiripunth who provided valuable guidance and support throughout my study. I am also indebted to Prof. Dr. Norio Shiratori, Prof. Dr. Yasushi Kato, Prof. Dr. Kaoru Takahashi, Prof. Dr. Bhed Bahadur Bista for their advises, challenging instruction and support. My greatly gratitude goes to Assoc. Prof. Dr. Paramote Wardkein and Assoc. Prof. Dr. Wipa Sangpisit for their moral support and encouragement. I would like to thank Prof. Dr. Richard Deming, Dr. Jeerasuda Koseeyaporn, Mr. Ittipoom Boonpikam, Mr. Adisak Kangsarikit and Mr. Sorrakarn Poolchareon for proof reading this thesis. Many thanks to Mr. Purich Chaochanaphun, all of my colleagues at Communication Network Laboratory, ReCCIT (Research Center for Communications and Information Technology), my colleagues at CRSC (Computer Research and Service Center), KMITL, Thailand and my colleagues at Information Engineering Laboratory, SNCT (Sendai National College of Technology), Sendai, Japan for their cheerfulness, spiritual and support.

The financial support from JICA during research in Japan, JGN-2 Project and Research Grant of Kiban, Ministry of Education Science and Culture, Japan could not be left mentioned.

Last but not least, to Mom, my brothers and sister for their unconditional love, unquestioning support and unbounded encouragement.

This thesis is dedicated to the memory of my father, Somchai Maneerat (1934-2003), whose absence has had as much impact on my life as his presence surely would have.

TABLE OF CONTENTS

PAGE

TABLE OF CONTENTS (Continue)

PAGE

TABLE OF CONTENTS (Continue)

PAGE

TABLE OF CONTENTS (Continue)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continue)

LIST OF FIGURES (Continue)

FIGURE **PAGE**

CHAPTER 1

INTRODUCTION

1.1 STATEMENT AND SIGNIFICANT OF THE PROBLEM

A protocol is a set of rules that is used to control between different communication systems in order to ensure that the interaction between them is fruitful. However, if the specification uses natural language to describe the protocol, it is difficult to strictly and correctly specify. Formal specifications and verifications of the protocol are important and necessary before the protocol can be implemented and used practically and properly. The formal specification and verification will detect errors and decrease wastes of resources that will be occurred from failure problems of a protocol specification so that this research emphasizes the protocol specification and verification methods. This research uses Formal Description Techniques (FDTs) [1, 30, 31] to describe the specification formally and unambiguously. Most of communication procedure is a concurrent system that consists of a number of communicating entities. The algorithms proposed in this research guarantee the equivalency before and after the composition of service and protocol. It is necessary to have a method for specifying concurrency and communication ability. The process calculi such as Calculus for Communicating Systems (CCS) [11] and Communicating Sequential Processes (CSP) [12] have been proposed as a specification method. The entities are modeled as processes that are represented by algebraic expressions, and dealt with the concurrency among them. There are researches, e.g. π-calculus [22] that extends the idea of the process calculus and mobility, Timed CSP [23], and E-LOTOS [24], that extend to time. On the other hand, several specification methods based on a state machine model have been proposed [13-15]. They are not so much abstract as the process calculus, but close to real implementation. There are some extensions to mobility, locality, and time. Many researches have proposed the composite of communication systems [3, 7, 27, 28]. There are some dreaded ways in which protocols may not function properly because of non-effective specification method. An approach that can specify a communication protocol correctly and unambiguously, is needed.

1.2 GOAL AND OBJECTIVE

The communication is an important part in the working systems that have protocols to be agreements between them. The agreement could hold on the different communication processes to communicate to each other. The protocol design using natural language may be ambiguous; furthermore, the faulty translation may occur. A specification language has to be based on a set of Formal Description Techniques (FDTs). LOTOS (Language Of Temporal Ordering Specification), one of the effective specification languages, is used in this research. Many techniques for specifying of computer network protocol have progressed significantly in the past decade. Many protocol specification methods use variety models to specify and verify them [4, 5, 8]. The state machine model is one of such models that are used for the protocol specification [21, 25, 27]. This research treats to the specification methods that will help to specify and verify a communication protocol to work properly. This research proposed the composition method of service and protocol specifications using asynchronous model and the specification and verification method for communication system based on state machine model with constraints that can adapt to various specification purposes.

1.3 CONTENT OF THE RESEARCH

This research proposed two methods for protocol specification. The first method is a composition method of service and protocol specifications in asynchronous model. This method will combine service specifications and protocol specifications individually and simultaneously using asynchronous model. The equivalence between the composed service and the composed protocol is maintained by weak bisimulation equivalence in LOTOS language. The other method is the specification method for concurrent system, a communication system, based on state machine model. The time and geography constraints were introduced into the model which is useful to specify and verify a communication system before a real implementation.

The content of thesis is divided into 6 chapters. Chapter 1 contains introduction and objective of the thesis. The other chapters are organized as follows.

Chapter 2 provides the communication protocol, including concept of network layered architecture, problems of communication protocol and ISO layer specification.

The Formal Description Language is given in Chapter 3 where the LOTOS, language is used to describe a communication specification protocol. The syntax and ability of LOTOS are also explained.

In Chapter 4, the composition method of service and protocol specifications (the first proposed method) is described where the service and protocol models; which are used to model communication service and protocol; are explained. In addition, a support system based on the proposed approach is generated which shall be useful for specify and verify a communication protocol.

Later the other method for specification and verification using the state machine model is given in Chapter 5. The basic state machine model is explained and, time and geography constraints of the model are introduced by a concurrent system. Furthermore, a simulator tool based on the proposed approach is produced which is available for communication system specification and verification.

Last but not least, Chapter 6 is devoted to conclusion and suggestion of thesis.

Finally, an introduction to Petri Net, a comparison between Petri Net and Finite State Machine and parts of research that were published on international journal and conferences are given in Appendix.

CHAPTER 2

COMMUNICATION PROTOCOLS

2.1 INTRODUCTION

In computing, a protocol is a convention or standard that controls or enables the connection, communication, and data transfer between two computing endpoints. In its simplest form, a protocol can be defined as the rules governing the syntax, semantics, and synchronization of communication. Protocols may be implemented by hardware, software, or a combination of the two. At the lowest level, a protocol defines the behavior of a hardware connection.

Protocols should be distinguished from technical standards, which specify how to build a computer or a similar hardware device, or specify how the contents of a file are structured, or describe the static structure of a network interface. Protocols are generally used to define real-time communications behavior, while standards are used to govern the structure of information committed to long-term storage. It is difficult to generalize about protocols because they vary so greatly in purpose and sophistication. The basic protocol functions include connection, disconnection, addressing, error control, flow control, and synchronization.

Generally, the communication system protocol is often divided into many smaller protocols that work together. Only the simplest protocols are used alone. Most protocols, especially in the context of communications or networking, are layered together into protocol stacks where the various tasks listed above are divided among different protocols in the stack. In practical, protocols are enormously complex because they involve describing the relationship functions of communication in many processes of protocols. Therefore, designing correct protocol so that they can be implemented properly is necessary in communication networking.

2.2 CONCEPT OF A LAYERED ARCHITECTURE

As a basic concept, connectionless data transmission complements the concept of connection-oriented data transfer throughout the Open System Interconnection (OSI) architecture [1, 29-31]. The OSI reference model describes how information from an application program in one computer moves through a network medium to an application program in another computer. As a basis for deriving standard OSI services and protocols, it is a conceptual model composed of

seven layers, each specifying particular network functions. The model was developed by the International Organization for Standardization (ISO) and it is now considered the primary architectural model for computer network communications. The OSI model divides the tasks involved with moving information between networked computers into seven smaller, more manageable task groups. A task or group of tasks is then assigned to each of the seven OSI layers. Each layer is reasonably self-contained so that the tasks assigned to each layer can be implemented independently. This enables the solutions offered by one layer to be updated without adversely affecting the other layers. The relationship among the OSI reference model is shown in Figure 2.1.

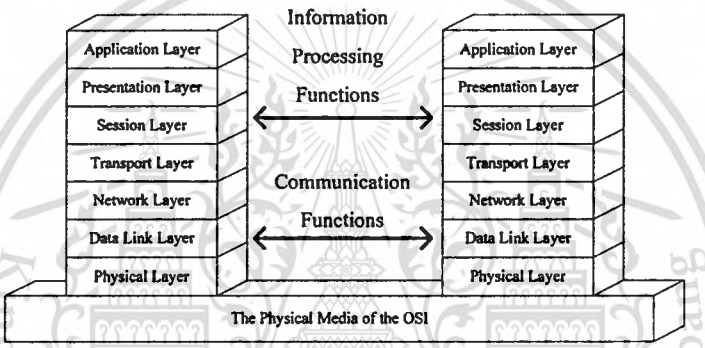


Figure 2.1 The OSI Reference Model

2.2.1 THE NEED FOR LAYERING

A protocol is more like a language that can be shared by many people. A protocol may become a standard, if all of the players in the game that would like to use that protocol all politically agree that it shall be the protocol of choice for use in, and between nations. When the protocol is ratified by the governing bodies as the shared and agreed upon system, it becomes an official standard.Often, a standard attempts to divest itself of being labeled as a protocol and tries to use language to describe how a protocol may be created to conform to the standard, as in the case of the OSI 7 Layer model.

Because of the complexity of communication system, a layered approach is adopted for the reference model. The complete communication subsystem is broken down into a number of layers each of which performs a well-defined function. ISO chooses a technique of layering to structure the OSI model. Complex communication protocols are structured in layers, and each layer is further structured in entities. An entity uses a protocol of that layer to communicate with another

entity on the same layer but in a different system. This is called peer-to-peer communication. Within a system there are one or more active entities which implement the functions of that layer (usually referred to as the (N) layer). Each entity communicates with entities above and below across and interface. The interface is called a Service Access Point (SAP). The (N-1) entity provides services to an (N) entity via the invocation of Service Primitives (SP). A primitive specifies the function to be performed and is used to pass data and control information. The layer concept of the OSI reference model is shown in Figure 2.2.

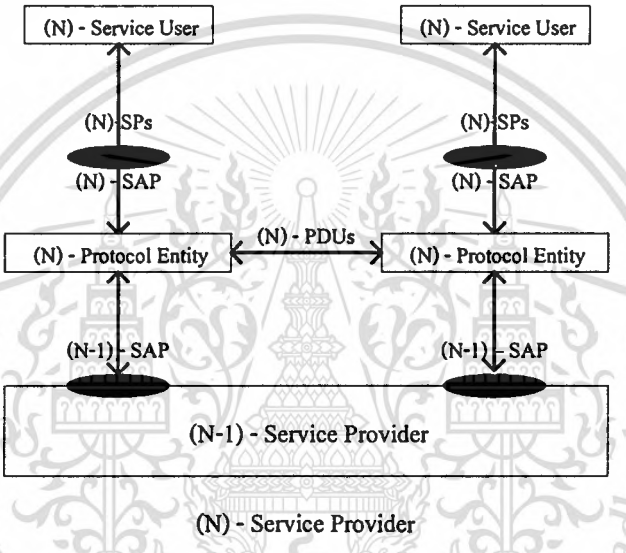


Figure 2.2 Layer concept of the OSI reference model

2.2.2 LAYERING

The main idea in OSI is that the process of communication between two end points in a communication network can be divided into layers, with each layer adding its own set of special, related functions. Each communicating user or program is at a computer equipped with these seven layers of function. So, in a given message between users, there will be a flow of data through each layer at one end down through the layers in that computer and, at the other end, when the message arrives, another flow of data up through the layers in the receiving computer and ultimately to the end user or program. An active element of a layer on a particular system is an entity. In practice, an entity may be a software module, program or process. Peer entities are entities within the same layer, but possibly on difference systems. A service is a capability

provided by one layer to the next higher layer. A Service Primitive is a simple component of a service.

2.2.3 LAYER OPERATION

OSI divides telecommunication into seven layers. The layers are in two groups. The upper four layers are used whenever a message passes from or to a user. The lower three layers (up to the network layer) are used when any message passes through the host computer. Messages intended for this computer pass to the upper layers. Messages destined for some other host are not passed up to the upper layers but are forwarded to another host. The seven layers are:

Layer 7: The application layer

The application layer is concerned with high-level functions that provide support to the application programs using the network for communication. This layer provides a means for application programs to access the system interconnection facilities to exchange information. As far as the application layer is concerned, a program running in one computer sends a message, and the program running in the other computer receives it. The application layer is not concerned with any of the details related to how the message gets from the source computer to the destination computer.

Layer 6: The presentation layer

This is a layer, usually part of an operating system, that converts incoming and outgoing data from one presentation format to another (for example, from a text stream into a popup window with the newly arrived text). Sometimes it is called the syntax layer.

Layer 5: The session layer

This layer sets up, coordinates, and terminates conversations, exchanges, and dialogs between the applications at each end. It deals with session and connection coordination.

Layer 4: The transport layer

The transport layer manages the end-to-end control (for example, determining whether all packets have arrived) and error-checking. It ensures complete data transfer. This layer may also control the rate at which messages flow through the network to prevent and control congestion.

Layer 3: The network layer

This layer handles the routing and forwarding of the data (sending it in the right direction to the right destination on outgoing transmissions and receiving incoming transmissions at the packet level).

Layer 2: The data-link layer

The data link layer is responsible for providing data transmission over a single connection from one system to another. Control mechanisms in the data link layer handle the transmission of data units, often called frames, over a physical circuit. This layer is also concerned with how bits are grouped and the beginning and ending of a "frame" of data. With some types of data links, the data link layer may also perform procedures for flow control (starting & stopping data), frame sequencing, and recovery from transmission errors.

Layer 1: The physical layer

This layer conveys the bit stream through the network at the electrical and mechanical level. It provides the hardware means of sending and receiving data on a carrier. This layer addresses the cables, connectors, modems, and other devices that used to permit machines to physically communicate and controls the generation and detection of signals that are interpreted as 0 bits and 1 bits.

2.3 PROBLEMS OF COMMUNICATION PROTOCOLS

The computer communication technology now is very complex. Many manufacturers of data processing equipment have developed their own architecture and protocol. It is complicated for equipments from different manufacturers to communicate together. It brings to complexity of

computer communications. Many protocols work properly, but almost of them suffer from unexpected failures. When the correct condition in the protocols is changed, there probably are faulty behaviors in protocols such as waiting for signal from the other party. No signal can be sent and the system is deadlock. Another problem is looping that occurs when the parties become locked. Where timeout and retransmission of messages occur, it is possible for two or more copies of the same packet or message to be accepted by the receiver as separate messages. It is possible that the loss of messages can be occurred.

2.4 ISO LAYER SPECIFICATION

Each layer of the OSI Model consists of service and protocol entities. The service in each layer is provided by means of a protocol. This section will describe service and protocol specifications in a layer perspective.

2.4.1 SERVICE SPECIFICATION

Each layer provides a set of services to its users from above. The layer can be seen as a black box which allows a certain set of interactions with other users. This description of the input/output behavior of the protocol layer constitutes the service specification of a protocol.

Usually, a service specification is based on a set of service primitives which describe the operations at the interface through which the service is provided. The execution of a service primitive is associated with the exchange of parameter values between the entities involved.

Service primitives should not be executed in an arbitrary order and with arbitrary parameter values. The service specification must reflect these constraints by defining the allowed sequences of operations directly, or by making use of a state of the service which may be changed as a result of some operations.

2.4.2 PROTOCOL SPECIFICATION

A protocol consists of a number of messages which are passed between respective peers and establishes the rules for dialogue exchanges when a communication requirement between any two end users is initiated. Any messages, which are being directed to their respective peer entities, are then passed between end systems using the connected physical medium and action according to the relevant procedures and semantics of each protocol concerned. Entities (processes or modules) local to each user communicate with one another via the services of the lower layer.

The interaction among entities in providing services of the layers constitutes the actual protocol. Hence a protocol specification must describe the operation of each entity within a layer in response to commands from its users, messages from the other entities (via the lower layer service, also called Protocol Data Unit (PDU)), exchange of interactions (via service primitives) with service users at service access points, and internally initiated actions (such as timeouts).

Thus, a protocol specification typically includes: a list of the types and formats of messages exchanged between the entities; and the rules governing the reaction of each entity to user commands, messages from other entities, and internal events.

2.4.3 THE PROCEDURE FOR PROTOCOL SPECIFICATION

Figure 2.3 depicts the procedure for the specification of communication protocols at the layer N. The procedure begins with the (N)-service specification which describes what services the (N)-protocol entities provide for their service users. In the service specification, the service access points (SAPs), service primitives observed at the SAPs and their temporal ordering are defined. The abstract service facilities may also be defined in the service specification for the ease understanding some service behaviors when necessary. These service facility definitions are of great help for later protocol specification. The service specification is then refined to the (N)-protocol specification.

The (N)-protocol specification narrates a set of protocol entities which may be obtained by refining or splitting the service facilities. These protocol entities communicate with their peers through the (N-1) services to provide services to the service user at the upper SAPs. These services provided by the (N)-protocol must be the same as those defined previously in the (N)-service specification. In other words, the (N)-protocol specification implements the (N)-service specification. The procedure is repeated on the (N-1)-service specification to get the (N-1)-protocol specification and so on until the lowest layer.

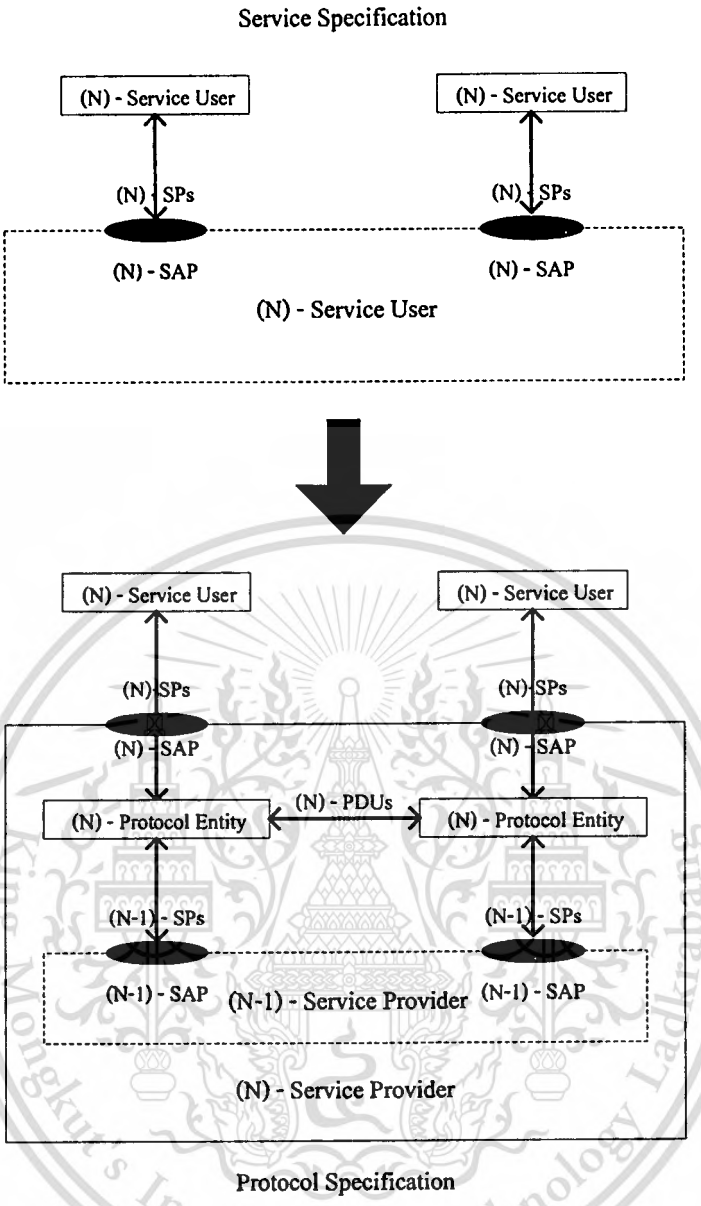


Figure 2.3 A procedure for protocol specification

CHAPTER 3

FORMAL DESCRIPTION LANGUAGE

3.1 INTRODUCTION

In protocol design, an effective specification language has to be based on a set of general purpose language elements. A special purpose language has to be invented for the design and to be described behaviors and events of services in protocols. Specifications of communication protocols were written in natural languages that can be unclear and ambiguous so formal description languages have to be invented. The protocol specifications which are expressed in a formal description language can be analyzed, verified, and interpreted unambiguously. The formal description languages were developed to meet OSI requirements. There are various formal description techniques (FDTs) that have been developed for specifying communication protocols. The main techniques are Finite State Machine Model, Extended Finite State Machine Model, Process Algebra, and Temporal Logic. As a result of development, two new FDTs – Estelle (ISO 9074, 1989) which is based on an Extended Finite State Machine Model, and LOTOS (Language of Temporal Ordering Specification, (ISO 8807, 1989)) which is based on Milner's Calculus of Communicating Systems were formed and standardized by ISO. The SDL (Specification and Description Language, ITU z.100, 1987) was developed by ITU (International Telecommunication Union).

In this thesis, LOTOS and Finite State Machine are used to describe protocol specifications. They are expressed behaviors, functions, and service of protocol. This chapter is the LOTOS introduction. As the Finite State Machine will be introduced and applied in chapter 5.

3.2 LOTOS

The basic idea of LOTOS is that systems can be specified by defining the temporal relations among the interactions that constitute the externally observable behavior of a system. LOTOS is made up of two components: (i) a data type component, which is based on the algebraic specification language ACT ONE (Ehrig and Mahr, 1985), and (ii) a behavior component, which is based on process algebra [30]. This process algebra, Milner's Calculus of Communicating Systems (CCS) (Milner, 1980), includes the concepts of parallel processes that communicate through a gather mechanism, allowing two or more processes are specified together.

In LOTOS, a system can be seen as a set of processes which interact and exchange data with each other and their environment. Every process communicates through a set of named gates. At such gates, communication is observable in terms of atomic synchronous interactions. These interactions associated with gates can also pass parameters to other processes participating in the interactions. Whereas the values exchanged by the processes are represented by value expressions, the communication behavior of the processes is represented by behavior expressions. In this thesis, it focuses on basic LOTOS which is dealt with this research but data type component of LOTOS will not be explained.

3.3 PROCESSES

In LOTOS, a system is modeled as a collection of processes interacting with each other. A process possibly consists of several sub-processes, each of which is a process in itself. A process in LOTOS is described in terms of its observable behavior, which can be represented by a sequence of all observable (inter)actions up to a distinct point of time as shown in Figure 3.1.

where A, B are processes

C is an observed action at A

B is the behavior of A after observing C

Figure 3.1 A transition of process A and B

Figure 3.2 shows two processes: P1 and P2, each of which can be modeled as a black box capable of communication with its environment. The mechanisms inside these boxes are not observable therefore, in principle, not part of the model. Thus, a process can be described by the specification of its interactions. The atomic form of interaction is an event, which is a unit of synchronized communication that may exist between two processes P1 and P2 can be defined from the sequences of observable events that may occur at the gates a, b and c of the process P1, and at the gate c and d of the process P2.

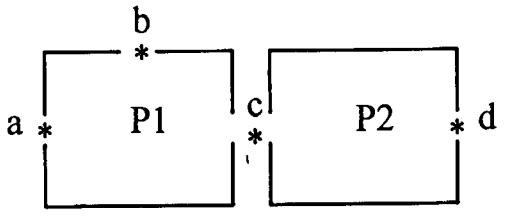


Figure 3.2 Two interacting process: P1 with gates a, b, c and P2 with gates c, d

3.4 BEHAVIOR EXPRESSION IN BASIC LOTOS

Basic LOTOS is a simplified version of the language where process synchronization is achieved, but without data exchange. In other words, no data types in basic LOTOS are used. An action in basic LOTOS will be identified as a gate.

A behavior expression is an essential component of a process definition, and can be formed by applying the defined behavior operators. A behavior expression may also include instantiations of other processes. The complete list of basic LOTOS behavior expressions is given in Table 3.1, including all basic LOTOS operators.

Table 3.1 A list of basic LOTOS behavior expressions

Name	Syntax
Inaction	**stop**
Action prefix (Internal)	**i**; *P*
Action prefix (Observable)	*a*; *P*
Choice	*P*1 [] *P*2
Parallel composition (Selective)	*P*1 $\|[g_1,\ldots, g_n]\|$ *P*2
Parallel composition (Pure interleaving)	*P*1 ||| *P*2
Parallel composition (Full synchronization)	*P*1 || *P*2
Hiding	**hide** $g_1,\ldots, g_n$ **in** *P*
Process instantiation	$P[g_1,\ldots, g_n]$
Successful termination	**exit**
Sequential composition (Enabling)	*P*1 >> *P*2
Disabling	*P*1 [> *P*2

3.4.1 PROCESS TERMINATION

In basic LOTOS, **exit** and **stop** are used to terminate a process. The **exit** operator represents the successful termination of a process. The **exit** denotes a process which performs the successful termination action (gate) δ and becomes **stop**. However, the action δ cannot be used directly in a specification but only via the **exit** operator.

The **stop** operator indicates the unsuccessful termination of a process. It represents the completely inactive process. It cannot offer anything to the environment, and cannot perform any internal actions. A process is in deadlock if at a given moment in its execution, all alternatives for future events are equivalent to **stop**.

3.4.2 TWO BASIC OPERATORS

3.4.2.1 ACTION PREFIX

The action prefix operator (;) is used to produce a new behavior expression out of an existing one by prefixing it with an action (gate) name. It expresses sequential composition of events. Example of action prefix is:

a; b; **stop**

Given a and b are event names in a process expression, a will be executed then b can be executed and process is terminated by **stop** operator.

3.4.2.2 CHOICE

If *P*1 and *P*2 are behavior expressions, *P*1 [] *P*2 denotes a process with the alternative composition of processes *P*1 and *P*2 where it is ready to behave as *P*1 or as *P*2. The choice operator is commutative and associative. As a consequence, *P*1 [] *P*2 is equivalent to *P*1 [] *P*2, and *P*1 [] *P*2 [] *P*3 is equivalent to both (*P*1 [] *P*2) [] *P*3 and *P*1 [] (*P*2 [] *P*3).

3.4.3 PROCESSES AS TREES

The behavior of a LOTOS process can be represented as a tree-like structure. The initial state of the process is represented by the root of the tree, and the edges of the tree are labeled by event names. As shown in Figure 3.3, two processes, *P*1 and *P*2, have the same gates, but different behavior; consequently they have different tree structures.

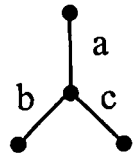
Here, our technique is mainly concerned with a labeled transition system (LTS) to describe the formal semantics of a process. *Act* is defined as a set of actions, *Act** as a set of sequences of

actions in *Act* and **i*** as zero or more **i** (internal) actions. *Act*(*P*) is defined as the set of actions which process *P* can execute.

Process *P*1 [a, b, c] :=

a; (b; **stop** [] c; **stop**)

endproc



Process *P*2 [a, b, c] :=

a; b; **stop** [] a; c; **stop**

endproc

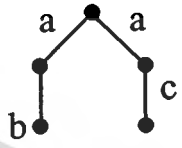


Figure 3.3 Two different processes with their behavior represented in tree structure

[Definition 1] A labeled transition system (LTS) L is a quadruple $< S, A, T, s_0 >$, where S is a nonempty set of states, A is a subset of *Act*, $T \subseteq S \times A \times S$ is a transition relation and $s_0 \in S$ is the initial state of L.

In this definition, if $t = a_1 \ldots a_n \in (Act - \{\mathbf{i}\})^*$, then $s \Rightarrow^{t} s'$ stands for

$$s \xrightarrow{\mathbf{i}^*} \xrightarrow{a_1} \xrightarrow{\mathbf{i}^*} \xrightarrow{a_2} \xrightarrow{\mathbf{i}^*} \ldots \xrightarrow{\mathbf{i}^*} \xrightarrow{a_n} \xrightarrow{\mathbf{i}^*} s'$$

Here, the notation $s_1 \xrightarrow{a} s_2$ represents the transition from state s_1 to state s_2 by the execution of the action a.

3.4.4 RECURSION

The use of the recursive occurrence of process identifiers in behavior expressions allows us to define infinite behavior. The recursion can be mutual when process abstractions refer to the names of other process abstractions. As illustrated in the example below, the behavior of the process *P*1 is recursive by referring to itself, while the recursion of the processes *P*2 and *P*3 is mutual.

process *P*1[a, b] := a; b; *P*1[a, b] endproc

process *P*2[a, b] := a; *P*3[a, b] endproc

process *P*3[a, b] := b; *P*2[a, b] endproc

3.4.5 NONDETERMINISM AND INTERNAL ACTION

If there exists more than one alternative in a process that starts with an event which is also enabled by the environment then the choice is nondeterministically resolved between such alternatives. A simple example of nondeterminism is represented by the following expression:

a; b; **stop** [] a; d; **stop**

where the result of observing a is not determined. The unobservable (internal) action is also not determined, as shown by the expressions:

i; b; **stop** [] **i**; c; **stop**

3.4.6 SEQUENTIAL COMPOSITION

The idea of sequential composition operator is that the second process is enabled only if when the first process terminates successfully. There are two ways to express the sequential composition. The first is composing two processes in parallel by synchronization of the last action of the first process with the first action of the latter. The other is having a separate operator for the composition which is a direct way. The LOTOS enabling operator (>>) is a direct way to express sequential composition of processes.

For example, if there are two behavior expressions, then the first expression must terminate successfully in order for the second one can be enabled. Execution of an exit in the first expression results in an action on a special gate δ. The enabling operator causes δ to become an **i** (internal action), and the execution continues with the last expression.

p; q; **exit** >> r; **stop**

is equivalent to

p; q; **i**; r; **stop** or p; q; r; **stop**,

whereas, p; q; **stop** >> r; **stop**

is equivalent to

p; q; **stop**

The expression on the right-handed side of the enabling operator cannot be executed because the expression on the left-handed side cannot terminate successfully.

3.4.7 PARALLELISM

LOTOS provides various facilities to represent concurrent processes. Three parallel composition operators in LOTOS are selective parallel, full synchronization and pure interleaving.

All of the parallel composition operators are described in this section.

3.4.7.1 SELECTIVE PARALLEL

The selective parallel operator, |[]|, is used when two or more processes need to synchronize on an observable action or more. Given $P1$ and $P2$ are processes. If g_1 ,..., g_n are some common observable actions of $P1$ and $P2$, then $P1$ $|[g_1,..., g_n]|$ $P2$ represents the parallel execution of $P1$ and $P2$ that provides the synchronization on the list of actions $g_1,..., g_n$. For example:

p; q; r; **exit** |[p]| s; p; r; **exit**

is equivalent to

s; p; (q; r; r; **exit** [] r; q; r; **exit**)

As illustrated in the above example, both processes execute independently until one of them reaches a common action, at which point it must wait to synchronize with the other. In this example, p is an action of both processes. Thus, process 1 must wait for process 2 to reach action p before offering action q. Consequently after synchronization on p, both processes continue independently with all of the remaining actions.

3.4.7.2 FULL SYNCHRONIZATION

The full synchronization, ||, is used when two or more processes want to synchronize all observable actions. The composition $P1$ || $P2$ represents the parallel execution with completed synchronization on all observable actions. Clearly when synchronized action is the list of all actions, the selective parallel operator becomes the full synchronization operator. For example,

a; b; c; **exit** || a; b; c; **exit**

is equivalent to

a; b; c; **exit**

Another example,

p; q; **exit** || s; p; r; **exit**

is equivalent to

stop

3.4.7.3 PURE INTERLEAVING

The Pure interleaving, |||, is used to express two or more processes that are composed together without synchronization. The composition $P1$ ||| $P2$ represents the parallel execution without any synchronization. In other words, processes are composed in parallel independently. For example,

a; b; c; **exit** ||| c; d; **exit**

is equivalent to

a; (b; c; d; **exit** [] c; (b; d; **exit** [] d; b; **exit**))

[]

c; (d; a; b; **exit** [] a; (d; b; **exit** [] b; d; **exit**))

3.4.8 DISRUPTION

The disabling operator ([>) represents an interruption of a process by another process. Given two processes $P1$ and $P2$, the expression $P1$ [> $P2$ means that, at any point during the execution of $P1$, an initial action of $P2$ can occur. If such an action occurs, control is transferred to $P2$, and the remaining action of $P1$ cannot be executed, only $P2$ will be executed. If $P1$ terminates unsuccessfully, controlling is transferred to $P2$, while if $P1$ terminates successfully, $P2$ does not start execution. For example,

p; q; **exit** [> r; t; **stop**

is equivalent to

p; (q; (**exit** [] r; t; **stop**) [] r; t; **stop**) [] r; t; **stop**

3.4.9 HIDING

The hiding operator allows some observable actions of a process to transform into unobservable actions. In particular, the protocol designer can compose a system using LOTOS operators while hiding the details of interprocess communications that are not relevant at a higher level of abstraction. The hiding operator is denoted by **hide** G **in** P, where G is a set of actions, and P is a composite process or subprocesses in parallel. For example,

(**hide** q **in** p; q; r; **exit**) || p; r; **exit**

is equivalent to

p; **i**; r; **exit**

because q is become an internal action.

Next chapter, the basic LOTOS behavior expressions are applied to a composition method of service and protocol specifications. The concept of process is used to describe behaviors in a communication system. Furthermore, a weak bisimulation equivalence is introduced and used to maintain equivalence between composed service specifications and composed protocol specifications.

CHAPTER 4

COMPOSITION OF SERVICE AND PROTOCOL SPECIFICATIONS IN ASYNCHRONOUS COMMUNICATION SYSTEM

4.1 INTRODUCTION

Communication system consists of two elements, protocol and service. Protocol is a set of processes (entities) communicating with each other under defined rules to accomplish a common task. Service is a set of requested tasks that are processed by protocol. Therefore, the proper design of service and protocol specifications becomes important. A service is specified based on the temporal ordering of actions that may occur at different Service Access Points (*SAP*s). A protocol is specified so that each communicating process provides service at its *SAP* by exchanging messages with each other.

In a research related to communication system design, the technique to derive protocol specifications from service specifications was presented and the technique of partial functional protocol specification was discussed [2, 4-7]. Bista et al. proposed a compositional approach for constructing communication services and protocols simultaneously [3]. They considered service and protocol as parallel elements. The advantage of this approach is that it is easy to design and verify the specifications because the composed protocols were defined in terms of sub-functions. They used Langerak's algorithm to decompose service specifications and to construct the equivalent protocol specifications [2]. Four major operators in LOTOS, i.e., enabling, choice, parallel, and disabling, were used in the composition method. They applied this technique to a synchronous communication model to show its availability.

In this research, the composition technique of Bista et al. has modified and extended in order to apply it to the *asynchronous* communication model. The weak bisimulation concept is used to combine service specifications and protocol specifications individually and simultaneously while maintaining the equivalence between composed service and composed protocol specifications. Finally, a support system is developed.

4.2 DEFINITION OF TRANSITION AND WEAK BISIMULATION EQUIVALENCE

A technique using the labeled transition system will be used to describe the formal semantics of a process. *Act* is defined as a set of actions, *Act** as a set of sequences of actions in *Act* and **i*** as zero or more **i** (internal) actions. *Act*(*P*) is defined as the set of actions which process *P* can be executed.

[Definition 1] A labeled transition system (LTS) L is a quadruple $< S, A, T, s_0 >$, where S is a nonempty set of states, A is a subset of *Act*, $T \subseteq S \times A \times S$ is a transition relation and $s_0 \in S$ is the initial state of L.

In this definition, if $t = a_1 \ldots a_n \in (Act - \{i\})^*$, then $s \Longrightarrow^{t} s'$ stands for

$$s \xrightarrow{i^*} \xrightarrow{a1} \xrightarrow{i^*} \xrightarrow{a2} \xrightarrow{i^*} \ldots \xrightarrow{i^*} \xrightarrow{an} \xrightarrow{i^*} s'$$

Here, the notation $s_1 \xrightarrow{a_n} s_2$ represents the transition from state s_1 to state s_2 by the execution of the action a_n.

Given an expression B, x an action and B' an another expression, B may perform action x and transform into B'. The labeled transition is as follow:

$$B \xrightarrow{x} B'$$

For example, expression is *a*; *b*; *c*; **stop**. The transition will be derived as follow:

$$a; b; c; \textbf{stop} \xrightarrow{a} b; c; \textbf{stop}$$

The standard equivalence, used in this research, is an observational equivalence which is called "behavioral equivalence" or "weak bisimulation equivalence". The equivalence can describe a system at various levels of abstraction. It can describe structure of a system in term of subcomponents and behaviors of a system from an external observer's point of view. The idea of the weak bisimulation equivalence is that two systems are considered as equivalent whenever an external observer cannot see different action sequence of system from an observer's view or environment. Using the operational semantics given in the above definitions, a bisimulation relation is defined:

[Definition 2] Let $L_1 = <S_1, A_1, T_1, s_{10}>$ and $L_2 = < S_2, A_2, T_2, s_{20} >$ be labeled transition systems. A binary relation $R \subseteq S_1 \times S_2$ is a weak bisimulation relation if $(s_1, s_2) \in R$ implies that, for all $t \in (Act - \{i\})^*$,

1. if $s_1 \Rightarrow^t s_1'$ for some s_1', then $s_2 \Rightarrow^t s_2'$ and $(s_1', s_2') \in R$ for some s_2'
2. if $s_2 \Rightarrow^t s_2'$ for some s_2', then $s_1 \Rightarrow^t s_1'$ and $(s_1', s_2') \in R$ for some s_1'.

L_1 is weakly bisimilar to L_2 (or L_1 is weakly bisimulation equivalent to L_2) written as $L_1 \approx L_2$, if $(s_{10}, s_{20}) \in R$ for some weak bisimulation R. A process P is weakly bisimilar to a process Q (or P is weakly bisimulation equivalent to Q), written as $P \approx Q$, if the labeled transition systems of P and Q are weakly bisimulation equivalent. More details of weak bisimulation equivalence concept and proof are described by Milner [11].

For example, given expression P is *a*; (*b*; **stop** [] **i**; *c*; **stop**) [] *a*; *c*; **stop** and expression Q is *a*; (*b*; **stop** [] **i**; *c*; **stop**). The expression P and Q can be expressed in tree structures. From external observer's view, both P and Q have same observable action sequence. P is weakly bisimulation equivalent to Q and can be written as *a*; (*b*; **stop** [] **i**; *c*; **stop**) [] *a*; *c*; **stop** $\approx$ *a*; (*b*; **stop** [] **i**; *c*; **stop**). The weak bisimulation is shown in Figure 4.1 which the pairs of nodes are connected by the dash lines.

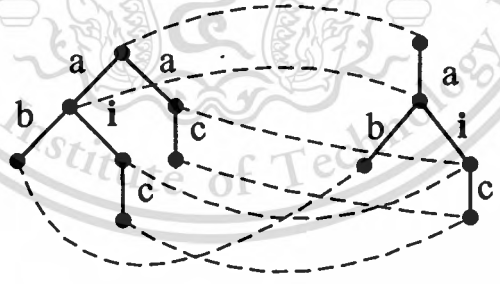


Figure 4.1 A bisimulation

A typical example of two different descriptive levels found in the OSI architecture is provided by the concepts of protocol and service [24, 39]. The (N)-protocol specification describes a set of protocol entities which may be obtained by refining or splitting the service facilities. These protocol entities communicate with their peers through the (N-1) services to provide services to the service user at the upper SAPs. These services provided by the (N)-

protocol must be the same as those defined previously in the (N)-service specification. In other words, the (N)-protocol specification implements the (N)-service specification. The procedure is repeated on the (N-1)-service specification to get the (N-1)-protocol specification and so on until the lowest layer. The specification of the N-service is implemented by the composition of the N-protocol entities with the (N-1)-service, and it seems natural to require that the two descriptions be equivalent.

4.3 SERVICE AND PROTOCOL MODELS

In this section, first the models of service and protocol in asynchronous communication system are described. Next, a polling mechanism is introduced to preserve the order of actions and used to let the behaviors of service and protocol be equivalent. Finally, the decomposition algorithm that is used to derive the equivalent protocol specification from a service specification automatically is summarized.

4.3.1 SERVICE MODEL

A service is modeled as a black box shown in Figure 4.2 (a). Here, user knows only what is provided at each *SAP* but user does not know how it is provided. The black box represents all of the lower layers including local node and, network and remote node. A service is described by the temporal ordering of actions that occur at *SAP*s. Assume that there are two *SAP*s, where *SAP*1 is at node 1 and *SAP*2 is at node 2. The service specification is in an *action-prefix form*, e.g., $S = \sum\{a_i; A_i \mid i \in I\}$ for some finite index set I where each A_i is either a process identifier or an expression in an action-prefix form. The behavior of a specification is represented by an LTS. $\sum$ is defined as the generalized choice among behavior expressions distinguished by the index set I. For example, if $S = \sum\{a_i; A_i \mid i \in I\}$ for some finite index set I where each A_i is either a process or an expression in an action-prefix form, and when $I = \{1, 2\}$, expression is $S = a_1; A_1 \; [] \; a_2; A_2$ and furthermore if $A_1 = b_1;$ **stop** and $A_2 = b_2;$ **stop** then $S = a_1; b_1;$ **stop** [] $a_2; b_2;$ **stop**. In this case, the LTS of this specification S is shown in Figure 4.3.

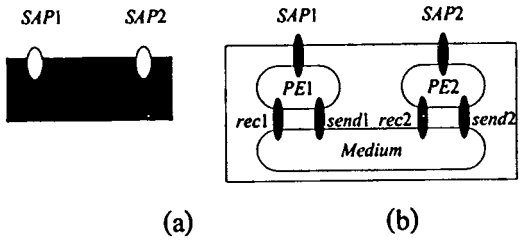


Figure 4.2 (a) Service model and (b) Protocol model

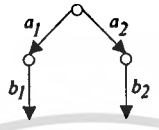


Figure 4.3 LTS of service specification

4.3.2 PROTOCOL MODEL

In contrast to service, a protocol is modeled as a white box shown in Figure 4.2(b). A protocol is a specification of communication entities that communicate with each other to provide a service at *SAPs*. Assume that the protocol model has two *SAPs*, (*SAP*1 and *SAP*2), with two protocol entities, *PE*1 and *PE*2. The protocol specification specifies the behavior of the protocol entities mediated by *Medium* which can be thought of as the whole of the lower layers. *PE*1 sends messages to *PE*2 through synchronized gate *send*1 and receives messages from *PE*2 through synchronized gate *rec*1. On the other hand, *PE*2 sends and receives messages to and from *PE*1 via synchronized gates *send*2 and *rec*2, respectively.

According to Figure 4.2(b), a protocol is expressed by the following expression in LOTOS where *Medium* is defined in Section 4.3.4.

(*PE*1 ||| *PE*2) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The service and the protocol specifications must be weakly bisimulation equivalent when internal actions, which is occurred, except that *SAPs*, are hidden from the environment.

The weak bisimulation equivalence between the service and the protocol specifications basically means that, from an external observer's point of view, the actions that occur at SAPs of the service specification are indistinguishable from the actions that occur at SAPs of the protocol specification.

4.3.3 POLLING MECHANISM

In general, protocol entities communicate in parallel via *Medium* as shown in Figure 4.2(b). Then, the order of actions in protocol and service specifications may be different when the actions at different *SAP*s were chosen by the environment. The service and the protocol may not be bisimilar, because the temporal order may not be maintained. Therefore, a mechanism will be applied to exchange a polling message between entities in order to maintain the order. Figure 4.4(a) shows the LTS of a simple service specification where action a_1 occurs at *SAP*1 and action b_2 occurs at *SAP*2. Figure 4.4(b) shows the LTS of the corresponding protocol specification using the polling mechanism for each node. Nodes 1 and 2 exchange the polling messages (*send*1!*poll*, *rec*2!*poll*, *send*2!*poll* and *rec*1!*poll*), when processes exchange message (*message*), by synchronizing at the gate (*gate*), it is represented as *gate*!*message*, with each other at the point where actions at different *SAP*s are possible to occur. It is easily shown that Figure 4.4(a) and 4.4(b) are weakly bisimulation equivalent when the polling messages were hided. The exchange of polling messages is thus a simple but a powerful way of accomplishing the weak bisimulation equivalence. Figure 4.5 shows the LTS of protocol specification without polling mechanism. In this case, each action occurs at each *SAP* independently. So the LTSs shown in Figure 4.4(a) and 4.5 are not weakly bisimulation equivalent because the temporal order is different.

The polling mechanism was used when the composition technique requires one to preserve the weak bisimulation equivalence between service and protocol specifications as well as when a service is decomposed into the corresponding protocol.

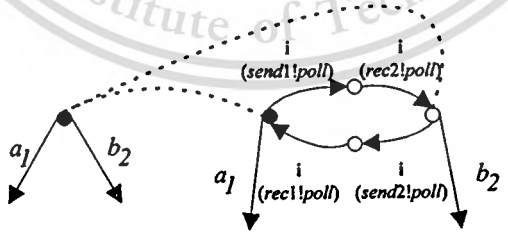


Figure 4.4 (a) LTS of service specification and (b) LTS of protocol specification with polling mechanism

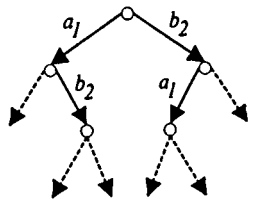


Figure 4.5 LTS of protocol specification without polling mechanism

4.3.4 SUMMARY OF THE DECOMPOSITION ALGORITHM

The basic idea of the decomposition algorithm [2] is to send either a sent or a received action signal after an observable action, then decompose a service specification *S*. Assume that *Med*1 and *Med*2 are buffers within *Medium* which is able to hold one message simultaneously. A message from *PE*1 to *PE*2 is sent through synchronized gate *send*1 and received at synchronized gate *rec*2. A message from *PE*2 to *PE*1 is sent via synchronized gate *send*2 and received at synchronized gate *rec*1. *Medium* is defined as follows,

$$Medium = Med1 \;|||\; Med2$$

where $Med1 = \sum\{send1!m;\ rec2!m;\ Med1 \mid m \in M\}$

$$Med2 = \sum\{send2!m;\ rec1!m;\ Med2 \mid m \in M\}$$

where M is the universe of messages.

In this algorithm, specifications of protocol entities were identified by **T1act**(*S*), **T1pas**(*S*), **T2act**(*S*), and **T2pas**(*S*). **T1act**(*S*) and **T1pas**(*S*) correspond to *PE*1. **T2act**(*S*) and **T2pas**(*S*) correspond to *PE*2. **T1act**(*S*) and **T1pas**(*S*) are not independent of each other. If an action in **T1act**(*S*) has been executed, **T1pas**(*S*) should be notified in order to produce the appropriate behavior after the action. This notification is done by a message on synchronization via a synchronization gate. **T1act**(*S*) is a process that contains sending actions and **T1pas**(*S*) is a process that contains receiving actions. **T2act**(*S*) and **T2pas**(*S*) are similarly explained.

As any internal actions are not treated in this reseach, the decomposition that contains the internal action **i** is omitted. The decomposition algorithm is shown as follows. Assume that an expression *S* in an action-prefix form has been given. This algorithm decomposes *S* into **T1act**(*S*), **T1pas**(*S*), **T2act**(*S*), and **T2pas**(*S*), where actions of node 1 are a_i ($i \in$ I) and actions of node 2 are b_j ($j \in$ J).

Decomposition Algorithm

$$S = \sum \{a_i; A_i \mid i \in I\} \; [] \; \sum \{b_j; B_j \mid j \in J\}$$

Then,

$$\mathbf{T1act}(S) = \sum \{a_i; send1!\, m_i; \mathbf{T1pas}(A_i) \mid i \in I\}$$
$$[]\ send1!poll; (\sum \{rec1!\, m_j; \mathbf{T1act}(B_j) \mid j \in J\}$$
$$[]\ rec1!poll; \mathbf{T1act}(S))$$

$$\mathbf{T1pas}(S) = \sum \{ rec1!\, m_j; \mathbf{T1act}(B_j) \mid j \in J\}$$
$$[]\ rec1!poll; (\sum \{ a_i; send1!\, m_i; \mathbf{T1pas}(A_i) \mid i \in I\}$$
$$[]\ send1!poll; \mathbf{T1pas}(S))$$

$$\mathbf{T2act}(S) = \sum \{ b_j; send2!\, m_j; \mathbf{T2pas}(B_j) \mid j \in J\}$$
$$[]\ send2!poll; (\sum \{rec2!\, m_i; \mathbf{T2act}(A_i) \mid i \in I\}$$
$$[]\ rec2!poll; \mathbf{T2act}(S))$$

$$\mathbf{T2pas}(S) = \sum \{ rec2!\, m_i; \mathbf{T2act}(A_i) \mid i \in I\}$$
$$[]\ rec2!poll; (\sum \{ b_j; send2!\, m_j; \mathbf{T2pas}(B_j) \mid j \in J\}$$
$$[]\ send2!poll; \mathbf{T2pas}(S))$$

[Theorem 1] Let S be a process in an action-prefix form and **T1act**(S) and **T2pas**(S) defined by the decomposition algorithm. Then

$$S \approx \textbf{hide}\ send1, rec1, send2, rec2\ \textbf{in}\ (\mathbf{T1act}(S) \;|||\; \mathbf{T2pas}(S))$$
$$|[send1, rec1, send2, rec2]|\ Medium$$

<Example of decomposition>

The decomposition algorithm is applied to a simple service specification as described below.

Given $S = a_1; b_2;$ **stop**, $S' = b_2;$ **stop**, and $S'' =$ **stop**

The result of decomposition of the service is shown as follows.

$$\mathbf{T1act}(S) = a_1; send1!a_1; \mathbf{T1pas}(S')\ []\ send1!poll; rec1!poll; \mathbf{T1act}(S)$$

$$\mathbf{T1pas}(S') = rec1!b_2; \mathbf{T1act}(S'')\ []\ rec1!poll; send1!poll; \mathbf{T1pas}(S')$$

$$\mathbf{T1act}(S'') = send1!poll; rec1!poll; \mathbf{T1act}(S'')$$

$$\mathbf{T2pas}(S) = rec2!a_1; \mathbf{T2act}(S')\ []\ rec2!poll; send2!poll; \mathbf{T2pas}(S)$$

$$\mathbf{T2act}(S') = b_2; send2!b_2; \mathbf{T2pas}(S'')\ []\ send2!poll; rec2!poll; \mathbf{T2act}(S')$$

$$\mathbf{T2pas}(S'') = rec2!poll; send2!poll; \mathbf{T2pas}(S'')$$

4.4 COMPOSITION METHOD

In this section, the composition method for service and protocol specifications is proposed. First, outline of composition of specifications is described. Next, the characteristics of the composition method are given. Then, the composition methods shall be explained.

4.4.1 OUTLINE OF COMPOSITION

The composition method assumes that protocol specifications $P1$ and $P2$ are derived from the service specifications $S1$ and $S2$ respectively by applying the decomposition algorithm described in Section 4.3.4. Under this assumption, $S1$ and $S2$, as well as $P1$ and $P2$ are composed. Then, the composition of the service and the protocol specifications would be weakly bisimilar and is described as follows (see Figure 4.6).

$S1*S2 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** ((**T1act**($S1$) * **T1act**($S2$)) |||

(**T2pas**($S1$) * **T2pas**($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

Here, the symbol * represents the LOTOS operator which can be any of the following: enabling, choice, parallel, or disabling. The enabling ">>" can be used for serializing the phases, the choice "[]" for selecting features, the parallel "|[*G*]|" for combining functions executed in parallel, and the disabling "[>" for disconnection or interruption.

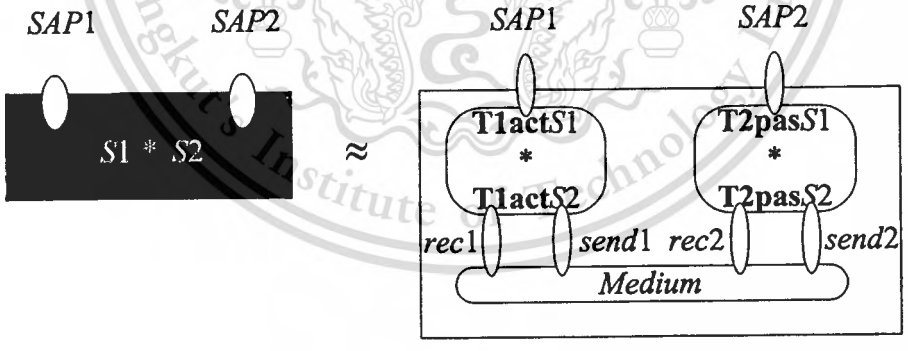


Figure 4.6 Relation between the composed service and protocol specifications

4.4.2 CHARACTERISTICS OF COMPOSITION METHOD

In this section, the preconditions and the characteristics of composition and decomposition of the specifications are explained. In order to manifest that the composition of the protocols can be done without any contradiction, it is important to keep the weak bisimulation equivalence between the composition of the service specifications which represent the external behaviors of

the system and the composition of the protocol specifications which represent how to provide the services in the system. The protocol specifications are derived by the decomposition of the service specifications using the Langerak's decomposition algorithm. In case of decomposition in the asynchronous model, maintaining the order of actions is controlled by sending and receiving the synchronization actions through the medium between the entities.

The expression S as the object of the decomposition means the service which may contain some choices between different nodes. Therefore the polling messages should be added to the derived protocol specification to maintain the weak bisimulation equivalence with the service specification. Two service specifications, which correspond to the protocol specifications $P1$ and $P2$, have been considered. The polling messages exchanged between entities of $P1$ and the polling messages exchanged between entities of $P2$ are distinguished as *poll* and *poll'*, respectively, as shown in Figure 4.7, 4.8 and 4.9. The composition method becomes more simple using polling mechanism.

The basic idea of the composition of service and protocol specifications is described as follows.

Given the service specifications $S1$ and $S2$ corresponding to the following protocol specifications $P1$ and $P2$ respectively, we have;

$P1 = (\mathbf{T1act}(S1) \,|||\, \mathbf{T2pas}(S1)) \,|[send1, rec1, send2, rec2]|\, Medium$

$P2 = (\mathbf{T1act}(S2) \,|||\, \mathbf{T2pas}(S2)) \,|[send1, rec1, send2, rec2]|\, Medium$

$S1$ and $S2$ are weakly bisimulation equivalent to $P1$ and $P2$ respectively, after hiding the notification actions *send*1, *rec*1, *send*2 and *rec*2, as follows.

$S1 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P1$

$S2 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P2$

In order to compose the protocol specifications corresponding to the service specifications under the preconditions mentioned above, the entities that correspond to the same node (*SAP*) in $P1$ and $P2$ were composed. Composition of service and protocol specification has been done as follows.

Service specification :

$S1 * S2$

Protocol specification:

$((\mathbf{T1act}(S1)*\mathbf{T1act}(S2)) \,|||\, (\mathbf{T2pas}(S1)*\mathbf{T2pas}(S2))) \,|[send1, rec1, send2, rec2]|\, Medium$

The composition of the services should become weakly bisimulation equivalent to the composition of the protocols as shown below when the notification actions *send*1, *rec*1, *send*2, and *rec*2 are hidden.

$S1$ * $S2 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** ((**T1act**($S1$)***T1act**($S2$)) |||

(**T2pas**($S1$)***T2pas**($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The symbol "*" means ">>", "|[*G*]|", "[]", or "[>" in the LOTOS operators. The protocol composition methods which are choice, enabling, and parallel compositions agree with the expression mentioned above. The polling messages, which have been kept during the decomposing service specification, work well in maintaining equivalency when the choice between different nodes exists.

<Example of composition>

An example is shown where the composition operator is the choice. Service specifications $S1$ and $S2$ are given as follows.

$S1 = a_1; b_2;$ **stop**

$S2 = c_2; d_1;$ **stop**

After decomposition of services into protocol entities, the entities at the same node together are composed. The entities of each node, node 1 and node 2, are shown in Figure 4.7 and 4.8, respectively. The resultant protocol specification is shown in Figure 4.9.

Note that an action subscript i (i = 1, 2), such as a_1 and c_2, represents the action executed at the SAP i (node i). This notation is used throughout the rest of the paper.

As shown in this example, in order to make the composition methods more general and simpler, several polling messages are introduced in protocol specification. However, unnecessary polling messages can be removed without any problem. For instance, in Figure 4.7, polling messages can be removed without any effect to the system except polling messages between actions a_1 and *rec*1!c_2. If the polling messages are not for the choice between different nodes, they can be removed.

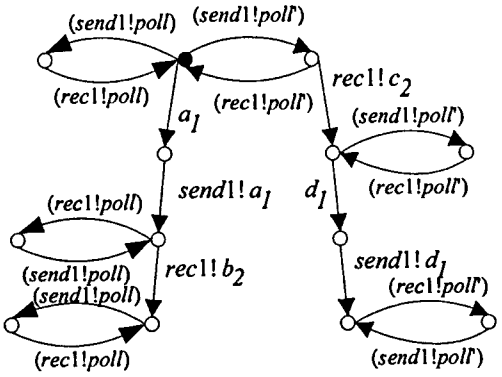


Figure 4.7 Protocol entity 1

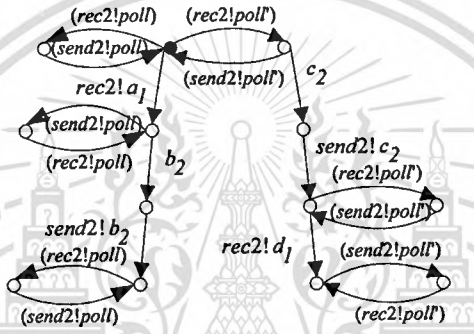
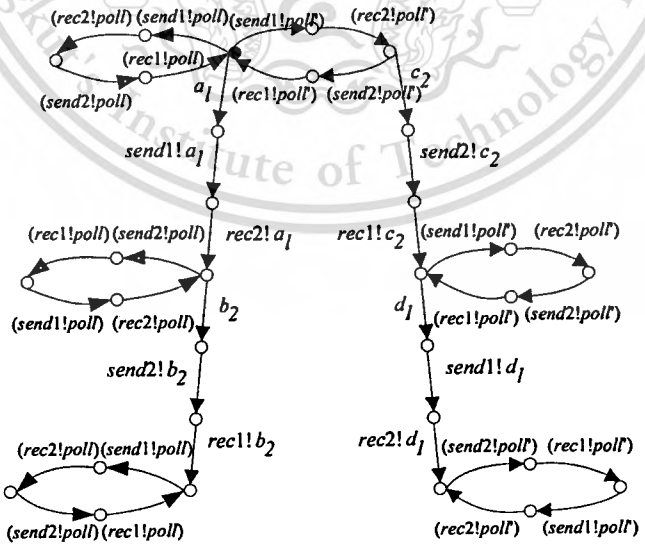


Figure 4.8 Protocol entity 2



((T1act(*S*1) [] T1act(*S*2)) ||| (T2pas(*S*1) [] T2pas(*S*2)))
|[*send*1, *rec*1, *send*2, *rec*2**]|** *Medium*

Figure 4.9 LTS of resultant protocol specification

In case of disabling, the composition of the services and the protocols does not become weakly bisimulation equivalent by using the above expression.

Disabling is the composition that a service can interrupt every state of another service. The polling messages, which have been kept during the decomposition of the service specification, work to maintain the equivalency only in the case of choice between the initial actions. Therefore, the other composition method needs to be applied. The disabling expression is converted into an expression that contains choices. The derivation process of disabling method is described in Section 4.3.4. Intermediate behavior expressions P and Q, which correspond to node 1 and node 2 respectively, have been created for disabling composition as follows.

$S1$ [> $S2 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** (P ||| Q) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

4.4.3 COMPOSITION METHODS

In this section, four types of the composition methods corresponding to the LOTOS operators, ">>", "|[G]|", "[]", and "[>" are shown.

Due to the introduction of polling messages, the composition methods ensure that the protocol specification and the service specification are weakly bisimilar. The proof of the correctness of the composition methods can be done by the expansion technique [2]. In the expansion technique, the protocol specification which consists of parallel composition of protocol entities and the medium is expanded (flattened) using the inference rules of the LOTOS parallel operator until the protocol specification cannot be expanded [3].

4.4.3.1 ENABLING

Enabling composition between two service specifications $S1$ and $S2$ is shown as "$S1$ >> $S2$" using the LOTOS operator ">>". This means that after $S1$ normally completed, $S2$ occurs. So we directly compose the protocol entities at the same node of $P1$ and $P2$, and the equivalency is maintained.

[Method 1]

The following protocol specifications $P1$ and $P2$ are derived from the service specifications $S1$ and $S2$ by the decomposition algorithm.

$P1$ = (**T1act**($S1$) ||| **T2pas**($S1$)) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

$P2$ = (**T1act**($S2$) ||| **T2pas**($S2$)) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

$S1$ and $S2$ are weakly bisimulation equivalent to $P1$ and $P2$, respectively when the notification actions *send*1, *rec*1, *send*2, and *rec*2 are hidden and are shown below.

$S1 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P1$

$S2 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P2$

The compositions of the services and the protocols are executed as follows.

Service: $S1 >> S2$

Protocol: ((**T1act**($S1$) >> **T1act**($S2$)) ||| (**T2pas**($S1$) >>

T2pas($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

<Example of Method 1>

Service specifications $S1$ and $S2$ are given as follows.

$S1 = a_1; b_2;$ **exit**, $S1' = b_2;$ **exit**, $S1'' =$ **exit**

$S2 = c_2; d_1;$ **stop**, $S2' = d_1;$ **stop**, $S2'' =$ **stop**

Decomposing them can get the protocol specifications, $P1$ and $P2$.

$P1$ = (**T1act**($S1$) ||| **T2pas**($S1$)) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

T1act($S1$) = a_1; *send*1!a_1; **T1pas**($S1'$) [] *send*1!*poll*; *rec*1!*poll*; **T1act**($S1$)

T1pas($S1'$) = *rec*1!b_2; **T1act**($S1''$) [] *rec*1!*poll*; *send*1!*poll*; **T1pas**($S1'$)

T1act($S1''$) = *send*1!*poll*; *rec*1!*poll*; **T1act**($S1''$)

T2pas($S1$) = *rec*2!a_1; **T2act**($S1'$) [] *rec*2!*poll*; *send*2!*poll*; **T2pas**($S1$)

T2act($S1'$) = b_2; *send*2!b_2;**T2pas**($S1''$) [] *send*2!*poll*; *rec*2!*poll*; **T2act**($S1'$)

T2pas($S1''$) = *rec*2!*poll*; *send*2!*poll*; **T2pas**($S1''$)

$P2$ = (**T1act**($S2$) ||| **T2pas**($S2$)) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

T1act($S2$) = *send*1!*poll*′; (*rec*1!c_2; **T1act**($S2'$) [] *rec*1! *poll*′; **T1act**($S2$))

T1act($S2'$) = d_1; *send*1!d_1; **T1pas**($S2''$) [] *send*1! *poll*′; *rec*1! *poll*′; **T1act**($S2'$)

T1pas($S2''$) = *rec*1! *poll*′; *send*1!*poll*′; **T1pas**($S2''$)

T2pas($S2$) = *rec*2! *poll*′; (c_2; *send*2!c_2; **T2pas**($S2'$) [] *send*2! *poll*′; **T2pas**($S2$))

T2pas($S2'$) = *rec*2!d_1; **T2act**($S2''$) [] *rec*2! *poll*′; *send*2! *poll*′; **T2pas**($S2'$)

T2act($S2''$) = *send*2! *poll*′; *rec*2! *poll*′; **T2act**($S2''$)

Services and protocols are weakly bisimilar when the notification actions are hidden.

$S1 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P1$

$S2 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P2$

Then, the compositions of the specifications by applying Method 1 are as follows.

Service specification:

$S1 >> S2 = a_1; b_2;$ **exit** >> $c_2; d_1;$ **stop**

Protocol specification:

((**T1act**($S1$) >> **T1act**($S2$)) ||| (**T2pas**($S1$) >> **T2pas**($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The compositions of the services and protocols are weakly bisimilar when the notification actions are hidden.

$S1$ >> $S2$ ≈ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** ((**T1act**($S1$) >> **T1act**($S2$)) |||

(**T2pas**($S1$) >> **T2pas**($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

4.4.3.2 PARALLEL

If G is a set of the synchronization actions at $S1$ and $S2$, the parallel composition between two service specifications $S1$ and $S2$ is shown as "$S1$ |[G]| $S2$" using the LOTOS operator "|[G]|" and $G = Act(S1) \cap Act(S2)$. If $G = \varnothing$, $S1$ and $S2$ become an asynchronous parallel composition, and if $G \neq \varnothing$, they become a synchronous parallel composition synchronized at the actions in G. In case that $S1$ and $S2$ have no synchronization actions, the actions of $S1$ and $S2$ can occur independently with any order. On the other hand, in case that $S1$ and $S2$ have some synchronization actions, these actions must occur synchronously at $S1$ and $S2$, and the other actions occur independently. In the case of protocol composition, it could be composed the protocol entities in parallel at the same node of $P1$ and $P2$, respectively. If $P1$ and $P2$ have some synchronization actions, they will be synchronously composed in their actions and notification actions.

[Method 2]

The precondition is the same as Method 1. When we express the synchronization actions as $G = Act(S1) \cap Act(S2)$, $G1 = Act(\textbf{T1act}(S1)) \cap Act(\textbf{T1act}(S2))$ and $G2 = Act(\textbf{T2pas}(S1)) \cap Act(\textbf{T2pas}(S2))$, the execution of the parallel composition of the services and the protocols are as follows.

Service: $S1$ |[G]| $S2$

Protocol: ((**T1act**($S1$) |[$G1$]| **T1act**($S2$)) ||| (**T2pas**($S1$) |[$G2$]| **T2pas**($S2$)))

|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

<Example of Method 2>

Service specifications $S1$ and $S2$ are given as follows.

$S1 = a_1$; b_2; **stop**, $S1' = b_2$; **stop**, $S1'' =$ **stop**

$S2 = c_2$; a_1; **stop**, $S2' = a_1$; **stop**, $S2'' =$ **stop**

Decomposing them can get the protocol specifications, $P1$ and $P2$.

$P1$ = (**T1act**($S1$) ||| **T2pas**($S1$)) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

T1act($S1$) = a_1; *send*1!a_1; **T1pas**($S1'$) [] *send*1!*poll*; *rec*1!*poll*; **T1act**($S1$)

T1pas($S1'$) = *rec*1!b_2; **T1act**($S1''$) [] *rec*1!*poll*; *send*1!*poll*; **T1pas**($S1'$)

T1act($S1''$) = *send*1!*poll*; *rec*1!*poll*; **T1act**($S1''$)

T2pas($S1$) = *rec*2!a_1; **T2act**($S1'$) [] *rec*2!*poll*; *send*2!*poll*; **T2pas**($S1$)

T2act($S1'$) = b_2; *send*2!b_2;**T2pas**($S1''$) [] *send*2!*poll*; *rec*2!*poll*; **T2act**($S1'$)

T2pas($S1''$) = *rec*2!*poll*; *send*2!*poll*; **T2pas**($S1''$)

$P2$ = (**T1act**($S2$) ||| **T2pas**($S2$)) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

T1act($S2$) = *send*1!*poll*′; (*rec*1!c_2; **T1act**($S2'$) [] *rec*1! *poll*′; **T1act**($S2$))

T1act($S2'$) = a_1; *send*1!a_1; **T1pas**($S2''$) [] *send*1! *poll*′; *rec*1! *poll*′; **T1act**($S2'$)

T1pas($S2''$) = *rec*1! *poll*′; *send*1!*poll*′; **T1pas**($S2''$)

T2pas($S2$) = *rec*2! *poll*′; (c_2; *send*2!c_2; **T2pas**($S2'$) [] *send*2! *poll*′; **T2pas**($S2$))

T2pas($S2'$) = *rec*2!a_1; **T2act**($S2''$) [] *rec*2! *poll*′; *send*2! *poll*′; **T2pas**($S2'$)

T2act($S2''$) = *send*2! *poll*′; *rec*2! *poll*′; **T2act**($S2''$)

Services and protocols are weakly bisimilar when the notification actions are hidden.

$S1$ ≈ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P1$

$S2$ ≈ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P2$

Then, the compositions of the specifications by applying Method 2 are given as follows.

Service specification:

$S1$ |[a_1]| $S2$ = a_1; b_2; **stop** |[a_1]| c_2; a_1; **stop**

Protocol specification:

((**T1act**($S1$) |[a_1, *send*1!a_1]| **T1act**($S2$)) ||| (**T2pas**($S1$) |[*rec*2!a_1]| **T2pas**($S2$)))

|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The compositions of the services and protocols are weakly bisimilar when the notification actions are hidden.

$S1$ |[a_1]| $S2$ ≈ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** ((**T1act**($S1$) |[a_1, *send*1!a_1]| **T1act**($S2$)) |||

(**T2pas**($S1$) |[*rec*2!a_1]| **T2pas**($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

4.4.3.3 CHOICE

Choice composition between two service specifications, $S1$ and $S2$, is shown as "S1[]S2" by using the LOTOS operator "[]". This means that either $S1$ or $S2$ is selected. The polling

messages, which have been kept when decomposing the service specifications, work to maintain the equivalency.

[**Method 3**]

The precondition is the same as Method 1. The choice composition of the services and the protocols is excuted as follows.

Service: $S1$ [] $S2$

Protocol: ((**T1act**($S1$) [] **T1act**($S2$)) ||| (**T2pas**($S1$) [] **T2pas**($S2$)))

|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

<Example of Method 3>

Service specifications $S1$ and $S2$ are given as follows.

$S1 = a_1; b_2;$ **stop**, $S1' = b_2;$ **stop**, $S1'' =$ **stop**

$S2 = c_2; d_1;$ **stop**, $S2' = d_1;$ **stop**, $S2'' =$ **stop**

Decomposing them will get the protocol specifications, $P1$ and $P2$

$P1$ = (**T1act**($S1$) ||| **T2pas**($S1$)) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

T1act($S1$) = a_1; *send*1!a_1; **T1pas**($S1'$) [] *send*1!*poll*; *rec*1!*poll*; **T1act**($S1$)

T1pas($S1'$) = *rec*1!b_2; **T1act**($S1''$) [] *rec*1!*poll*; *send*1!*poll*; **T1pas**($S1'$)

T1act($S1''$) = *send*1!*poll*; *rec*1!*poll*; **T1act**($S1''$)

T2pas($S1$) = *rec*2!a_1; **T2act**($S1'$) [] *rec*2!*poll*; *send*2!*poll*; **T2pas**($S1$)

T2act($S1'$) = b_2; *send*2!b_2;**T2pas**($S1''$) [] *send*2!*poll*; *rec*2!*poll*; **T2act**($S1'$)

T2pas($S1''$) = *rec*2!*poll*; *send*2!*poll*; **T2pas**($S1''$)

$P2$ = (**T1act**($S2$) ||| **T2pas**($S2$)) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

T1act($S2$) = *send*1!*poll*′; (*rec*1!c_2; **T1act**($S2'$) [] *rec*1! *poll*′; **T1act**($S2$))

T1act($S2'$) = d_1; *send*1!d_1; **T1pas**($S2''$) [] *send*1! *poll*′; *rec*1! *poll*′; **T1act**($S2'$)

T1pas($S2''$) = *rec*1! *poll*′; *send*1!*poll*′; **T1pas**($S2''$)

T2pas($S2$) = *rec*2! *poll*′; (c_2; *send*2!c_2; **T2pas**($S2'$) []*send*2! *poll*′; **T2pas**($S2$))

T2pas($S2'$) = *rec*2!d_1; **T2act**($S2''$) []*rec*2! *poll*′; *send*2! *poll*′; **T2pas**($S2'$)

T2act($S2''$) = *send*2! *poll*′; *rec*2! *poll*′; **T2act**($S2''$)

Services and protocols are weakly bisimilar when the notification actions are hidden.

$S1 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P1$

$S2 \approx$ **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P2$

Then, the compositions of the specifications can be obtained by applying Method 3 as follows.

Service specification:

$S1$ [] $S2 = a_1$; b_2; **stop** [] c_2; d_1; **stop**

Protocol specification: (See in Figs. 4.6, 4.7 and 4.8)

((**T1act**($S1$) [] **T1act**($S2$)) ||| (**T2pas**($S1$) [] **T2pas**($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The compositions of the services and protocols are weakly bisimilar when the notification actions are hidden.

$S1$ [] $S2$ = **hide** *send*1, *rec*1, *send*2, *rec*2 **in** ((**T1act**($S1$) [] **T1act**($S2$)) |||

(**T2pas**($S1$) [] **T2pas**($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

4.4.3.4 DISABLING

Disabling composition between two service specifications $S1$ and $S2$ shown as "$S1$ [> $S2$" is using the LOTOS operator "[>". The choice composition method cannot be applied to disabling composition directly, because the equivalency is not maintained. Therefore, the research needs to consider another method. The expression of disabling is converted into another one consisting of only the choice, and considered the disabling method as a repetition of choice. Before going to the method, the process of developing the disabling method is shown in an example.

[Derivation Process of Disabling Method]

Assume the following service specifications $S1$ and $S2$.

$S1 = a_1$; b_2; **stop**

$S2 = c_2$; d_1; **stop**

$B = d_1$; **stop**

Let S be the following service composed by disabling.

$S = S1$ [> $S2 = a_1$; b_2; **stop** [> c_2; B

The above expression can be converted into another one consisting of only the choice shown below.

$S = a_1$; (b_2; (**stop** [] c_2; B) [] c_2; B) [] c_2; B

The above expression is divided into each choice expression as follows.

$A_0 = a_1$; A_1 [] c_2; B

$A_1 = b_2$; A_2 [] c_2; B

A_2 = **stop** [] c_2; $B = c_2$; B

The protocol specifications of $S1$ and $S2$ correspond to the following $P1$ and $P2$ which have a style without the polling messages.

$P1=(E1 \;|||\; E2)\; |[send1, rec1, send2, rec2]|\; Medium$

$P2=(F1 \;|||\; F2)\; |[send1, rec1, send2, rec2]|\; Medium$

where entities correspondence are as follows.

$E1$ and $F1$ are entities at node1.

$E2$ and $F2$ are entities at node2.

The specifications of each entity are as follows.

$E1= a_1; send1!a_1; rec1!b_2;$ **stop**

$E2= rec2!a_1; b_2; send2!b_2;$ **stop**

$F1= rec1!c_2; d_1; send1!\, d_1;$ **stop**

$F2= c_2; send2!c_2; rec2!\, d_1;$ **stop**

Generally, the weak bisimulation equivalence is not preserved by a straightforward composition between protocol entities corresponding to the same node of $S1$ and $S2$.

$S1 [> S2 \not\approx$ **hide** $send1, rec1, send2, rec2$ **in** $((E1 [> F1) \;|||\; (E2 [> F2))$

$|[send1, rec1, send2, rec2]|\; Medium$

So the intermediate behavior expressions, P and Q, which satisfy the next formula, are introduced.

$S1 [> S2 \approx$ **hide** $send1, rec1, send2, rec2$ **in** $(P \;|||\; Q)\; |[\, send1, rec1, send2, rec2]|\; Medium$

P and Q correspond to the protocol entities at node 1 and node 2, respectively.

In order to derive P and Q, the decomposition of A_0 is considered. A_0 is a choice between a_1 and c_2 at the different nodes. The polling messages have to be added. Many parts of the protocol specification corresponding to a_1 and c_2 at node 1 are "a_1; $send1!a_1$" and "$rec1!c_2$", respectively and the parts of the protocol specification corresponding to a_1 and c_2 at node 2 are "$rec2!a_1$" and "c_2; $send2!c_2$", respectively. Therefore, the intermediate behavior expressions corresponding to A_0 are shown in the following.

$P= a_1; send1!a_1; P_1 [] send1!poll; (rec1!c_2; B [] rec1!poll; P)$

$Q= rec2!a_1; Q_1 [] rec2!poll; (c_2; send2!c_2; B [] send2!poll; Q)$

Next, the decomposition of A_1 is considered. A_1 is a choice between b_2 and c_2 at the same node so it does not need to add the polling message. Many parts of the protocol specification correspond to b_2 and c_2 at node 1 are "$rec1!b_2$" and "rec1!c2", respectively, and the parts of the protocol specification correspond to b_2 and c_2 at node 2 are "b_2; $send2!b_2$" and "c_2; $send2!c_2$",

respectively. Therefore, the intermediate behavior expressions corresponding to A_1 are given in the following.

$P_1 = rec1!b_2;\ P_2\ []\ rec1!c_2;\ B$

$Q_1 = b_2;\ send2!b_2;\ Q_2\ []\ c_2;\ send2!c_2;\ B$

Next, the decomposition of A_2 is considered. A_2 does not include choice. So, the intermediate behavior expressions corresponding to A_2 consist of only the actions related with c_2 as follows.

$P_2 = rec1!c_2;\ B$

$Q_2 = c_2;\ send2!c_2;\ B$

Finally, the intermediate behavior expressions P and Q can be achieved by substituting P_2 and Q_2 for P_1 and Q_1 respectively, and by substituting P_1 and Q_1 for P and Q respectively. Namely, when the disabling composition is excuted, the expression of disabling composition is converted into another one consisting of only the choice operators, then it derives P and Q, and P and Q are finally composed.

[Method 4]

When the disabling composition is excuted, the expression of disabling composition is converted into another one consisting of only the choice operators, then P and Q are derived, and finally P and Q are composed.

P1 and *P2* shown below are derived from the service specifications *S*1 and *S*2 by applying the Langerak's decomposition algorithm without polling messages.

$P1 = (E1 \ |||\ E2)\ |[\ send1, rec1, send2, rec2]|\ Medium$

$P2 = (F1 \ |||\ F2)\ |[\ send1, rec1, send2, rec2]|\ Medium$

where the correspondence of entities is shown below.

*E*1 and *F*1 are entities at *node*1.

*E*2 and *F*2 are entities at *node*2.

Based on the above mention, the disabling composition of the services and the protocols is excuted as follows.

Service: *S*1 [> *S*2

Protocol: $(P \ |||\ Q)\ |[send1, rec1, send2, rec2]|\ Medium$

The intermediate behavior expressions P and Q are derived by repetition of the procedure shown below.

If **InitM** (*E*1) $\cup$ **InitM** (*F*1) $\subseteq$ **ActR** (*N*1) or

InitM ($E1$) $\cup$ **InitM** ($F1$) $\subseteq$ **ActR** ($N2$)

then P = **InitM**($E1$); (**Der** ($E1$)[>$F1$) [] $F1$

else P = **InitM**($E1$); (**Der** ($E1$)[>$F1$) [] *send*1!*poll*; ($F1$ [] *rec*1!*poll*; P)

If **InitM** ($E2$) $\cup$ **InitM** ($F2$) $\subseteq$ **ActR** ($N1$) or

InitM ($E2$) $\cup$ **InitM** ($F2$) $\subseteq$ **ActR** ($N2$)

then Q = **InitM**($E2$); (**Der** ($E2$) [> $F2$) [] $F1$

else Q = **InitM**($E2$); (**Der** ($E2$) [> $F2$) [] *rec*2!*poll*; ($F2$ [] *send*2!*poll*; Q)

where **Der**($E1$) [> $F1$ means executing this procedure once again with the substition of $E1$ by **Der**($E1$), and **Der**($E2$) [> $F2$ means executing this procedure once again with the substition of $E2$ by **Der**($E2$).

The functions in this method are defined as follows.

ActR(Ni) is the set of the actions which can occur at node i and its notification actions related to these actions.

InitM(E) is the set of the pair of the actions which can occur at first in entity E and its transmission notification actions, or the set of the receiving notification actions.

Der(E) is the set of the behavior expressions immediately after the execution of the **InitM**(E).

<Example of each function>

If $E = a_1$; *send*1!a_1; *rec*1!b_2; **stop**

then **InitM**(E) = {a_1; *send*1!a_1}

Der(E) = {*rec*1!b_2; **stop**}

ActR($N1$) = {a_1; *send*1!a_1}

ActR($N2$) = {*rec*1!b_2}

4.5 SUPPORT SYSTEM

The support system has five working stages; those are (i) entering service specifications ($S1$ and $S2$), (ii) the selection of the composition operator, (iii) the decomposition, (iv) the result of composition, and (v) LTS of composition, as shown in Figure 4.10.

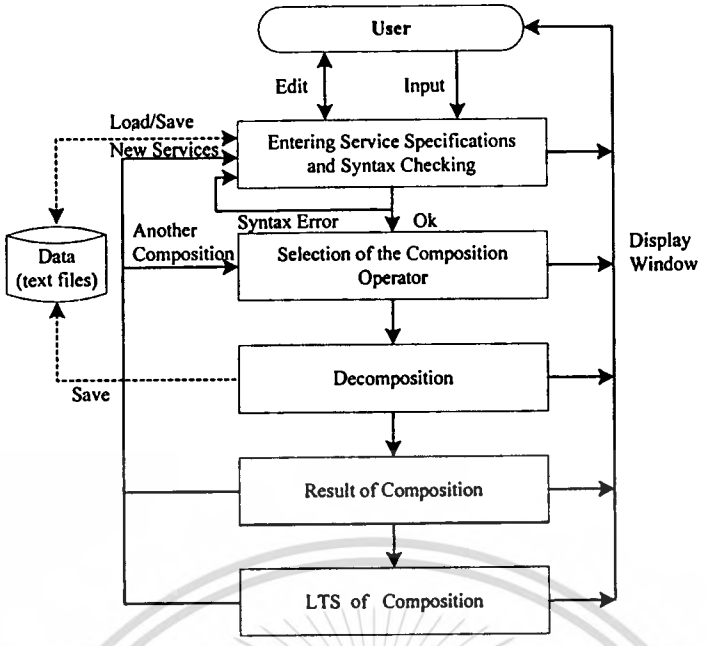


Figure 4.10 Flowchart of the support system

At the first stage, a user enters service specifications through the service specification windows and may save them as text files and reload them to specify again. One different point from LOTOS in an action prefix form is that a node number is put within "{ }". After the user enters $S1$, the support system will check the syntax of expression based on the LOTOS syntax. If $S1$ has syntax errors, the support system will show these errors. If $S1$ has no errors, the support system permits the user to enter $S2$. Entering $S2$ is the same as the case of $S1$.

At the second stage, the support system shows the composition method window for the user to select the LOTOS operator, which can be one of four types of composition operators.

At the third stage, the support system decomposes service specifications into protocol specifications automatically and shows the protocol specification window which includes protocol specifications of $S1$ and $S2$.

At the fourth stage, the support system shows that the composition of the services and the composition of protocols are weakly bisimilar when notification actions (internal actions) are hidden. The support system provides the user with the selection of another composition, the composition of new services, or LTS.

At the last stage, if LTS is selected, the support system will show LTS of the composed services.

<Application examples>

The proposed composition method and the simulation tool are applied to construct service and protocol specifications for Bulk Data Transfer part of the FTAM (File Transfer, Access and Management) [9, 10] OSI Application Layer as an example for their evaluation.

FTAM has many services such as F-READ, F-WRITE, F-TRANSFER-END, F-DATA, F-CANCEL, etc. For this application, we consider two alternative services of data transfer in FTAM, i.e., F-READ and F-WRITE services. In FTAM service, there are two associated users, *Initiator* and *Responder*. The Initiator requests a service from the Responder, and then the Responder replies to the request. After the Initiator executes an F-READ request (F-WRITE request), the F-READ service (F-WRITE service) begins.

The F-READ service specifies a data transfer from the Responder at a node to the Initiator at another node. The data will be transferred until the transfer is completed. The Responder executes an F-DATA-END request to inform the Initiator that the data transfer is completed. Then, the Initiator confirms the completion using an F-TRANSFER-END service. After that, the F-READ service is finished.

Similarly, the F-WRITE service specifies a data transfer from the Initiator to the Responder. The data will be transferred until the transfer is completed. The Initiator executes an F-DATA-END request to inform the Responder that the data transfer is completed. Then, the Initiator confirms the completion using an F-TRANSFER-END service. After that, the F-WRITE service is finished.

The LTSs of the F-READ and F-WRITE service specifications are shown in Figure 4.11. The F-READ service will start when a read request signal (F-Rd_Req1) is issued, and the F-WRITE service will start when a write request signal (F-Wt_Req1) is issued. At first, the support system shows the asynchronous protocol model to the user as shown in Figure 4.12. The user just puts the F-READ and F-WRITE service specifications into the support system as shown in Figure 4.13. Then, the support system will ask the user to select one of the four composition operators (choice, enabling, parallel and disabling) as shown in Figure 4.14. The user selects the choice operator because the two services are composed alternatively. Then, as shown in Figure 4.15, the protocol specification is automatically derived (composed) from the composed service specification, in accordance with the composition method. Finally, it is displayed that the composed service specification and the composed protocol specification are weakly bisimilar when the notification actions are hidden, as shown in Figure 4.16. Similarly, other services such

as F-TRANSFER-END and F-CANCEL can also be composed, and the corresponding equivalent protocol specification can be derived, using the proposed composition method and the support system.

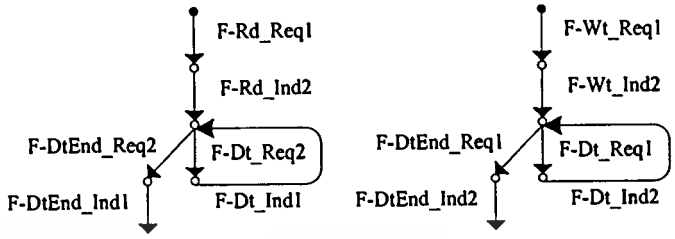


Figure 4.11 (a) LTS of F-READ service ($S1$) and (b) LTS of F-WRITE service ($S2$)

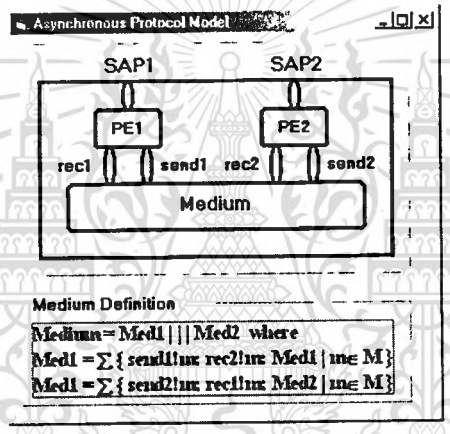
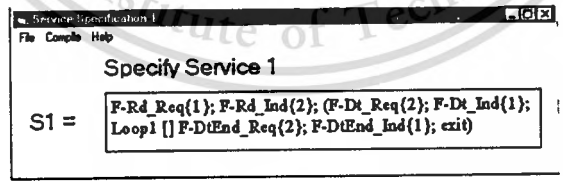
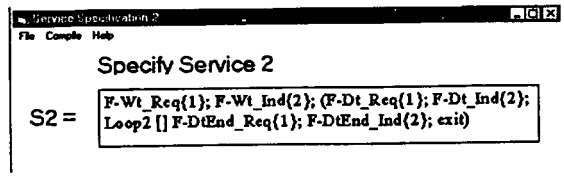


Figure 4.12 Display of asynchronous protocol model



(a)



(b)

Figure 4.13 (a) Input of service specification $S1$ and (b) Input of service specification $S2$

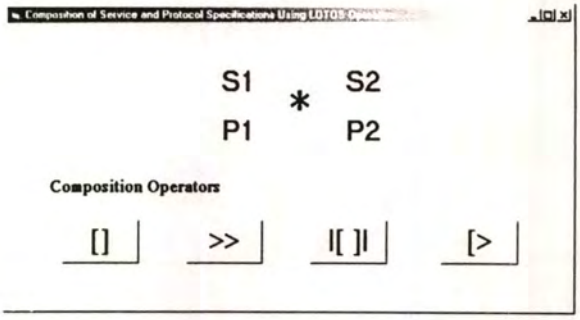


Figure 4.14 Selection of a composition operator

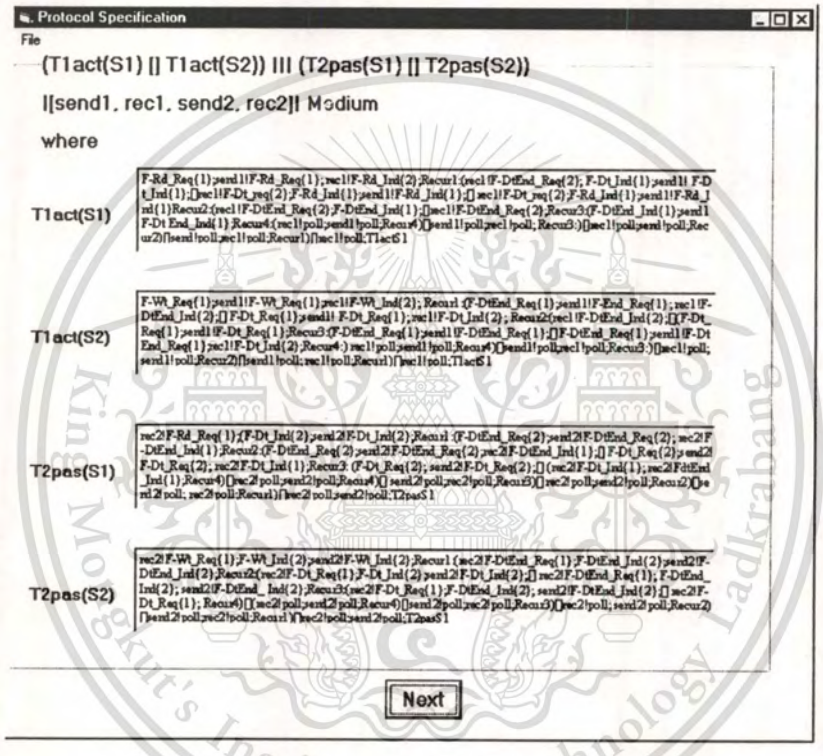


Figure 4.15 Derived protocol specifications of F-READ and F-WRITE services

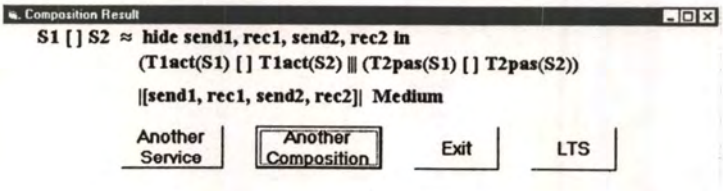


Figure 4.16 Composition result of composed services and composed protocol

The other example is a data transfer function of protocol in a computer communication system. Signals Dat_Req, Dat_Ind, Dis_Req and Dis_Ind represent Data Request, Data Indication, Disconnection Request and Disconnection Indication, respectively. The number 1 and 2 are nodes that the signals are sent. The LTSs of data request and disconnection functions are shown in Fugure 4.17.

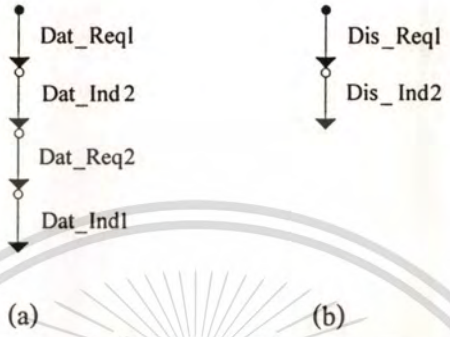
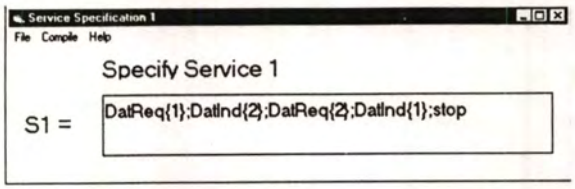


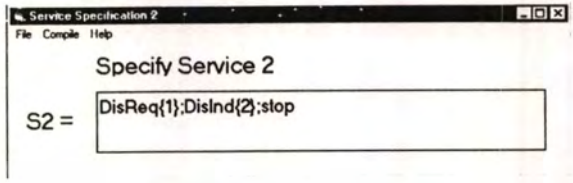
Figure 4.17 (a) LTS of data request service (*S*1) and (b) LTS of disconnection service (*S*2)

During the connection establishment phase, computer at node 1 and node 2 are connecting to transfer data. When computer at node 1 wants to cancel the connection, it will send Dis_Req1 to inform computer at node 2 to disconnect. After that, computer at node 2 will send Dis_Ind2 to indicate disconnection to computer at node 1. Therefore, the connection establishment is canceled. Figure 4.18 shows the service specifications of data transfer function. The disabling operator is selected to apply to this example. The protocol specifications of these services are derived automatically as shown in Figure 4.19. Last, the support system displays the composed service specification and the composed protocol specification which are weak bisimar each other according to the composition method, as shown in Figure 4.20.

Finally, it is noted that protocol specifications derived by applying the proposed method are somewhat complex since they contain some redundant polling messages. However, as suggested in Section 4.4.2, they could be removed, thereby relaxing the problem of efficiency.



(a)



(b)

Figure 4.18 (a) Data request service specification *S*1 and (b) Disconnection service specification *S*2

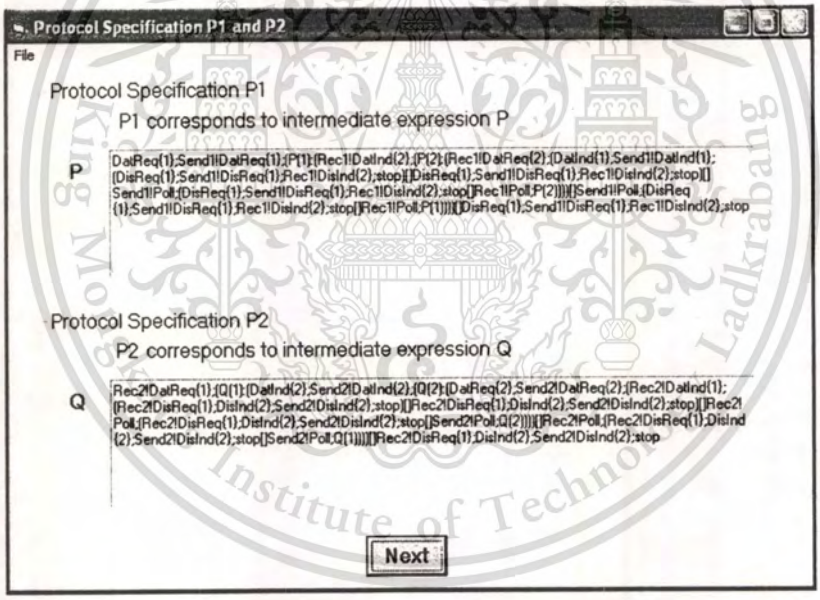


Figure 4.19 Derived protocol specifications of Data request and Disconnection services

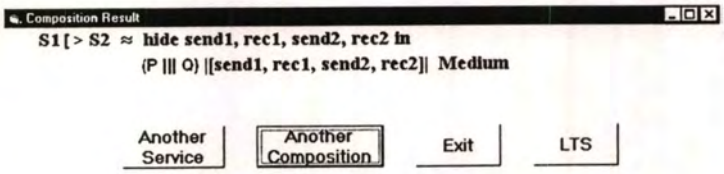


Figure 4.20 Composition result of composed services and composed protocol

Protocol specification using LOTOS is complex which is in a text format. It is difficult to understand and verify. A solution for this problem can be solved, alleviated by using a graphical method that can be used to describe a communication system. Finite State Machine model is one of such specification models which is powerful and easy to understand. The Finite State Machine model is used to be the model a communication system using the proposed specification method. It will be explained in next chapter.

CHAPTER 5

SPECIFICATION AND VERIFICATION METHOD BASED ON STATE MACHINE MODEL

5.1 INTRODUCTION

In the design of a concurrent system, it is desirable to rigorously specify the system without ambiguity. This approach makes an analysis and verification of the system possible at the system specification stage, and thus the system property such as an anomalous behavior can be investigated prior to the implementation stage.

Since a concurrent system consists of a number of communicating entities, a specification method for such a system needs to consider both concurrency and communications. The *process calculi* such as CCS [11] and CSP [12] have been proposed as specification methods. They model the entities as processes that are represented by algebraic expressions, and can deal with the concurrency among them. Also, there are some works that extend the ideas of the process calculi to mobility, e.g. π-calculus [22], and works that extends to time, e.g. Timed CSP [23] and E-LOTOS [24]. On the other hand, several specification methods based on state machine model have been proposed [13-15]. They are not abstract as the process calculi and are closer to implementation. There are some works that extend to mobility and locality [25], and to time [16-21]. Furthermore, state space analysis methods that can handle time have been proposed [20, 21].

This research is one of the specification methods based on state machine model with time, geography, and mobility constraints. First, a basic state machine is introduced. It is modeled so that it can interact with the environment via channels. By defining a communication system as a collection of such state machines, the whole behavior of the communication system is given in terms of interactions among the state machines and their individual internal activities. These notations form the foundation of a specification method.

Next, a constrained state machine is introduced which can be adapted to various specification purposes. Three types of constraint, time, geography and mobility are proposed in this research. They are employed into a state machine model. For time constraint, a specification method of state transition is also used in addition to execute enabling condition of the state transition which consists of the current state, event and timing. For geography constraint, a

specification method of communication is applied to a concurrent system to describe mobility of communication. For mobility constraint, a specification method restricts a state machine to be in only a location at a moment. On the other hand, ranges of channels cannot overlap.

A simulator tool is produced for design and analysis based on the specification method. It can be used to analyze the communication system in state diagram form. By using this tool, the behaviors of a specified system can be investigated and verified before the implementation.

Finally, a comparison of Finite State Machine model and Petri Net model for a communication system design is described. The comparison will explain why Finite State Machine is more suitable for this research. Moreover, limitation of Petri Net is also mentioned.

5.2 BASIC STATE MACHINE

Finite state machine (FSM) or finite automaton has become a standard model for representing object behaviors. A finite state machine is a model of behaviors composed of states, transitions and actions. A state stores information about the past, i.e. it reflects the input changes from the system start to the present moment. A transition indicates a state change and is described by a condition that would need to be fulfilled to enable the transition. An action is a description of an activity that is to be performed at a given moment. On the other hand, actions are normally associated with transitions. In general, FSM can be represented using a state diagram (or state transition diagram). Figure 5.1 demonstrates a state diagram of a basic state machine.

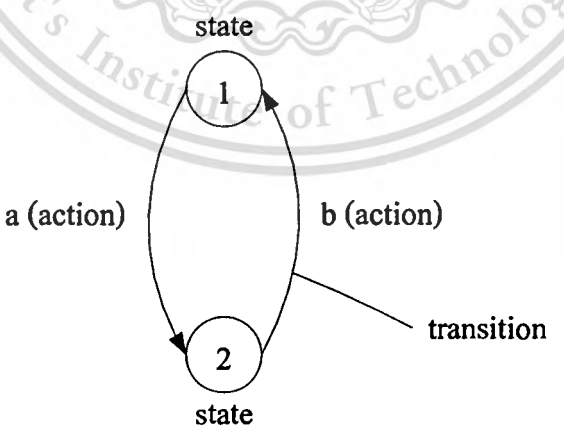


Figure 5.1 A basic state machine

A (*basic*) *state machine* has *variables* in addition to the usual (control) states. Each variable has its own *type*; such as *integer* or *boolean*. Let V be a set of variables, then a *valuation* of V is a mapping that associates each variable of V, a value in its domain. It can write **V** for the set of valuations of V.

A state machine can perform interaction with the environment or other state machines. This interaction appears as an input or output of data via a *channel*. A set of channels accepting denotes an input set I as C^{in} and a set of channels accepting denotes an output set O as C^{out}. In the following definition, a data input/output of a state machine via a channel is formulated as an *event* that causes a state transition of state machine.

Definition 1 A *state machine* M is a 5-tuple:

$$M = \langle Q, V, \sum, T, \theta \rangle$$

- Q is a finite set of *control states*.
- V is a finite set of *variables*.
- $\sum$ is a set of three *events*, those are $\sum^{in}$ ($\sum^{in} \subseteq C^{in} \times I$), $\sum^{out}$ ($\sum^{out} \subseteq C^{out} \times O$) and M, $\{\tau\}$ which are a set of input event from environment, a set of output event to the environment and a set of *internal* event, respectively.
- $T \subseteq (Q \times \mathbf{V}) \times \sum \times (Q \times \mathbf{V})$ is a *transition relation*.
- $\theta \subseteq Q \times \mathbf{V}$ is a set of *initial state*.

It can say that $s \in Q \times \mathbf{V}$ is a *state* of the state machine M. Since there may be a case where the domain of a variable is infinite, the number of states of M may not be finite.

It can be written $s \xrightarrow{\sigma} s'$ for a $(s, \sigma, s') \in T$. If there exists a state s' such that $s \xrightarrow{\sigma} s'$ for a state s and an event σ, we write $s \xrightarrow{\sigma}$.

Notation 1 A visual representation of a state machine is a usual state transition diagram. If an explicit specification of variable values is needed, it is attached to the control state. A basic structure of state transition is represented as follows:

$$current\_state \xrightarrow{guard \rightarrow event} next\_state$$

The *guard* is a boolean expression that qualifies the values of the variables related with the transition. If it is not necessary or it is always *true*, it can be omitted. The *event* is written as follows if it is an input.

channel_name?*variable_name* "value"

The expression represents the state machine inputs data with the value specified by "value" via the channel specified by *channel_name*, and assigns it to the variable specified by *variable_name*. If an input value is not apparent, part of "value" is omitted. If only "value" is important, part of *variable_name* is omitted. If data is structured; for example, a protocol message whose *type* is defined and which has some parameters, it is written as follows.

channel_name?(*parameter*, ..., *parameter*)

Part of *parameter* is represented as the above described *variable_name* "value". The *event* is written as follows if it is an output.

channel_name!*expression*

The expression represents the state machine outputs data with the value obtained by interpreting the specified *expression* via the channel specified by *channel_name*. The *expression* may be a constant (embraces by double quotation marks). For a structured data, its form conforms to the above described as follows.

channel_name!(*parameter*, ..., *parameter*)

An expression is written in *parameter* part.

If the *event* is internal, the processing statements that may change the variable should be written. The symbol τ represents an occurrence of some internal event.

Example 5.1 Figure 5.2 shows a notation example of state machines for data transmission in a communication system in which Sender transmits data to Receiver via Medium. The control states of Sender are *S*0 and *S*1. Sender inputs data into the variable *din* via the channel *p* from the environment and delivers it to Medium via the channel *s*. Sender's initial state is *S*0. The control states of Medium are *M*0, *M*1 and *M*2, and its initial control state is *M*0. There are four variables: *no*, *i*, *o* and *data*. *data* is a buffer for storing data. *no* represents the number of data in the buffer, *i* indicates the position in the buffer into which the next data is input, and *o* indicates the position in the buffer from which the next data is output. The initial values of these three variables are all 0's. *MAX* is a constant that represents the maximum of the number of data capable of storing. Within this range, data can be input into the buffer via the channel *s*. If there are one or more data in buffer, Medium can output data to Receiver via the channel *r*. While inputting and/or outputting data, the necessary update of the variable values is performed, with internal transitions. The control states of Receiver are *R*0 and *R*1, and its initial control state is *R*0. Receiver inputs data into the variable *dout* via the channel *r*, and outputs it to the environment via the channel *c*.

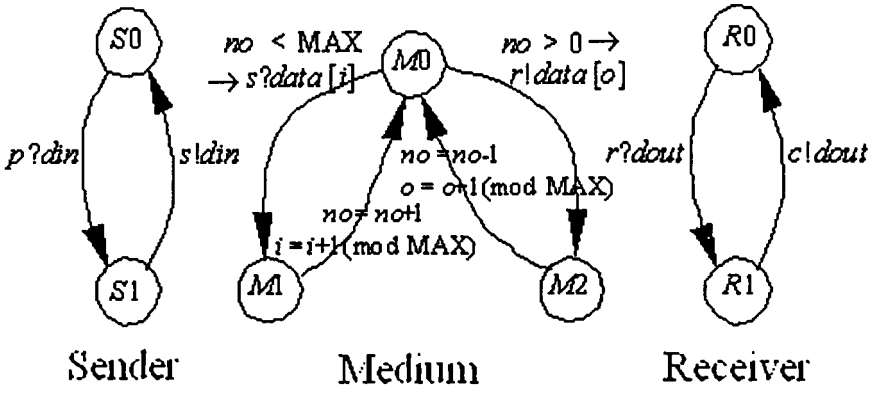


Figure 5.2 Data Transfer in a Communication System

From Figure 5.2, let us define

$\tau_1 = \{ no = no + 1; i = i + 1 (\text{mod MAX}) \}$

$\tau_2 = \{ no = no - 1; o = o + 1 (\text{mod MAX}) \}$

v = is a finite set of *variables*.

Medium consists of the following 5 sets according to Definition 5.1.

$Q = \{ \text{M0, M1, M2} \}$

$V = \{ no, i, o, data \}$

$\sum = \{ s?data[i], r!data[o], \tau_1, \tau_2 \}$

where

$\sum^{in} = \{ s?data[i] \}$

$\sum^{out} = \{ r!data[o] \}$

$\tau = \{ \tau_1, \tau_2 \}$

$T = \{ ((M0, \mathbf{v})\ s?data[i]\ (M1, \mathbf{v})), ((M0, \mathbf{v})\ r!data[o]\ (M2, \mathbf{v})),$

$((M1, \mathbf{v})\ \tau_1\ (M0, \mathbf{v})), ((M2, \mathbf{v})\ \tau_2\ (M0, \mathbf{v})) \}$

$\theta = \{ (\text{M0}, \mathbf{v}) \}$

Definition 2 A *computation* of a state machine *M* is an alternating sequence of states and events

$$\alpha = S_0 \sigma_0 S_1 \sigma_1 S_2 \sigma_2 S_3 \ldots$$

which satisfies the following conditions:

1. $S_0 \in \theta$
2. For all $i \geq 0$, $S_i \xrightarrow{\sigma_i} S_{i+1}$

The set of all computations of a state machine *M* is denoted as *Comp*(*M*).

Assume that for Medium in Example 1, a state, an input event and an output event are written as follows, respectively.

(control state, {*no*, *i* , *o*, *data*})

channel?data

channel!data

If *MAX* = 2 in Medium of Example 1, then the following sequence α is a computation of Medium:

α = (*M*0, {*no* = 0, *i* = 0, *o* = 0, *data*[] = *null*}) *s*?"d1"

(*M*1, {*no* = 0, *i* = 0, *o* = 0, *data*[] = "d1"}) τ

(*M*0, {*no* = 1, *i* = 1, *o* = 0, *data*[] = "d1"}) *s*?"d2"

(*M*1, {*no* = 1, *i* = 1, *o* = 0, *data*[] = "d1"."d2"}) τ

(*M*0, {*no* = 2, *i* = 0, *o* = 0, *data*[] = "d1"."d2"}) *r*!"d1"

(*M*2, {*no* = 2, *i* = 0, *o* = 0, *data*[] = "d1"."d2"}) τ

(*M*0, {*no* = 1, *i* = 0, *o* = 1, *data*[] = "d1"."d2"}) *s*?"d3"

(*M*0, {*no* = 1, *i* = 0, *o* = 1, *data*[] = "d3". "d2"}) .....

where "*d*1", "*d*2", "*d*3" , ... represent data values.

Definition 3 Let $\alpha = S_0\sigma_0S_1\sigma_1S_2\sigma_2S_3$ ... be a computation of a state machine *M*. The sequence obtained by removing all states and all τ-events from α is said to be a *trace* of *M*.

A trace is as an observation of a computation of a state machine from the outside (environment). The set of all traces of state machine *M* is denoted as *Trace*(*M*).

From the above mentioned example, a trace of Medium is written as follows.

tr = *s*?"d1" *s*?"d2" *r*!"d1" *s*?"d3" ...

5.3 CONCURRENT SYSTEM

A concurrent system consists of a number of communicating state machines. This section defines the behavior of the concurrent system by constructing *system state sequences* that can reflect the behavior specific to each of the component state machines and the interactive behavior among them.

Figure 5.3 illustrates a model of a concurrent system which consists of n state machines (SM1, SM2, ..., SMn) communicate to each other. Data input events and data output events between state machines are internal events of concurrent system. They cannot be observed from

outside. On the other hand, data input events and data output events that communicate to environment or the other concurrent systems are external events which are observable events.

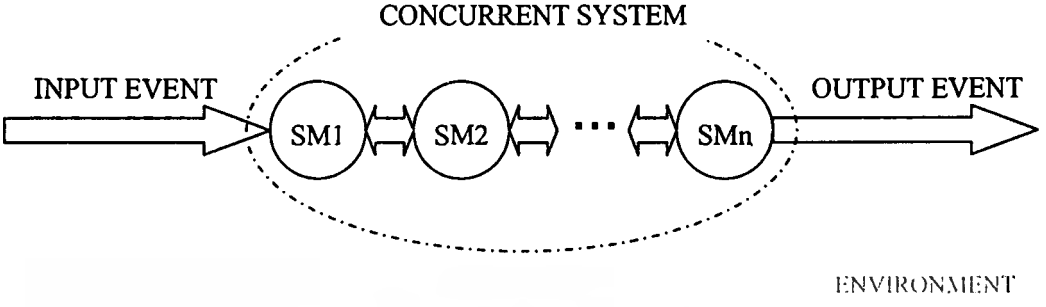


Figure 5.3 Concurrent system model

Definition 4 Let $M_1,\ldots, M_n$ be component state machines of a concurrent system. A pair $S = ((c_1,\ldots, c_i,\ldots, c_n), \mathbf{v}_1 \cup \ldots \cup \mathbf{v}_i \cup \ldots \cup \mathbf{v}_n)$ comprising a tuple of control states and a set of valuations from each component state machine $M_i (1 \leq i \leq n)$ is called a system state of the concurrent system. The projection $(c_i, \mathbf{v}_i)$ of a system state s into the state of the state machine M_i is denoted as $S^{(i)}$. An alternating sequence $\alpha = S_0 \sigma_0 S_1 \sigma_1 S_2 \sigma_2 S_3 \ldots$ of system states and events is called a *system state sequence* of the concurrent system.

Note that the assumption is $V_i \cap V_j = \phi$ for all i and $j (1 \leq i \leq n, 1 \leq j \leq n, i \neq j)$.

If a channel c is common among at least two state machines, then c is called a *shared channel.*

For a system state s and some output event $c!d$, there is $s^{(i)}$ such that $s^{(i)} \xrightarrow{c!d}$, then it is denoted the set of transitions of the other state machines such that $s^{(j)} \xrightarrow{c?d} (c_j', \mathbf{v}_j')$ for some state $(c_j', \mathbf{v}_j')$ as $Peer(s^{(i)} \xrightarrow{c!d})$. If there exist two or more transitions from one state machine in *Peer*, i.e. *nondeterministic* input transitions, then one of them is selected as an element of *Peer*. Note that in the following definition of valid system state sequences, each of possible *Peers* constructed in such a way that is accounted as *Peer*.

Definition 5 Let $M_1,\ldots, M_n$ be component state machines of a concurrent system. A system state sequence $\alpha = S_0 \sigma_0 S_1 \sigma_1 S_2 \sigma_2 S_3 \ldots$ is a *valid system state sequence* of the concurrent system if the following conditions are satisfied.

1. $s_0 = ((c_1,\ldots, c_i,\ldots, c_n), \mathbf{v}_1 \cup \ldots \cup \mathbf{v}_i \cup \ldots \cup \mathbf{v}_n)$ where $s_0^{(i)} \in \theta_i$ for all $i (1 \leq i \leq n)$

2. Let $s_j = ((c_1,\ldots, c_n), \mathbf{v}_1 \cup \ldots \cup \mathbf{v}_n)$ for each $j \geq 0$. Then σ_j and s_{j+1} satisfy one of the following conditions (a), (b), or (c)

   (a) $\sigma_j = \tau$

   for some i ($1 \leq i \leq$ n), shared channel c, data d and state $(c_i', \mathbf{v}_i')$ of state machine M_i,

   $s_j^{(i)} \xrightarrow{c!d} (c_i', \mathbf{v}_i')$

   $Peer(s_j^{(i)} \xrightarrow{c!d}) = \{ s_j^{(j1)} \xrightarrow{c?d} (c_{j1}', \mathbf{v}_{j1}'), \ldots, s_j^{(jm)} \xrightarrow{c?d} (c_{jm}', \mathbf{v}_{jm}')\} \neq \phi$ and

   $s_{j+1} = s_j[c_i'/c_i, \mathbf{v}_i'/\mathbf{v}_i] [c_{j1}'/c_{j1}, \mathbf{v}_{j1}'/\mathbf{v}_{j1}] \ldots [c_{jm}'/c_{jm}, \mathbf{v}_{jm}'/\mathbf{v}_{jm}]$

   (b) σ_j is an input or output event via a non-shared channel

   for some i ($1 \leq i \leq n$),

   $S_j^{(i)} \xrightarrow{\sigma_j} (c_i', \mathbf{v}_i')$ and

   $S_{j+1} = S_j[c_i'/c_i, \mathbf{v}_i'/\mathbf{v}_i]$

   (c) $\sigma_j = \tau$ (internal event within a single state machine)

   for some i ($1 \leq i \leq n$),

   $s_j^{(i)} \xrightarrow{\sigma_j} (c_i', \mathbf{v}_i')$ and

   $s_{j+1} = s_j[c_i'/c_i, \mathbf{v}_i'/\mathbf{v}_i]$

A valid system state sequence represents a sequence of the permissible behavior within the whole concurrent system.

This definition assumes that communication among state machines is *synchronous*. For a channel that is shared among two or more state machines, an output from a state machine is input to all the state machines that can accept it at that time (i.e. multicast). This type of data input/output is hidden from the environment as an internal event. A data input/output via non-shared channel is performed independently of other state machines, and it is exposed to the environment of the concurrent system. An internal event is also independent of other state machines.

Example 5.2 The permissible behavior of a concurrent system consisting of the state machine Sender, Medium and Receiver of Example 5.1 is shown in Figure 5.2 where MAX = 1. In the figure, all the valid system state sequences are represented in the form of a graph starting from the initial system state. The control states in a system state are of the form (Sender control state, Medium control state, Receiver control state), and they are accompanied with the value of the variable *no* which is necessary for the understanding of transitions. Incidentally, an input/output

event that becomes an internal event as the result of interaction among state machines is shown in the form of (channel name <data value>) that accompanies the τ label on the transition.

The most typical valid system state sequence is the one that begins with the initial system state and ends with the initial system state again. That is

$((S0, M0, R0), \{no = 0\})\ p?dt$

$((S1, M0, R0), \{no = 0\})\ \tau(s\langle dt\rangle)$

$((S0, M1, R0), \{no = 0\})\ \tau$

$((S0, M0, R0), \{no = 1\})\ \tau(r\langle dt\rangle)$

$((S0, M2, R1), \{no = 1\})\ c!dt$

$((S0, M2, R0), \{no = 1\})\ \tau$

$((S0, M0, R0), \{no = 0\})$.

By observing the events in this sequence, Sender receives data from the environment via *p* channel and sends the data to Medium via *s* channel. Then Medium sends data to Receiver via channel *r* after that Receiver will send data out to the environment via channel *c*. The possible behaviors of the data transfer system example are shown in Figure 5.4.

5.4 STATE MACHINE WITH CONSTRAINTS

This section explains development of a state machine and concurrent models that can be applied to various specification purposes by adding constraints to the notion of a basic state machine described in section 5.2.

5.4.1 TIME CONSTRAINT

The basic state machine does not have the concept of time. This section introduces the concept of a time constraint into state transitions of a state machine to deal with time dependent computations and traces. Consequently, it becomes possible to specify real-time systems, communication protocol and so on.

Assume that a time domain φ which contains the minimum element 0 representing the starting time. φ is assumed to be totally ordered and dense. For example, the non-negative real number $\Re \geq 0$ can be φ. To extend φ to $\varphi^{\infty} = \varphi \cup \{\infty\}$ where $\infty \geq t$ for all $t \in \varphi^{\infty}$.

Definition 6 A *time sequence* φ is a sequence of time represented by

$$\varphi = t_0 t_1 t_2 t_3 t_4 \ldots$$

which satisfies the following requirements

(i) $t_i \in \varphi$ for all $i \geq 0$.

(ii) $t_i \leq t_{i+1}$ for all $i \geq 0$.

(iii) if φ is an infinite sequence, there exists $i \geq 0$ such that $t_i > t$ for all $t \in \varphi$

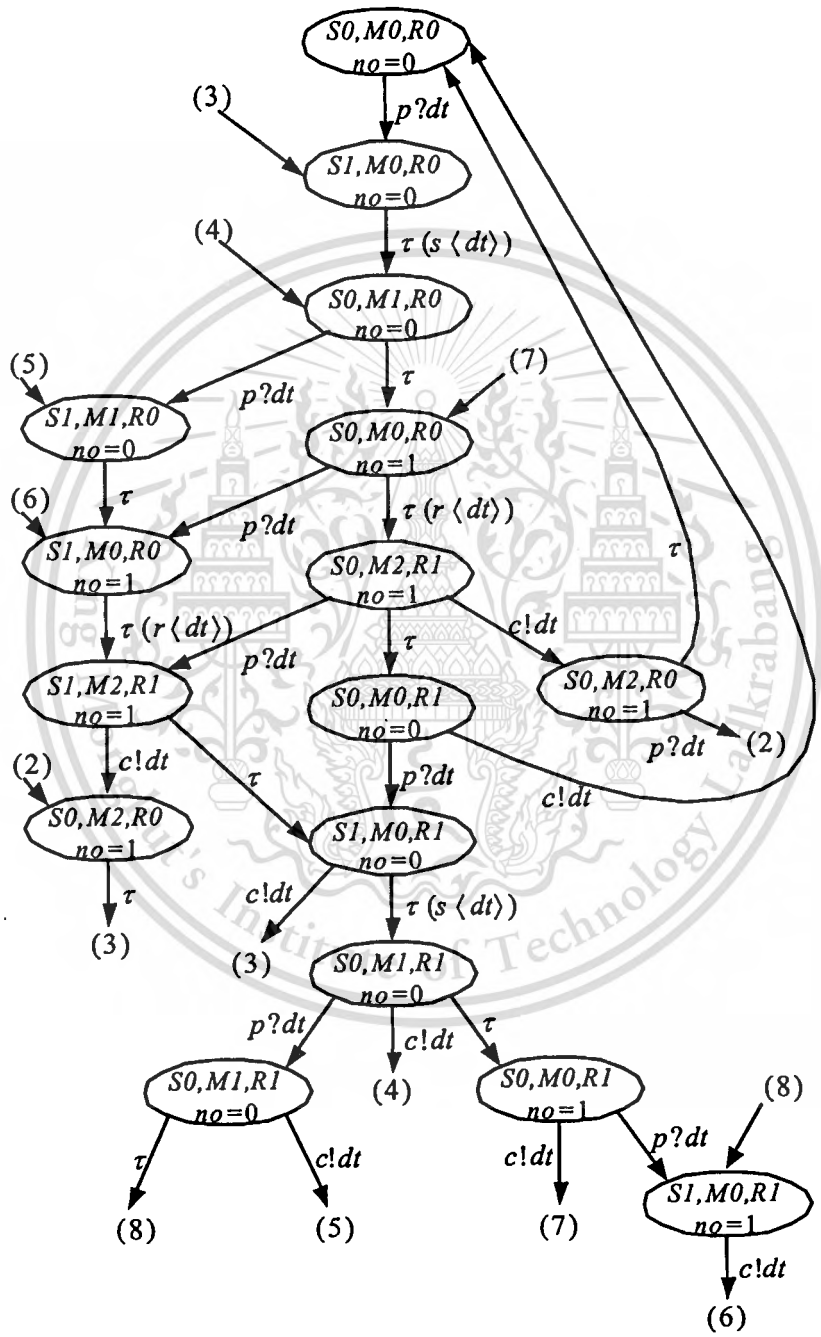


Figure 5.4 Possible behaviors of the data transfer system example

In a state machine model without time concept, the firing (enabling) condition of a transition is the current state of the state machine at the source of the transition and the corresponding event is possible to occur. In case of the time constraint, the *minimal delay l* and the *maximal delay u* until its firing are also taken into account in addition to the described firing condition. Let φ be a function that gives transitions their minimal delays and maximal delays, as a time constraint. Then we denote the minimal delay of a transition μ as $\varphi_l(\mu)$ and the maximal delay as μ as $\varphi_u(\mu)$, where $\varphi_l(\mu) \leq \varphi_u(\mu)$. The firing of each transition is assumed to be instantaneous (i.e., takes no time).

Definition 7 Let $M = \langle Q, V, \sum, T, \theta \rangle$ be a state machine, and φ ($\varphi : T \longrightarrow [\varphi \times \varphi^{\infty}]$) be a total function which gives each $\mu \in T$ a pair of $\varphi_l(\mu)$ and $\varphi_u(\mu)$, where $\varphi_l(\mu) \leq \varphi_u(\mu)$. Then a pair $< M, \varphi >$ is named to a *state machine with a time constraint* φ.

Assume that each state machine with a time constraint has a special, implicit channel *CLK* which can provide the state machine with the *current time* at any timing. This channel is not affected by the geography constraint.

Notation: When describing the minimal delay *l* and maximal delay *u* of a transition in a state diagram, it takes the following form:

$$current\_state \xrightarrow{guard \rightarrow event[l,u]} next\_state$$

The part of $[l, u]$ can be omitted if $l = 0$ and $u = \infty$.

An internal event of a state machine can occur irrespective of the environment, which is different from an input/output event. Therefore the corresponding transition must fire within its time limit. The following definition gives a time dependent friability of a transition, by considering it and delay.

Definition 8 Let *s* be a state of a state machine with a time constraint. For some transitions whose source states are s and events are internal, the minimum of their maximal delays is denoted as $mm(s)$. If the transitions whose events are internal do not exist, it is defined to be $mm(s) = \infty$. If μ is a transition from *s* and satisfies the following conditions, then μ is said to be *firable at (relative) time t*:

1. $\varphi_l(\mu) \leq t \leq \varphi_u(\mu)$
2. $t \leq mm(s)$

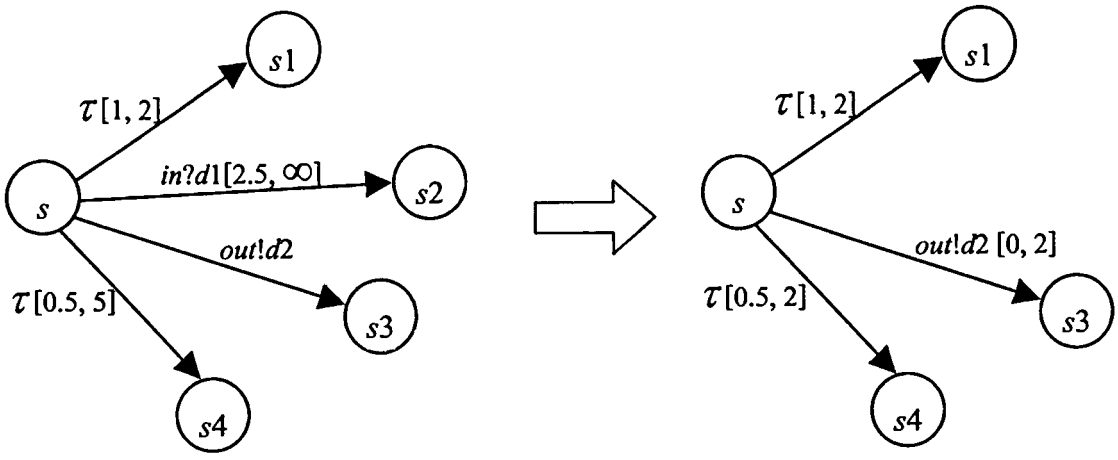


Figure 5.5 An example of firability of transitions

As shown in Figure 5.5, since *mm(s)* = 2, the firability of each transition from state *s* is described as follows.

$s \xrightarrow{\tau} s_1$ is firable at any time between 1 and 2.

$s \xrightarrow{in?d1} s_2$ is not firable at any time.

$s \xrightarrow{out!d2} s_3$ is firable at any time between 0 and 2.

$s \xrightarrow{\tau} s_4$ is firable at any time between 0.5 and 2.

Thus these transitions on the left part can be interpreted by the right part (see in Figure 5.5).

A computation of a state machine with a time constraint is modeled by using *time-stamped events* and a *time sequence*. A time-stamped event takes the form of $\langle \sigma, t \rangle$ and it represents that an event σ occurs at time *t*.

Definition 9 Let $\langle M, \varphi \rangle$ be a state machine with a time constraint. An alternating sequence of states and time-stamped events

$$\alpha = S_0 \langle \sigma_0, t_0 \rangle S_1 \langle \sigma_1, t_1 \rangle S_2 \langle \sigma_2, t_2 \rangle \ldots$$

is a *computation* of $\langle M, \varphi \rangle$ if α satisfies the following conditions:

1. $S_0 \sigma_0 S_1 \sigma_1 S_2 \sigma_2 \ldots$ is a computation of *M*.
2. $t_0 t_1 t_2 \ldots$ is a time sequence.
3. $s_0 \xrightarrow{\sigma_0} s_1$ is firable at t_0. And for all $i \geq 1$, $s_i \xrightarrow{\sigma_i} s_{i+1}$ is firable at $t_i - t_{i-1}$.

The set of all computations of $\langle M, \varphi \rangle$ is denoted as $Comp\langle M, \varphi \rangle$.

Definition 10 Let $\alpha = S_0 \langle \sigma_0, t_0 \rangle S_1 \langle \sigma_1, t_1 \rangle S_2 \langle \sigma_2, t_2 \rangle \ldots$ be a computation of a state machine with time constraint $\langle M, \varphi \rangle$. The sequence obtained by removing all states and all time-

stamped τ-events from α is said to be a *trace* of $\langle M, \varphi \rangle$. The set of all traces of $\langle M, \varphi \rangle$ is denoted as *Trace* ($\langle M, \varphi \rangle$).

5.4.2 GEOGRAPHY CONSTRAINT

Concurrent system can be modeled interactions using the basic state machines together with constraints. A communication system that concerns to mobility needs to have a geography constraint.

5.4.2.1 FIELD

The concept of a field [9, 10] is used for representing locality of communication among state machines. Assume that L is a finite set of *locations* where state machines may exist. The locations are represented by natural number (1, 2, ...). The set of all channels is denoted as C. The communication among the state machines is restricted to limited ranges and it is regarded that the ranges are different every channel.

Definition 11 A *field* F where state machines may exist is defined as follows.

$$F = \{(c, COM(c)) \mid c \in C, COM(c) \subseteq 2^L\}$$

For some c ($c \in C$) and P ($P \subseteq L$), $P \in COM(c)$ means that the state machines on the locations belonging to P can interact with each other via a channel c. On the other hand, the state machines on the locations not belonging to P cannot interact to each other via channel c. An element of $COM(c)$ is a *range* of c.

Example 5.3 Given $L = \{1, 2, 3, 4\}$ and $C = \{c_1, c_2\}$, an example of a field F is

$$F = \{(c_1, \{\{1, 2, 3\}\}), (c_2, \{\{1, 2\}, \{3, 4\}\})\}$$

Figure 5.6 shows an example in which a state machine $M1$ is on a location 1, $M2$ on location 2 and $M3$ on location 3. In this situation, all the state machines can interact to each other via a channel c_1. $M1$ and $M2$ can interact via channel c_2. However, $M3$ cannot interact to $M1$ or $M2$ via channel c_2 because $M3$ is not be in range of channel c_2 same as $M1$ and $M2$.

Assume the following properties are for a field.

1. If $COM(c)$ does not contain P such that $P \subseteq L$ and $l \in P$, then the state machines on the same location l cannot interact to each other via a channel c.

2. If $P, P' \in COM(c)$ ($P \neq P'$), then P and P' must be disjoint. Otherwise, the ranges of a channel c are overlapped for an l such that $l \in P \cap P'$, and thus it becomes impossible to uniquely identify which range is selected.

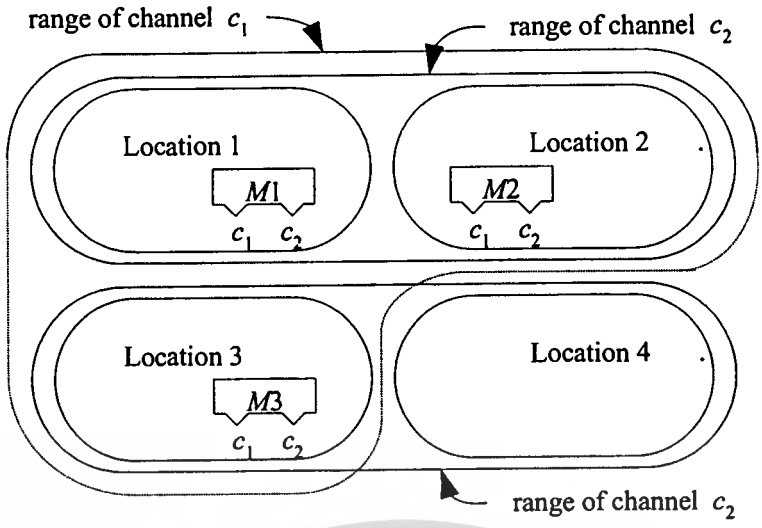


Figure 5.6 An example of a field

5.4.2.2 BEHAVIOR OF CONCURRENT SYSTEM

In case of basic state machines, the behavior of a concurrent system is defined by constructing system sequences that can reflect the behavior specific to each of the component state machines and the interactive behaviors among them.

First, the behavior of a concurrent system consisting of state machines without time constraint, but with a geography constraint is given and time constraint is introduced into it. In particular, a concurrent system with both of time and geography constraints, if the ranges of all channels cover all locations, then its behavior corresponds to the concurrent system with only time constraint.

Definition 12 Let $M_1, \ldots, M_n$ be component state machines of a concurrent system, and F be a field. Then, $\langle \{M_1, \ldots, M_n\}, F \rangle$ is called a *concurrent system consisting of state machines* $M_1, \ldots, M_n$ with a geography constraint F.

Definition 13 Let $\langle \{M_1, \ldots, M_n\}, F \rangle$ be a concurrent system with a geography constraint, and l_i be the location where a state machine M_i exists. A system state sequence $\alpha = S_0 \sigma_0 S_1 \sigma_1 S_2 \ldots$ is a *valid system state sequence* of $\langle \{M_1, \ldots, M_n\}, F \rangle$ if the following conditions are satisfied.

1. $s_0 = ((c_1, \ldots, c_i, \ldots, c_n), \mathbf{v}_1 \cup \ldots \cup \mathbf{v}_i \cup \ldots \cup \mathbf{v}_n)$ where $s_0^{(i)} \in \theta_i$ for all i $(1 \leq i \leq n)$
2. Let $s_j = ((c_1, \ldots, c_n), \mathbf{v}_1 \cup \ldots \cup \mathbf{v}_n)$ for all $j \geq 0$. Then σ_j and s_{j+1} satisfy one of the following conditions (a), (b) and (c).

   (a) $\sigma_j = \tau$

   For some i $(1 \leq i \leq n)$, shared channel c, data d and state $(c_i', \mathbf{v}_i')$ of state machine M_i,

$s_j^{(i)} \xrightarrow{c!d} (c_i', \mathbf{v}_i')$

$S = \{ s_j^{(j1)} \xrightarrow{c?d} (c_{j1}', \mathbf{v}_{j1}'), \ldots, s_j^{(jm)} \xrightarrow{c?d} (c_{jm}', \mathbf{v}_{jm}') \} \subseteq Peer(s_j^{(i)} \xrightarrow{c!d})$ and

$s_{j+1} = s_j[c_i'/c_i, \mathbf{v}_i'/\mathbf{v}_i] [c_{j1}'/c_{j1}, \mathbf{v}_{j1}'/\mathbf{v}_{j1}] \ldots [c_{jm}'/c_{jm}, \mathbf{v}_{jm}'/\mathbf{v}_{jm}]$

where l_i is in a range of c in F, and S is the largest non-empty subset of $Peer(s_j^{(i)} \xrightarrow{c!d})$ such that $l_{j1}, \ldots, l_{jm}$ are in r.

(b) σ_j is an input or output event via a non-shared channel c.

For some i $(1 \leq i \leq n)$,

$s_j^{(i)} \xrightarrow{\sigma_j} (c_i', \mathbf{v}_i')$ and

$s_{j+1} = s_j[c_i'/c_i, \mathbf{v}_i'/\mathbf{v}_i]$ where l_i is in a range of c in F.

(c) $\sigma_j = \tau$ (internal event within a single state machine)

For some i $(1 \leq i \leq n)$,

$s_j^{(i)} \xrightarrow{\sigma_j} (c_i', \mathbf{v}_i')$ and

$s_{j+1} = s_j[c_i'/c_i, \mathbf{v}_i'/\mathbf{v}_i]$

Example 5.4 Assume that $L = \{1, 2, 3\}$, $C = \{c, d\}$, $F = \{(c, \{\{1, 3\}, \{2\}\}), (d, \{\{1, 2, 3\}\})\}$, state machine specifications are given in Figure 5.7 and each state machine M_i $(1 \leq i \leq 3)$ is on location i. Then, all the valid system state sequences of this concurrent system are as follows.

$\alpha 1 =$ $((A0, B0, C0), \{y = .., x = ..\})$ $\tau((A1, B0, C0), \{y = .., x = ..\})$ $\tau(c\langle 3\rangle)$ $((A2, B0, C1), \{y = .., x = 3\})$ $d!10$ $((A2, B1, C1), \{y = .., x = 3\})$

$\alpha 2 =$ $((A0, B0, C0), \{y = .., x = ..\})$ $\tau((A1, B0, C0), \{y = .., x = ..\})$ $d!10$ $((A1, B1, C0), \{y = .., x = ..\})$ $\tau(c\langle 3\rangle)$ $((A2, B1, C1), \{y = .., x = 3\})$

$\alpha 3 =$ $((A0, B0, C0), \{y = .., x = ..\})$ $d!10$ $((A0, B1, C0), \{y = .., x = ..\})$ $\tau((A1, B1, C0), \{y = .., x = ..\})$ $\tau(c\langle 3\rangle)$ $((A2, B1, C1), \{y = .., x = 3\})$

Note that it is impossible to communicate between state machines $M1$ and $M2$ because the range of channel c in $M1$ is different from the range of channel c in $M2$.

Example 5.5 Consider the Contract Net Protocol [23] as an example of a concurrent system with time and geography constraints. The Contract Net Protocol is a task assignment protocol among agents in a multi-agent system. With respect to a task execution, an agent called a Manager announces it to agents call Contractors. This is called a TA (Task Announcement). A Contractor that received the TA decides either to accept it or not by evaluating it, and bids (Bid) on it if accepted. The Manager receives Bids from Contractors for some period. When the bid reception period is expired, the Manager awards (Award) to a Contractor whose Bid was evaluated as the

highest. The awarded Contractor executes the task and then reports (Report) the result to the Manager. A Contractor itself can work as a Manager. In this case, a hierarchical structure of Manager-Contractor is formed.

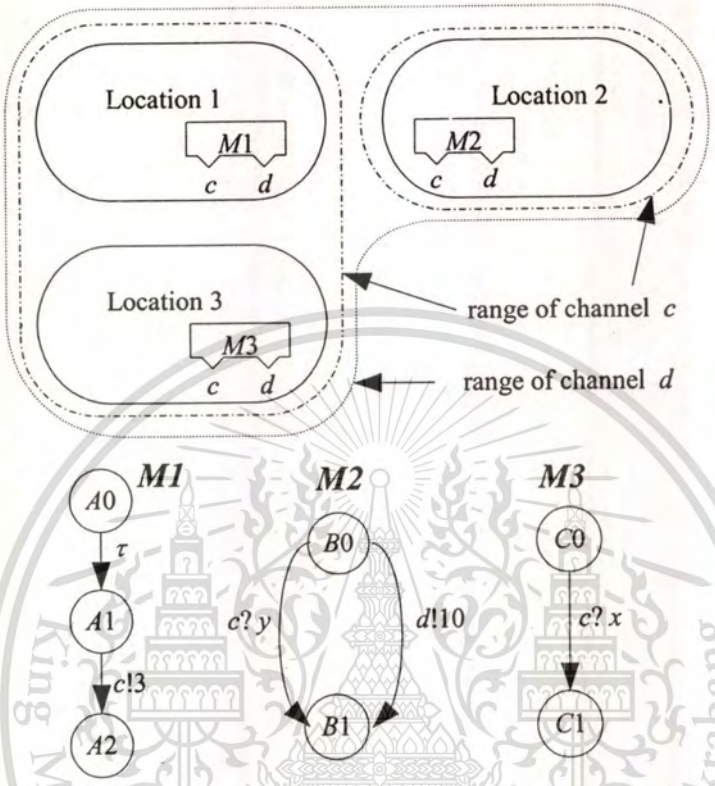


Figure 5.7 An example of a concurrent system with geography constraint

Figure 5.8 (a) shows an example of such a hierarchical structure. The most top Manager is called Controller. A manager at the middle layer also plays the role of a Controller. Each box in the figure represents a state machine to be specified. A small back circle and a line between them represent a channel among state machines and their association, respectively. The areas embraced by the dot line represent the field (the ranges of the channels) prescribed in this specification. It can be seen that a channel c is reused every Manager, by this geography constraint.

Figure 5.8 (b) depicts the specification of each state machine in figure 5.8 (a). The specification of the Controller is a simplified version of a Manager, and it is omitted here.

The specification of each Manager_i ($1 \leq i \leq 2$) is shown in the left side of Figure 5.8 (b). The initial value of a variable now is 0. The messages between each Manager_i and each related Contractor_{ij} ($j = 1..y$ if $i = 1$; $j = 1..z$ if $i = 2$) are exchanged via a channel c, and the messages

between each Manager$_i$ and the Controller are exchanged via a channel m_i. Each message consists of the message type and parameters. A parameter is represented by enclosing it in parentheses. A thing enclosed in double quotation marks is a constant, and the other is a variable. In the specification of Manager$_i$, the transition from state $M2$ to state $M3$ sets the time of TA. In state $M3$, the current time is continuously read which is used in the guard of the transition from state $M3$ to state $M5$. This guard being true means the bidding period (BP) is expired, and in this case the internal transition is immediately fired for the subsequent Award notification. A bid is acceptable within BP. The internal processing BidProc decides which Contractor is awarded, and it sets the identifier of the awarded Contractor to a variable ACID.

The specification of each Contractor$_{ij}$ is shown in the right side of Figure 5.7(b). The distinguished transition is the one from state C3 to state C4. If this Contractor was awarded for a TA, its processing TaskProc takes time from P_{min} to P_{max}. The transition from state C1 to state C0 corresponds to the case where this Contractor does not bid on the TA.

The valid system state sequences of this concurrent system are derived from the above definition, depending on the geography constraint and the specifications of the state machines. One of typical behavior is illustrated next. First, the Controller sends a TA to a Manager, for a task to be executed. The Manager sends it again to the related Contractors, then receives Bids from some of them, and then sends an Award to a Contractor which was selected as appropriate. The awarded Contractor executes the task and sends back a Report to the Manager. Finally the Manager sends it to the Controller. Also, the following behavior is reasoned. There is a case where some Contractor cannot bid on within the bidding period after a TA. This originates in the transition from state C1 to state C2 of a Contractor in which the transmission of a Bid can be performed at any time.

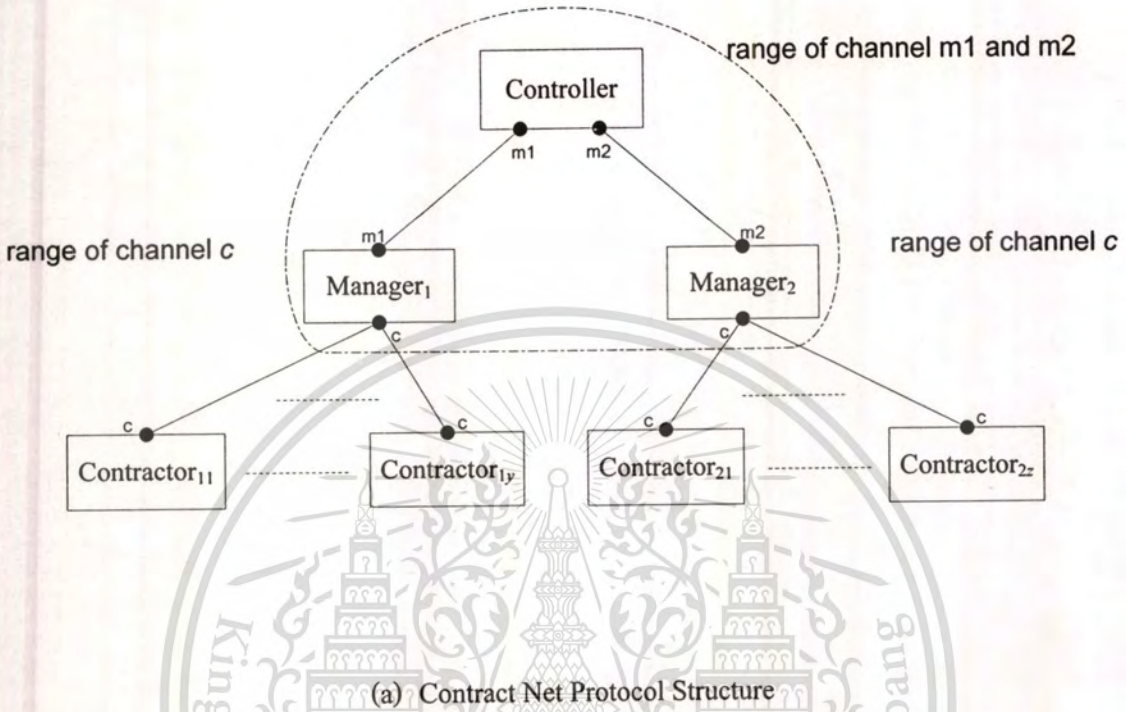
5.4.3 MOBILITY CONSTRAINT

Mobility constraint is a limitation that effect to a state machine model because of moving of any state machine in each location. This means that mobility constraint is limited by geography constraint. The existence of state machine in each location, state machine's position, is used as reference for all state machines in a communication system. The existence of state machine is defined.

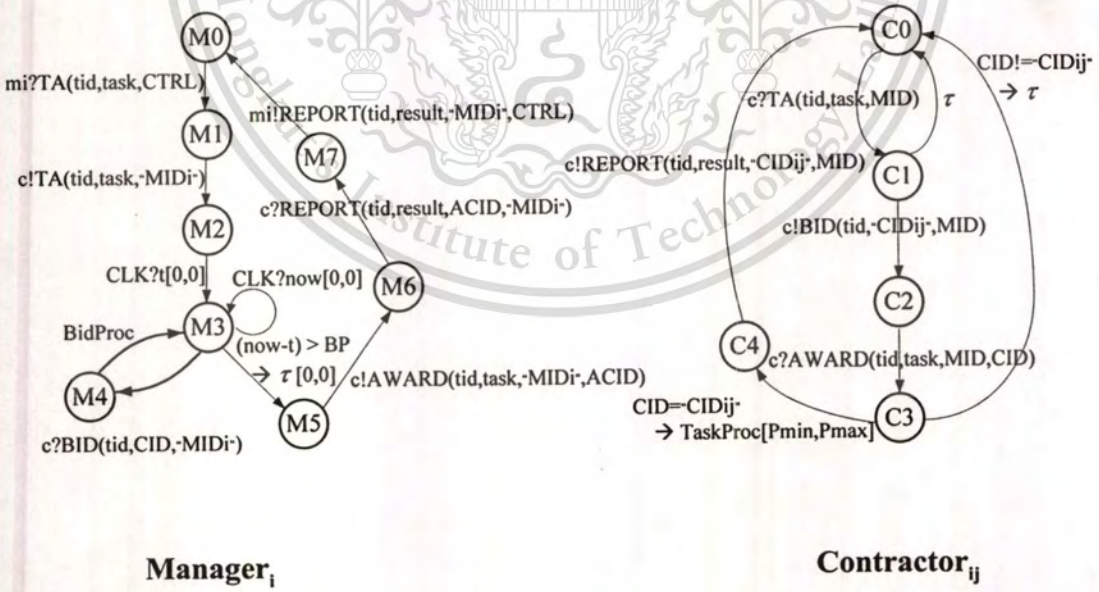
Definition 14 Let M is a set of all state machines, L is a set of locations and P is a set of state machines' position. Then P is defined as follow.

$$P = \{(l, N \mid l \in L, N \subseteq M\}$$

Because locations are not specified to overlap, so each state machine can stay in only a location at moment.



(a) Contract Net Protocol Structure



(b) Contract Net Protocol Specifications in State Machine Model

Figure 5.8 The contract net protocol

Example 5.6 Assume that $L = \{1, 2, 3, 4\}$, $C = \{c_1, c_2\}$ and $F = \{(c_1, \{\{1, 2, 3\}\}), (c_2, \{\{1, 2\}, \{3, 4\}\})\}$, state machine specifications are given in Figure 5.8.

From Figure 5.8, the state machines' position is:

$$P = \{(1, \{M1\}), (2, \{M2\}), (3, \{M3\}), (4, \phi)\}$$

From Figure 5.9, state machine $M2$ moves from location 2 to location 1. The state machines' position is:

$$P' = \{(1, \{M1, M2\}), (2, \phi), (3, \{M3\}), (4, \phi)\}$$

For mobility constraint, the definition is specified to the type of mobile communication system which supposes that each state machine can move freely. The limitation of communication depends on geography constraint in each location that state machines stay. The state machines' position definition can be used to specify the positions and moving behaviors of state machines. The definition can be applied for mobile communication system design.

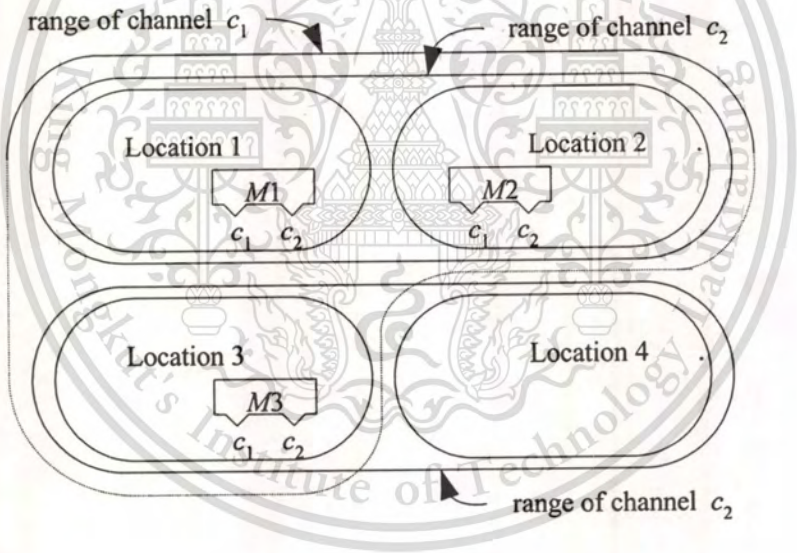


Figure 5.9 State machines' position in a communication system

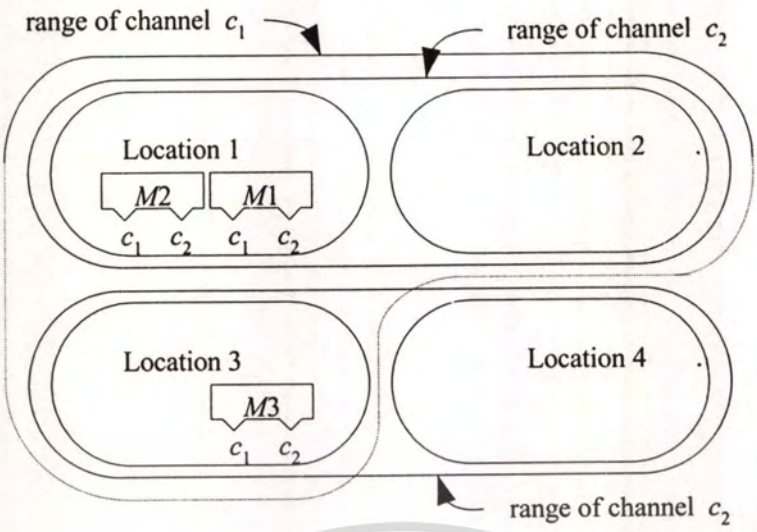


Figure 5.10 Moving of state machine *M*2

5.5 SUPPORT TOOL

This section, a simulator was developed based on the specification method using state machine model with constraints, time, geography and mobility. It is helpful for communication system design and analysis. The simulator, *"MGraphGen"*, can be used to design and specify specification of communication systems. The *MGraphGen* consists of two tools, an analysis tool and a simulation tool. They are effective to specification and simulation of communication system.

5.5.1 FEATURE OF THE *MGraphGen*

The *MGraphGen* needs specifications in text file format as an input and it shows the graphical result as an output. The advantages are as follows:

(i) To investigate specifications of communication system

(ii) To display specification in graphic diagram which is ease to understand and suit for use

(iii) Analysis of the computation and trace of user selected path in state machine model.

(iv) Can select order of working and transition path of state machine up to request

(v) Can find the shortest path that waste least time by define beginning state and terminating state

(vi) Investigation of behaviors of a concurrent system by using simulation tool

(vii) Can simulate system working to specify works and behaviors in the communication system

The simulator will work after get a specification file which is two types, state machine specification file and system specification file. File types are described in detail. The working diagram of *MGraphGen* is shown in Figure 5.11.

1. State machine specification file contains specifications of state machine with or without time constraint. An example of this file is as in Example 5.7.

2. System specification file contains specifications of concurrent state machines with constraints such as time, geography constraint and so on. However, a system specification file also refers to all state machine specification files that need to run in the system. An example of this file is as in Example 5.8.

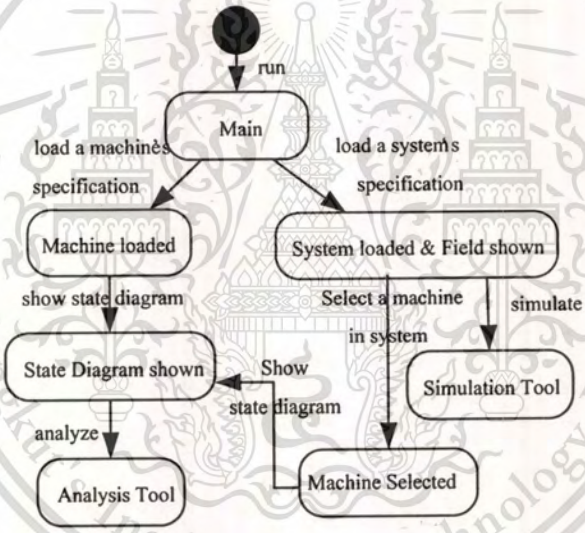


Figure 5.11 The working diagram of *MGraphGen*

When user opens *MGraphGen*, it will show a main window which has components as described and shown in Figure 5.12.

1. Menu Tab is used to call commands of tools.

2. Tool Box has many commands that always be used.

3. Display Window is an area for design and display a system.

4. Status Tab shows a status of program.

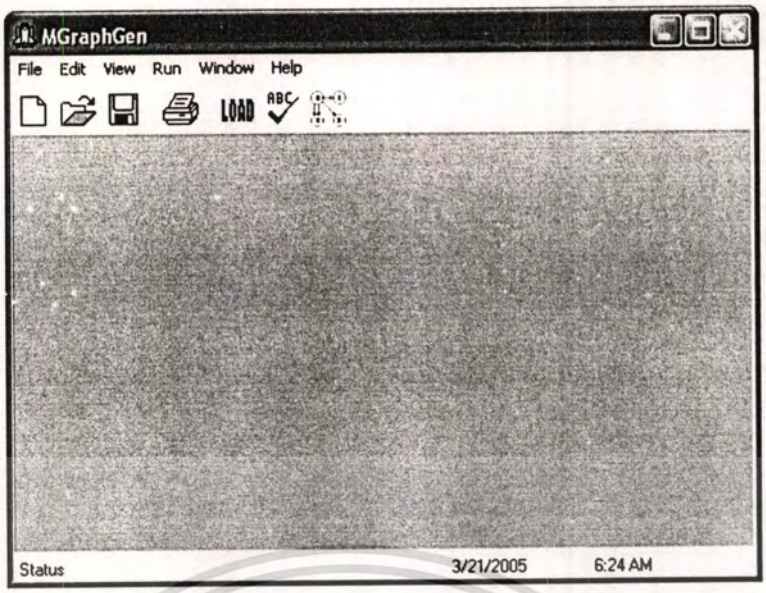


Figure 5.12 The main window of *MGraphGen*

Example 5.7 shows a state machine specification file of medium as in Example 5.1.

```
#MAX:int=1; i:int=0; o:int=0; no:int=0; data[]:int
1{no<MAX-->s?data[i]}2
2{tau(no=no+1;i=(i+1)%MAX)}1
1{no>0-->r!data[o]}3
3{tau(no=no-1;o=(o+1)%MAX)}1
```

From definition 2.1, state machine specification file consists of:

Q = { 1, 2, 3 }

V = { no, i, o, data[] }

$\sum = \sum^{in} + \sum^{out} + \tau$

Where $\sum^{in}$ = { s?data[i] }

$\sum^{out}$ = { r!data[o] }

τ = { tau(no=no+1;i=(i+1)%MAX), tau(no=no-1;o=(o+1)%MAX) }

T = { (1, {no, i, o, data[]}) *s?data[i]* (2, {no, i, o, data[]}),
(1, {no, i, o, data[]}) *r!data[o]* (3, {no, i, o, data[]}),
(2, {no, i, o, data[]}) tau(no=no+1;i=(i+1)%MAX) (1, {no, i, o, data[]}),
(3, {no, i, o, data[]}) tau(no=no-1;o=(o+1)%MAX) (1, {no, i, o, data[]}) }

θ = { (1, no, i, o, data[]) }

From Example 5.6, specifications are explained as follows.

The first line is a variable declaration. Let MAX, i, o, and no are integer and start values are 1, 0, 0, and 0 respectively. The data[] is an integer array.

The second line is a state transition from state 1 to 2 with an input event via channel s. A data is put in data[] at position i with a no < MAX condition.

The third line means a state transition from state 2 to 1 with an internal event and no, and i are changed.

The fourth line is a state transition from state 1 to 3 with an output event which data in data[] in position 0 is sent via channel r to environment with a no < MAX condition.

The last line is a state transition from state 3 to 1 with an internal event that no and i are changed.

Example 5.8 shows a system specification file of Contract Net System as in Example 5.5.

```
[Location]
1=Controller.txt
2=manager1.txt,contractor11.txt,contractor12.txt
3=manager2.txt,contractor21.txt,contractor22.txt
[SharedChannel]
m1={1,2}
m2={1,3}
c={2},{3}
```

From Definition 11 and 14, a set of service locations L, a set of channel C, range of channel F, and a set of state machine positions P can be shown as follows.

L = {1, 2, 3}

C = {m1, m2, c}

F = {(m1, {1, 2}), (m2, {1, 3}), (c, {{2}, {3}})}

P = {(1, {Controller}), (2, {manager1, contractor11, contractor12}),
(3, {manager2, contractor21, contractor22})}

To specify state machines in each service location from line 2, 3, and 4, a user has to specify all state machines in a system into specification files for reference and internal execution. Thus, state machine specification files should be created and input to *MGraphGen* before creating

a system specification file. After a specification file is input to the program, it will be checked syntax and shown as a state diagram.

5.5.2 ANALYSIS TOOL

This tool deals with the state machine specification and shows the state machine model in form of state diagram as a result. From this step, user can investigate paths or processes of the specified state machine through the state diagram by selecting the appropriate modes that there are two modes, the time shortest path or minimal delay path mode and the state sequence mode. However, the analysis tool specifications are as follows:

(i) A finite state machine could have maximum state to 20 states.

(ii) A state could not have transition more than 5 transitions.

(iii) The maximal delay could not be over 62,000 time units.

5.5.2.1 THE MINIMAL DELAY PATH

After *MGraphGen* loaded a state machine specification file, it will show a state diagram. A user can select the minimal delay path by choosing source and destination states. Then the analysis tool will show a transition path, which takes least time, and total execution time.

5.5.2.2 THE STATE SEQUENCE

When *MGraphGen* generated a state diagram, a user must select a state transition path. This tool will show a dialog box that a user can input delay time for every selected event. Then, *MGraphGen* will display a state sequence or a transition sequence which a user can analyze and verify.

Example 5.9 shows a state machine analysis using *MGraphGen* step by step.

1. Open a state machine specification file from the menu tab using command File -> Open then the file will show in the main window. A user can edit or save this file if there is any change (See in Figure 5.13).
2. Load a state machine specification file into MGraphGen using command Run -> Load (See in Figure 5.14).
3. Display status of a state machine using Run -> Graph (See in Figure 5.15).
4. To find the time shortest path, a user has to input source and destination states in From box and To box. Then, click the Search button after that the analysis tool will display the minimal delay path and total execution time in a dialog box (See in Figure 5.16).

5. When a user needs to specify any event, a user has to input minimal and maximal delay time in dialog boxes (See in Figure 5.17). If a user clicks Computation button, the analysis tool will show a state sequence which is a sequence of state transition including actions (See in Figure 5.18). But a user clicks Trace button, the analysis tool will show a trace which is a sequence of observable actions (See in Figure 5.19).

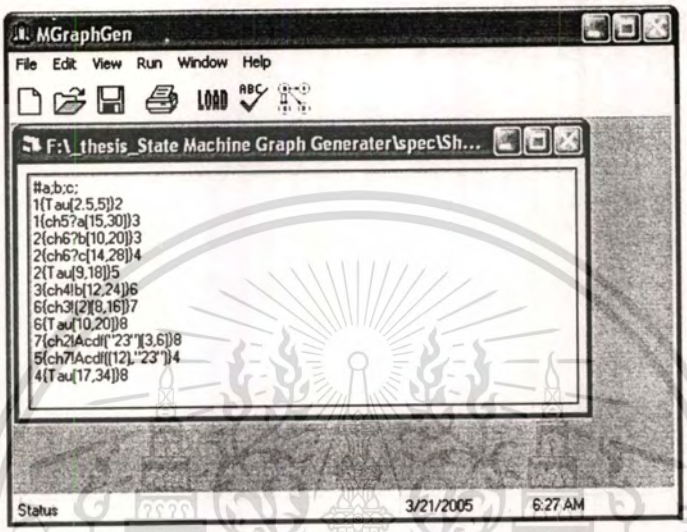


Figure 5.13 An example of state machine specification file

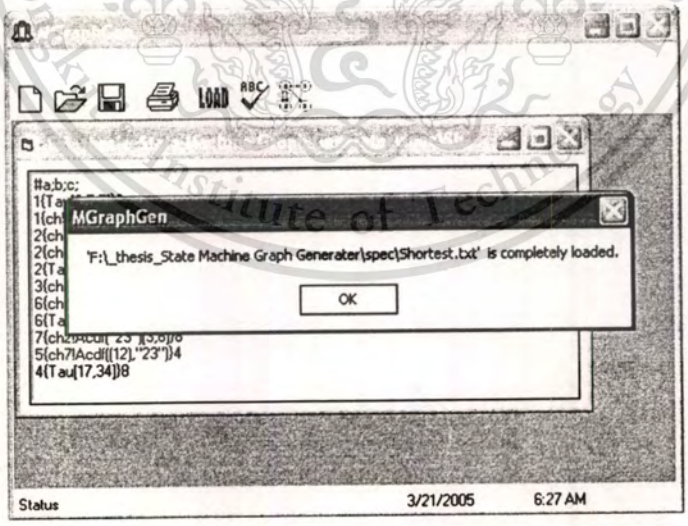


Figure 5.14 A dialog box after a state machine specification file is loaded

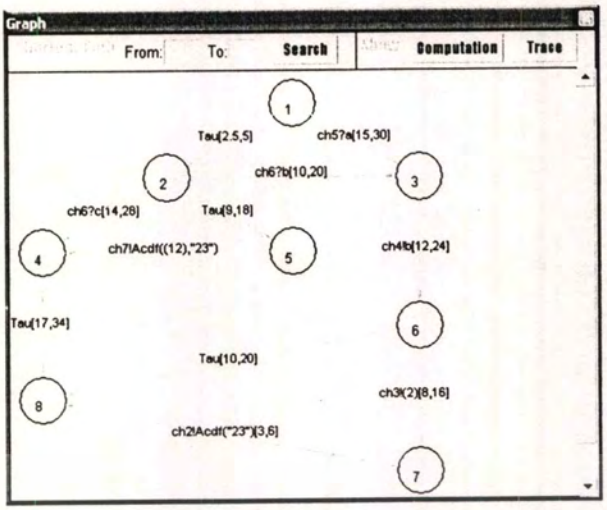


Figure 5.15 A state diagram

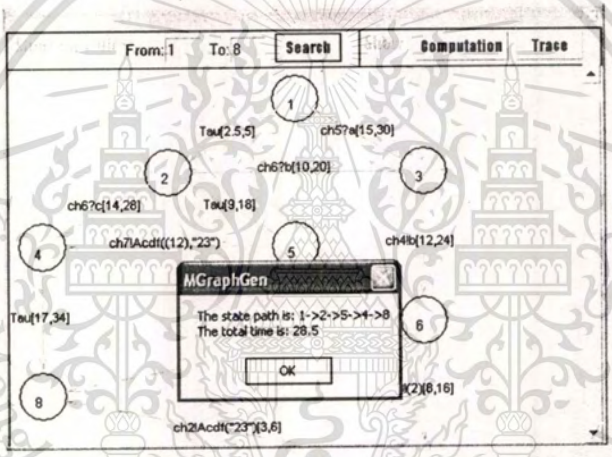


Figure 5.16 The time shortest path from state 1 to 8

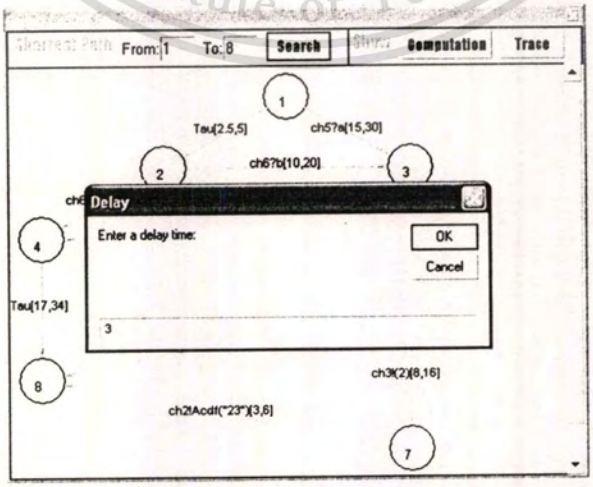


Figure 5.17 A delay time box

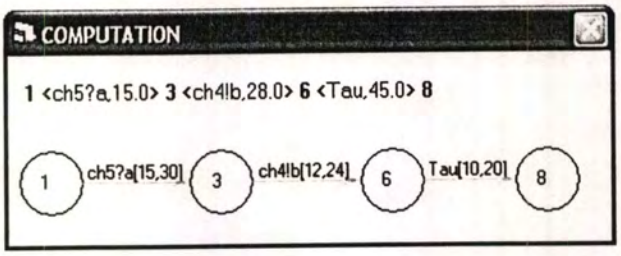


Figure 5.18 A state sequence or computation

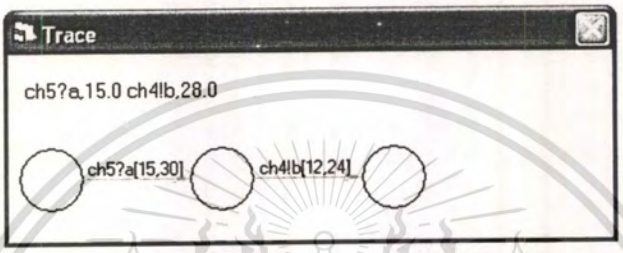


Figure 5.19 A sequence of observable actions or trace

5.5.3 SIMULATION TOOL

To simulate the concurrent system, this tool needs a system specification file as input file and state machine specification files as reference files. The result of this tool can help designers to easily understand the behaviors of the concurrent system. The features of simulation tool are as follows:

(i) Automatic simulation followed by input specification

(ii) Simulation command consists of start, pause, and stop

(iii) Manually input data from environment

(iv) Display of result as a system state sequence

(v) Display of result as a message sequence chart.

The simulation tool specifications are as follows:

(i) A concurrent system could not have more than 10 finite state machines.

(ii) The maximum of locations in a concurrent system is 18 locations.

(iii) The range of a channel could not cover more than 6 locations.

The simulation tool has three effective functions to support to design a concurrent system as follows.

5.5.3.1 FIELD DISPLAY

After a user open a system specification file, the simulator will display a field window to show range of shared channels of a concurrent system in a location. Moreover, a user can click on a state machine to display the state diagram too. Figure 5.20 depicts an example of Field Display when a user selects to view state machines in location 2.

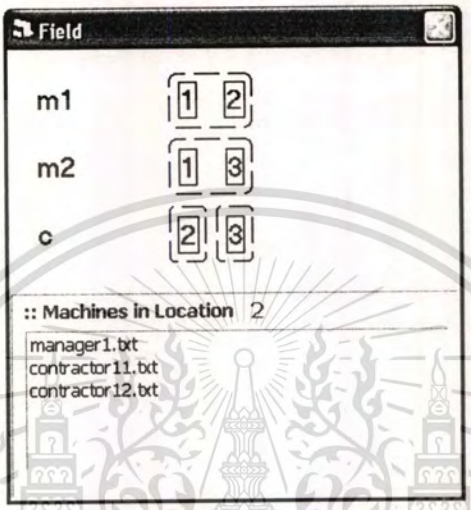


Figure 5.20 A Field Display when a user selects to view state machines in location 2

5.5.3.2 SIMULATION CONTROLLER

The Simulation Controller will control a working system simulation and display transition sequence of system. Figure 5.21 depicts a GUI (Graphic User Interface) of Simulation Controller and its components are described.

1. **Speed Control Tab** is used to control the speed of simulation. If a user needs to increase the simulation speed up or down, a user just slides a gauge on the tab (See number 1 in Figure 5.21).
2. **Simulation Control Buttons** is used to control a simulation via command buttons which consists of Start, Pause, and Stop buttons (See number 2 in Figure 5.21).
3. **Input Box** is used for input data into a system via an Environment Data Box after a user click the load button (See number 3 in Figure 5.21).
4. **Simulation Display** shows a transition sequence of system simulation which the last line is the present status of the system (See number 4 in Figure 5.21).

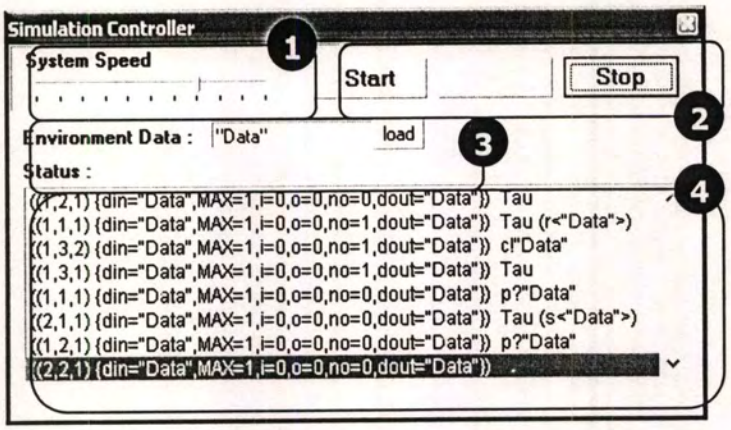


Figure 5.21 A GUI (Graphic User Interface) of Simulation Controller

5.5.3.3 MESSAGE SEQUENCE CHART

Message Sequence Chart is a window that shows a communication sequence of state machines. The sequence is very effective and useful to verify behaviors of concurrent system that consists of state machines. An example of a Message Sequence Chart of Example 5.1 when time passed 6 units is shown in Figure 5.22.

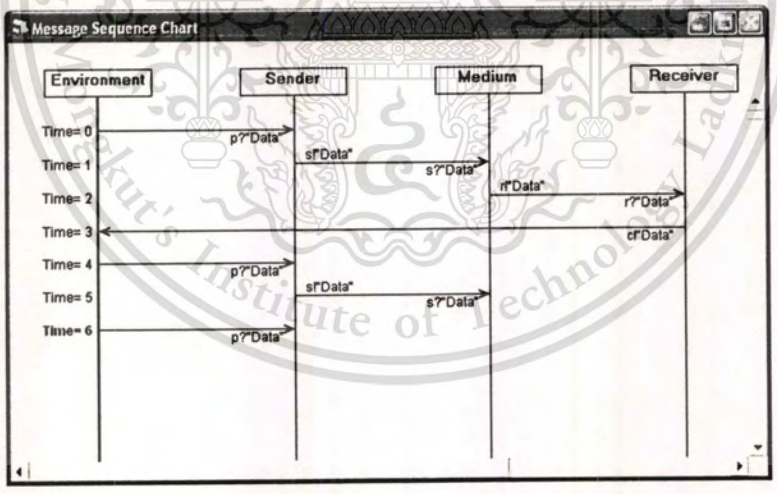


Figure 5.22 An example of a Message Sequence Chart of Example 5.1 when time passed 6 units

CHAPTER 6

CONCLUSION

The thesis has proposed two methods for protocol design. The first method is a composition method of service and protocol specifications which are written in LOTOS language based on weak bisimulation concept. LOTOS is a Formal Description Technique (FDT) which is developed for a communication protocol specification. The polling algorithm is also applied to the composition method for maintaining order of actions of service specifications and protocol specifications. An asynchronous model is used to model practical communication networks.

The protocol specifications are derived from decomposition of service specifications. Then, service specifications and protocol specifications will be composed individually and simultaneously. While the composed service specifications and the composed protocol specifications have to maintain equivalence between them based on weak bisimulation equivalence. The LOTOS operators, which are applied to the composition method, are enabling, parallel, choice, and disabling. Moreover, a support system based on the composition method is developed for verification of specifications. By applying the composition method, a good and effective protocol design can be achieved. The composition method is useful for specifying and verifying a correctness of specifications in the early stage of a protocol design. However, service specifications and protocol specifications are in a text format. It is complex to understand and verify the specifications. Therefore, a method, which is powerful and easy to understand, is proposed.

The other method is a specification method based on a state machine model. In this method, a communication system is defined as a collection of state machines. Behaviors of the communication system are given in terms of actions or internal actions among the state machines. Three constraints, which are time, geography, and mobility, are introduced into the state machine model. The state machine with constraints can be described various communication systems. Adding these constraints into the state machine model can help a protocol designer to apply this method to a specification of communication system.

In addition, *MGraphGen*, a supporting tool based on the specification method, is developed for design and analysis of communication systems. The state machines, which represent a communication system, will be shown as a state diagram that is easy to verify and analyze.

MGraphGen also provides a simulation tool that a user can verify and investigate specifications of the communication systems.

Both of the proposed methods could be adopt to use in the design of communication protocols effectively. They are very helpful and useful for communication system design. Furthermore, more constraints can be added to the second method which makes it suitable to be considered as a communication system.

REFERENCE

[1] ISO, Information Processing Systems, "Open Systems Interconnection-LOTOS-A Formal Description Technique based on the Temporal Ordering of Observational Behavior," ISO 8807, 1989.

[2] R.Langerak, "Decomposition of Functionality: A Correctness Preserving LOTOS Transformation," in Protocol Specification, Testing and Verification, pp.203-218, 1990.

[3] B.B.Bista, K.Takahashi and N.Shiratori, "A Compositional Approach for Constructing Communication Services and Protocols," IEICE Trans. Fundamentals, Vol.E82-A, No.11, pp.2546-2557, 1999.

[4] G.Bochmann and R.Gotzhein, "Deriving Protocol Specification from Service Specification," Proc. SIGCOMM'86, Vol.14, pp.144-156, 1986.

[5] C.Kant, T.Higashino and G.V.Bochmann, "Deriving Protocol Specifications from Service Specifications Written in LOTOS," Distributed Computing, Vol.10, No.1, pp.29-47, 1996.

[6] A.Khoumsi and K.Saleh, "Two Formal Methods for the Synthesis of Discrete Event Systems," Computer Networks and ISDN Systems, Vol.29, No.7, pp.759-780, 1997.

[7] H.-A.Lin and C.L.Tarng, "An Improved Method for Constructing Multiphase Communications Protocols," IEEE Transactions on Computers, Vol.42, No.1, pp.15-26, 1993.

[8] M.Nakamura, Y.Kakuda and T.Kikuno, "On Constructing Communication Protocols from Component-Based Service Specifications," Computer Communications, Vol.19, pp.1200-1215, 1996.

[9] ISO 8571-3, ISO – File service definition: File Transfer, Access and Management – Part 3, 1989.

[10] ISO 8571-4, ISO – File protocol definition: File Transfer, Access and Management – Part 4, 1989.

[11] R. Milner, "Communication and Concurrency," Prentice-Hall, 1989.

[12] C. A. R. Hoare, "Communication Sequential Process," Prentice-Hall, 1985.

[13] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. S. Trauring and M. Trakhtenbrot, "Statemate: a Working Environment for the Development of Complex Reactive System," IEEE Trans. Software Eng., Vol.16, No.4, pp.403-414, 1990.

[14] D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," ACM Trans. Software Engineering and Methodology, Vol.5, No.4, pp.293-333, Oct. 1996.

[15] E. Battiston, F. D. Cindio and G. Mauli, "Modular Algebraic Nets to Specify Concurrent Syatems," IEEE Trans. Software Eng., Vol.22, No.10, pp.689-705, 1996.

[16] A. Gabrielian and M. Franklin, "Multilevel Specification of Real-Time Systems," Commun. ACM, Vol.34, No.5, pp.50-60, May. 1991.

[17] F. Jahanian and A. K. Mok, "Modechart: A Specification Language for Real-Time Systems," IEEE Trans. Software Eng., Vol.20, No.12, pp.933-947, Dec. 1994.

[18] A. Coen-Porisini, C. Ghezzi and R. A. Kemmerer, "Specification of Realtime Systems Using ASTRAL," IEEE Trans. Software Eng., Vol.23, No.9, pp.572-598, Sept. 1997.

[19] S. Yamane, "Specification and Verification of Real-time Concurrent Software," (in Japanese) Trans. Information Processing Society of Japan, Vol.37, No.2, pp.188-203, Feb. 1996.

[20] R. Alur and D. Dill, "Automata for Modeling Real-Time Systems," Lecture Notes in Computer Science 443, pp.322-335, 1990.

[21] A. C. Shaw, "Communication Real-Time State Machines," IEEE Trans. Software Eng., Vol.18, No.9, pp.805-816, Sept. 1992.

[22] R. Milner, J. Parrow and D. Walker, "A Calculus of Mobile Processes, Part I and Part II," Journal of Information and Computation, Vol.100, pp.1-77, 1992.

[23] J. Davies and S. Schneider, "A Brief History of Timed CSP," Theoretical Computer Science, Vol.138, 1995.

[24] ISO/IEC, "Information Technology-Enhancements to LOTOS (E-LOTOS)," ISO/IEC15437, Aug. 2001.

[25] K. Takahashi, G. Itabashi and Y. Kato, "Specification of a Mobile System based on Finite State Model," Proc.1st International Conference on Parallel and Distributed Computing, Applications and Technologies, pp.209-214, 2000.

[26] Brinksma, E., "Information Processing System-Open Systems Interconnection-LOTOS-A Formal Description Technique based upon the Temporal Ordering of Observational Behaviour," Draft International Standard ISO8807, 1988.

[27] John Carrol and Darrel Long, "Theory of Finite Automata," Prentice-Hall, 1989.

[28] Eike Best, Raymond Devillers and Maciej Koutny, "Petri Net Algebra," Springer, 2000.

[29] Gerard J. Holzmann, "Design and Validation of Computer Protocols," Prentice-Hall, 1991.

[30] Richard Lai and Ajin Jiracheifpattana, "Communication Protocol Specification and Verification," Kluwer Academic Publishers, 1998.

[31] Tommaso Bolognesi, Jeroen van de Lagemaat and Chris Vissers, "LOTOSphere Software Development with LOTOS," Kluwer Academic Publishers, 1995.

APPENDIX A INTRODUCTION TO PETRI NET

APPENDIX A INTRODUCTION TO PETRI NET

Petri Net was originated from Dr. Carl Adam Petri's Ph.D. thesis (1962). It is a very simple concept and good for representing discrete event systems, and effective in analyzing system behaviors. It could be used to describe concurrent systems using combination principles.

A Petri Net may be viewed as a graph. Basic components of Petri Net are a set of place, a set of transition, a set of arc, and a set of token. The set of arc will be defined from input and output processes. The result arcs from both processes will force direction of token from a place to a transition and from a transition to a place. Places are represented as circles and transition as rectangles. Figure A1 illustrated examples of Petri Nets are explained as follows. A place s may be an input/output of transition t or both. If there is an arc leads from s to t, place s is an input of a transition. If there is an arc leads from t to s, place s is an output of a transition. In case of both, place s is called a side place of transition.

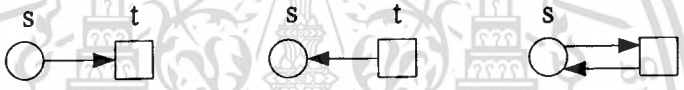


Figure A1 Examples of Petri Nets

The graph of Petri Net describes the structure of a system. The behavior of that system is defined with respect to a given marking (state) of the graph, called the initial marking. In general, a marking is a function from the set of places to the set of natural numbers, and if the marking of places s is n then s contains n tokens.

Figure A2 depicts as unmarked Petri Net, N, and two marked Petri Nets, (N, M0) and (N, M1). Note that the unmarked net is Petri Net which empty marking is assigned to each place. All three graphs shown in Figure A2 are identical, except the markings are different.

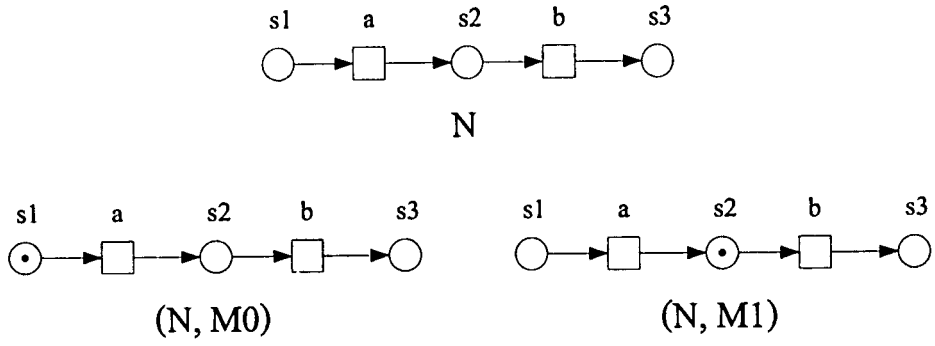


Figure A2 A Petri Net, N and the same net with marking M0, M1

Formally, Figure A1 can be described that places are s1, s2, s3 and transitions are a and b. The marking M0 can be explained as a function satisfying M0(s1) = 1, M0(s2) = 0 and M0(s3) = 0 and the marking M1 as a function satisfying M1(s1) = 0, M1(s2) = 1 and M1(s3) = 0.

APPENDIX B COMPARISON BETWEEN PETRI NET AND FINITE STATE MACHINE

APPENDIX B COMPARISON BETWEEN PETRI NET AND FINITE STATE MACHINE

In this research, the comparison of Finite State Machine model and Petri Net model for a communication system Design is to determine that which model is suitable for modeling a communication system.

Both Petri Net and Finite State Machine models can be used to describe a concurrent system. The priority, Petri Net can not be expressed. Due to the working of Petri Net depends on marking or tokens in each state. Therefore, it may have many transitions at the same time while Finite State Machine can execute only one action at a time. In addition, Petri Net is more complex to represent operations on data since data can be changed in many states at the same time. Although Petri Net and Finite State Machine are described in graphical diagrams which are easy to understand and analyze, however, Finite State Machine is less complex. Moreover, Finite State Machine concept is a step execution as Petri Net's is a dynamic execution. The comparison of functions between Petri Net and Finite State Machine is demonstrated in Table B.1.

Table B1 The comparison of functions between Petri Net and Finite State Machine

Function	Petri Net	Finite State Machine
1. Concurrency	●	●
2. Graphical	●	●
3. Data counting	-	●
4. Priority	-	●
5. Easy	-	●

Finite State Machine is flexible to model a concurrent system if a protocol designer needs to apply any constraint into the model by events while applying any constraint in Petri Net is difficult because any condition has to specify by transition path of token. Petri Net uses a token moving as a condition of transition and has to use a token to specify a current state in a system. But a transition in Finite Stat Machine model uses an event to be a condition.

APPENDIX C PUBLISHED PAPERS

Composition Method of Communication System Specifications in Asynchronous Model and Its Support System

Noppadol Maneerat[†,††], Ruttikorn Varakulsiripunth[†]
[†] Faculty of Engineering & Research Center for Communications and Information Technology,
[††] Computer Research and Service Center, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand

Daisuke Seki, Kazuki Yoshida, Kaoru Takahashi, Yasushi Kato
Sendai National College of Technology, Japan

Bhed Bahadur Bista
Faculty of Software and Information Science, Iwate Prefectural University, Japan

Norio Shiratori
Research Institute of Electrical Communication, Tohoku University, Japan

Abstract

One of the most important methods in communication system design, especially in protocol design, is a composition method. We propose a method for simultaneously composing service specifications and protocol specifications based on the LOTOS language. In this method, asynchronous communication model is adopted which is more realistic in actual communication networks. We use the concept of the weak bisimulation equivalence to represent the correctness of composition. A software support system based on the proposed composition method is also described.

1. Introduction

A protocol consists of a set of processes (entities) communicating with each other under the defined rules to accomplish a common task. There are two important stages in protocol design, i.e., service specification stage and protocol specification stage. In the service specification stage, a service is specified by the temporal ordering of actions that may occur at different Service Access Points (*SAPs*). In the protocol specification stage, a specification of a communicating process for each *SAP* is defined so that each process provides service at its *SAP* by exchanging messages with other processes.

There are researches for methods that design a protocol by decomposing service specification into protocol specification [2], [4], [5]. On the other hand, there are researches for methods that design a new protocol by designing partial functional protocol specification group independently [6], [7], [8]. In the latter research, the service is excluded although it is an important component for communication systems. For this reason, Bista et al. [3] proposed a composition method of a protocol that considers in parallel with service and protocol. In their method, service specifications are decomposed into equivalent protocol specifications by Langerak's decomposition algorithm [2]. The composition is done under synchronous protocol model. The specifications are specified in LOTOS [1]. The composition pattern corresponds to enabling, choice, parallel and disabling operators in LOTOS.

In this paper, we propose a composition method in *asynchronous* protocol model based on the above composition method. The composed service specification and the composed protocol specification preserve the equivalency.

At first, we summarise LOTOS. Next, we describe asynchronous model of service and protocol. Then we describe the basic concept of our composition approach for asynchronous model and propose a composition algorithm concretely. Then, a support system for the proposed composition method is given. Finally, we conclude this paper.

1531-2216/01 $10.00 © 2001 IEEE

2. LOTOS

LOTOS (Language Of Temporal Ordering Specification) [1] is an FDT (Formal Description Technique) developed by ISO for the formal description of distributed systems. The process behavior is expressed in terms of temporal ordering of its actions. Operators which we use in this paper are shown in Table 1.

Table 1. Basic LOTOS operators

Name	Operator
Inaction	**Stop**
Successful termination	**Exit**
Action prefix	a; P , i; P
Instantiation	P
Choice	P1 [] P2
Enabling	P1 >> P2
Parallel composition (Synchronization)	P1 |[G]| P2
Parallel composition (Interleaving)	P1 ||| P2
Disabling	P1 [> P2

3. Service and Protocol Models

In this section, we describe an asynchronous communication model. Next, we present a polling mechanism for preserving equivalency between a service specification and a protocol specification. Since, in this paper, we assume that a protocol specification is automatically derived from its service specification using the decomposition algorithm by Langerak [2], we also summarise the decomposition algorithm.

3.1 Service Model

As shown in Figure 1(a), a service specification is specified by the temporal ordering of actions that occur at *SAP*s. The service specification is in action-prefix form, i.e., $S = \sum\{ai; Ai \mid i \in I\}$ for some finite index set I where each Ai is either a process identifier or an expression in action-prefix form. The behavior of a specification is represented by an LTS (Labelled Transition System). For instance, for $I = \{1, 2\}$, we have $S = a1; A1 [] a2; A2$ and if $A1 = b1$; stop and $A2 = b2$; stop then, $S = a1; b1;$ stop $[] a2; b2;$ stop. In this case, the LTS of this specification S is shown in Figure 2.

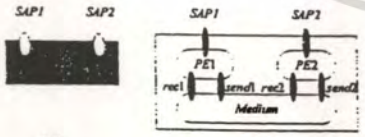


Figure 1. (a) Service model and (b) Asynchronous protocol model

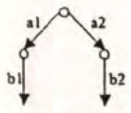


Figure 2. LTS of service specification

3.2 Protocol Model

As shown in Figure 1(b), a protocol specification is the specification of communication entities which communicate with each other to provide a service at *SAPs*. We suppose that the protocol model has two *SAPs*, i.e., *SAP1* and *SAP2*, and provide two protocol entities, *PE*1 and *PE*2. The protocol specification specifies the behavior of the protocol entities mediated by *Medium*.

According to Figure 1(b), a protocol is expressed by the following expression in LOTOS where *Medium* is defined in Section 3.4.

(*PE*1 ||| *PE*2) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The protocol is a realization of the service. The service and the protocol specifications must be equivalent when internal actions, which occurred except *SAPs*, are hidden from the environment. We apply weak bisimulation relation ($\approx$) as the basis of equivalency [1].

3.3 Polling Mechanism

In the asynchronous protocol model, each entity communicates via *Medium* as shown in Figure 1(b). Protocol entities communicate in parallel. The order of actions in protocol specification and the order of actions in service specification may be different when actions at different nodes are chosen by the environment. These may not be bisimilar, because the temporal order may not be preserved. Therefore, we apply a mechanism to exchange a polling message, which is internal action, between entities in order to preserve the order. Figure 3(a) is an LTS of a simple service specification where an action $a1$ occurs at *node*1 and an action $b2$ occurs at *node*2. Figure 3(b) is an LTS of the corresponding protocol specification using the polling mechanism in each node. Figure 3(a) and 3(b) are weak bisimulation equivalent.

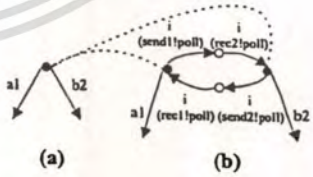


Figure 3. (a) LTS of service specification and (b) LTS of protocol specification with polling mechanism

3.4 Summary of Decomposition Approach

The basic idea of the decomposition approach [2] is to send either a send or a receive action signal after an observable action. We decompose a service specification *S*. It is assumed that *T1(S)* and *T2(S)* are protocol entities corresponding to *node*1 (*SAP1*) and *node*2 (*SAP2*), respectively. We suppose *Med*1 and *Med*2 are buffers within *Medium* that can hold one message concurrently. A message from *T1*(*S*) to *T2*(*S*) is sent at gate *send*1 and received at gate *rec*2. A message from *T2(S)* to *T1(S)* is sent at gate *send*2 and received at gate *rec*1. *Medium* is defined as follows, where *M* is the universe of messages.

Medium = *Med*1 ||| *Med*2 where

Med1 = $\sum\{send1!m; rec2!m; Med1 \mid m \in M\}$

Med2 = $\sum\{send2!m; rec1!m; Med2 \mid m \in M\}$

In this decomposition algorithm, we accomplish specifications of entities by **T1act***S*, **T1pas***S*, **T2act***S*, and **T2pas***S*. In Figure 1(b), **T1act***S* corresponds to *PE*1, and **T2pas***S* corresponds to *PE*2. **T1act***S* specifies a state that can observe an action in *T1(S)*, we call this the active version of *T1(S)*. **T1pas***S* specifies a state where *T1(S)* can receive message from *T2*(*S*), we call this the passive version of *T1(S)*. As well, *T2*(*S*) has the active version **T2act***S* and passive version **T2pas***S*.

Theorem 1 Let *B* be a process in action-prefix form and **T1act**(*B*) and **T2pas**(*B*) be defined by the decomposition algorithm [2]. Then

B = **hide** *send*1, *rec*1, *send*2, *rec*2 **in**
(**T1act**(*B*) ||| **T2pas**(*B*))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium* □

4. Composition Method

In this section, we propose our composition method for service and protocol specifications. First, the composition patterns of specifications are described. Next, the characteristics of the composition method is given followed by the composition algorithms.

4.1 Composition Patterns

Protocol specifications, *P*1 and *P*2 are derived from the service specifications, *S*1 and *S*2 by applying the decomposition algorithm described above. Service and protocol specifications must be weak bisimilar.

Under these conditions, we compose *S*1 and *S*2, and *P*1 and *P*2. The composition results of the service and the protocol specifications must be weak bisimilar again as follows (Figure 4).

*S*1**S*2 ≈ **hide** *send*1, *rec*1, *send*2, *rec*2 **in**
((**T1act***S*1 * **T1act***S*2) ||| (**T2pas***S*1 * **T2pas***S*2))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

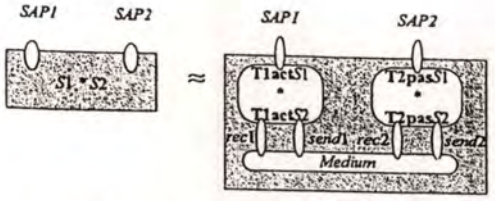


Figure 4. Relation between the composed service and protocol specifications

The symbol * is a LOTOS operator that means either enabling, choice, parallel or disabling. We describe these four composition patterns below when service specifications *S*1 and *S*2 have been given.

4.2 Characteristics of Composition Method

In this section, we explain the preconditions and the characteristics of the composition and the decomposition of the specifications. In order to manifest that the composition of the protocols can be done without including contradiction, it is important to preserve the weak bisimulation relation between the composition of the service specifications which represent the external behavior of the system and the composition of the protocol specifications which represent how to provide the services in the system. The protocol specifications are derived by the decomposition of the service specifications using the Langerak's decomposition algorithm [2]. In case of decomposition in the asynchronous model, preserving the order of actions is controlled by sending and receiving the synchronization actions through the medium between the entities.

The expression *B* as the object of the decomposition means the service which may contain some choice between different nodes. Therefore the polling messages should be added to the derived protocol specification to preserve the equivalency with the service specification. However, at the decomposition to the protocol which does not need the polling messages, it results in an increase of the redundant states. In this paper, we propose a composition algorithm which keeps the polling messages. Consider two service specifications which correspond to the protocol specifications *P*1 and *P*2, respectively. The polling messages exchanged between entities of *P*1 and the polling messages exchanged between entities of *P*2 are distinguished like *poll* and *poll'*. And by doing so, the composition algorithm becomes simple but the composition process becomes complex by an increase of the states.

The basic idea of the composition of service and protocol specifications is as follows.

Given the service specifications $S1$ and $S2$, these correspond to the following protocol specifications $P1$ and $P2$.

$P1$ = (T1act$S1$ ||| T2pas$S1$)
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*
$P2$ = (T1act$S2$ ||| T2pas$S2$)
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

$S1$ and $S2$ are weak bisimulation equivalent with $P1$ and $P2$, respectively when we hide the notification actions *send*1, *rec*1, *send*2 and *rec*2 as shown below.

$S1$ = **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P1$
$S2$ = **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P2$

In order to compose the protocol specifications corresponding to the service specifications under the above mentioned preconditions, we compose the entities each other corresponding to the same node (*SAP*) in $P1$ and $P2$. Namely, the composition of service specifications and protocol specifications are done as follows.

Service specification :
$S1$ * $S2$
Protocol specification:
((T1act$S1$ * T1act$S2$) ||| (T2pas$S1$ * T2pas$S2$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The composition of the services should become weak bisimulation equivalent with the composition of the protocols as shown below when we hide the notification actions *send*1, *rec*1, *send*2 and *rec*2.

$S1$ * $S2$ = **hide** *send*1, *rec*1, *send*2, *rec*2 **in**
((T1act$S1$ * T1act$S2$) ||| (T2pas$S1$ * T2pas$S2$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The symbol "*" means one of "[]", ">>" or "|[G]|" in the LOTOS operators. The characteristics of the protocol composition algorithm is that choice, enabling and parallel compositions are possible by using the above mentioned expression. The polling messages, which have been kept when we decompose the service specification, work well to preserve equivalency because they change the nodes, which can observe the actions, in case of existence of the choice between different nodes. In case of disabling, the composition of the services and the protocols does not become equivalent generally using the above expression as follows.

$S1$ [> $S2$ *not* = **hide** *send*1, *rec*1, *send*2, *rec*2 **in**
((T1act$S1$ [> T1act$S2$) ||| (T2pas$S1$ [> T2pas$S2$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

Disabling is the composition that a service can interrupt every states of another service. The polling messages, which have been kept when we decompose the service specification, work to preserve equivalence in only case of choice between the initial actions. So we must consider the another composition method. In the disabling, we create intermediate behavior expressions P and Q as follows, and consider to compose them. Namely, we propose an algorithm creating P and Q.

$S1$ [> $S2$ = **hide** *send*1, *rec*1, *send*2, *rec*2 **in**
(P ||| Q) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

4.3 Composition Algorithms

In this section, we show four types of the composition algorithms corresponding to the LOTOS operators ">>", "|[G]|", "[]" and "[>".

4.3.1 Enabling Enabling composition between two service specifications $S1$ and $S2$ is shown as "$S1$ >> $S2$" using the LOTOS operator ">>". It means that after $S1$ finished normally, $S2$ occurs. So we compose the protocol entities at the same node of $P1$ and $P2$ directly, and the equivalency is preserved.

[Algorithm 1]
We assume the following protocol specifications $P1$ and $P2$ from the service specifications $S1$ and $S2$ by the decomposition algorithm.

$P1$ = (T1act$S1$ ||| T2pas$S1$)
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*
$P2$ = (T1act$S2$ ||| T2pas$S2$)
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

$S1$ and $S2$ are weak bisimulation equivalent with $P1$ and $P2$, respectively, when we hide the notification actions *send*1, *rec*1, *send*2 and *rec*2 as shown below.

$S1$ = **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P1$
$S2$ = **hide** *send*1, *rec*1, *send*2, *rec*2 **in** $P2$

The compositions of the services and the protocols are executed as follows.

Service: $S1$ >> $S2$
Protocol: ((T1act$S1$ >> T1act$S2$) ||| (T2pas$S1$ >> T2pas$S2$)) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

They are weak bisimilar when we hide the notification actions.

4.3.2 Parallel Parallel composition between two service specifications $S1$ and $S2$ is shown as "$S1$ |[G]| $S2$" using the LOTOS operator "|[G]|" where $G = Act(S1) \cap Act(S2)$. G means a set of the synchronization actions at $S1$ and $S2$. We define $Act(\ )$ as set of actions. If $G = \varnothing$, $S1$ and $S2$ become an asynchronous parallel composition, and if $G \neq \varnothing$, they become a synchronous parallel composition synchronized at the actions of G. In case that $S1$ and $S2$ have no synchronization actions, the actions of $S1$ and $S2$ can occur independently with any order. On the other hand, in case that $S1$ and $S2$ have some synchronization actions, these actions must occur synchronously at $S1$ and $S2$, and the other actions occur independently. In case of protocol composition, we can compose the protocol entities in parallel at the same node of $P1$ and $P2$, respectively. If $P1$ and $P2$ have some synchronization actions, we compose $P1$ and $P2$ synchronized at such actions and their notification actions.

[Algorithm 2]

The precondition is the same as Algorithm 1. When we express the synchronization actions as $G = Act(S1) \cap Act(S2)$, $G1 = Act(\mathbf{T1act}S1) \cap Act(\mathbf{T1act}S2)$ and $G2 = Act(\mathbf{T2pas}S1) \cap Act(\mathbf{T2pas}S2)$, we execute the parallel composition of the services and the protocols as follows.

Service: $S1$ $|[G]|$ $S2$

Protocol: $((\mathbf{T1act}S1\ |[G_1]|\ \mathbf{T1act}S2)\ |||\ (\mathbf{T2pas}S1\ |[G_2]|\ \mathbf{T2pas}S2))\ |[send1, rec1, send2, rec2]|\ Medium$

They are weak bisimilar when we hide the notification actions.

4.3.3 Choice Choice composition between two service specifications $S1$ and $S2$ is shown as "S1[] S2" using the LOTOS operator "[]". It means that either $S1$ or $S2$ is selected. The polling messages, which have been kept when we decompose the service specifications, work to preserve equivalency.

[Algorithm 3]

The precondition is the same as Algorithm 1. We execute the choice composition of the services and the protocols as follows.

Service: $S1$ [] $S2$

Protocol: $((\mathbf{T1act}S1\ []\ \mathbf{T1act}S2)\ |||\ (\mathbf{T2pas}S1\ []\ \mathbf{T2pas}S2))\ |[send1, rec1, send2, rec2]|\ Medium$

They are weak bisimilar when we hide the notification actions.

4.3.4 Disabling Disabling composition between two service specifications $S1$ and $S2$ is shown as "$S1$ [> $S2$" using the LOTOS operator "[>". The choice composition algorithm can not be applied to disabling directly, because the equivalency is not preserved. So we must consider the another method. We convert the expression of disabling into the another one consisting of the choice only, and consider the disabling algorithm as a repetition of choice.

[Algorithm 4]

When we execute the disabling composition, we convert the expression of disabling composition into the another one consisting of the choice operators only, then we derive P and Q, and we compose P and Q finally.

We assume that $P1$ and $P2$ shown below are derived from the service specifications $S1$ and $S2$ by applying the Langerak's decomposition algorithm without polling messages.

$P1 = (E1\ |||\ E2)\ |[\ send1, rec1, send2, rec2]|\ Medium$

$P2 = (F1\ |||\ F2)\ |[\ send1, rec1, send2, rec2]|\ Medium$

where entity correspondence is as shown below.

$E1$ and $F1$ are entities at *node*1.

$E2$ and $F2$ are entities at *node*2.

Based on the above mentioned premise, we execute the disabling composition of the services and the protocols as follows.

Service: $S1$ [> $S2$

Protocol: $(P\ |||\ Q)\ |[send1, rec1, send2, rec2]|\ Medium$

5. Support System

The support system has four working stages; entering service specifications ($S1$ and $S2$), the selection of the composition operator, the decomposition and the result of composition as shown in Figure 5.

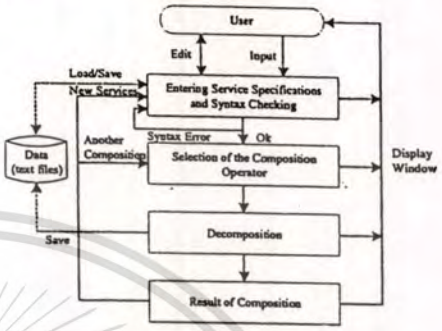


Figure 5. General flow of the support system

At the first stage, a user enters service specifications through the service specification windows and can save them as text files and reuse them to specify again. There is a little difference from LOTOS in action prefix form, that is, a node number is put within "{ }" symbol. After the user enters $S1$, the support system checks the syntax of expression based on the LOTOS syntax. If $S1$ has syntax errors, the support system shows these errors. If $S1$ has no error, the support system permits the user to enter $S2$. Entering $S2$ is the same as the case of $S1$.

At the second stage, the support system shows composition method window for the user to select the LOTOS operator. The user can select one of four types of composition operators.

At the third stage, the support system decomposes service specifications into protocol specifications automatically and shows the protocol specification window which includes protocol specifications of $S1$ and $S2$.

At the last stage, the support system shows that the composition of the services and the composition of protocols are weak bisimilar when notification actions (internal actions) are hidden. The support system provides the user with the selection of the another compositions or the composition of new services.

[Example] We use the data transfer in computer communication system to demonstrate the support system. In Figure 6(a), *node*1 requests to send data to *node*2 and in Figure 6(b), *node*1 requests to disconnect the connection to *node*2. This example shows the

interruption of data transfer by disconnection. The support system initially presents the user with protocol specification model as shown in Figure 7. Figure 8(a) and 8(b) show service specifications $S1$ and $S2$, respectively. Then, the system provides the user with a window to select one of four types of composition operators in order to compose services and protocols specifications as shown in Figure 9.

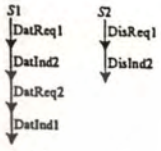


Figure 6. Service specifications $S1$ and $S2$

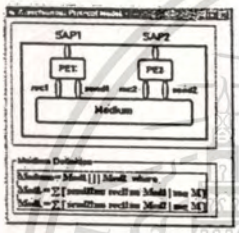


Figure 7. Asynchronous model

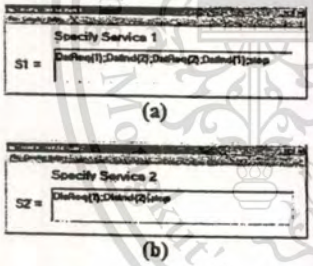


Figure 8. (a) Service specification $S1$ and (b) Service specification $S2$

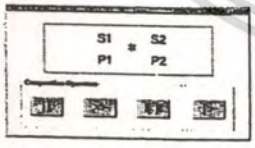


Figure 9. Composition operators

Figure 10 shows the composition result when the disabling operator is selected for composition. When we hide the notification actions, the composition of services and the composition of protocols are weak bisimilar.

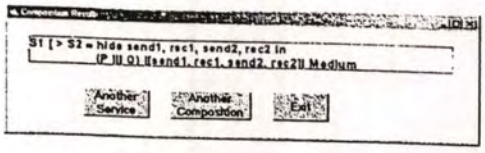


Figure 10. Composition result using disabling operator

6. Conclusion

We have proposed composition algorithms of service and protocol specifications, in case of four composition types, i.e., enabling, parallel, choice and disabling corresponding to the LOTOS operators, in the asynchronous communication model. In our previous algorithm [3], we used synchronous model. However, asynchronous model is more suitable in the actual communication networks. So we believe that the composition method in this paper contributes to appropriate design of services and protocols. Based on the proposed composition algorithms, we have developed its support system. It is expected that an efficient service and protocol development will be accomplished by using the support system.

References

[1] ISO, Information Processing Systems, "Open Systems Interconnection-LOTOS-A Formal Description Technique based on the Temporal Ordering of Observational Behavior," ISO 8807, 1989.

[2] R.Langerak, "Decomposition of Functionality: A Correctness Preserving LOTOS Transformation," in Protocol Specification, Testing and Verification, pp.203-218, 1990.

[3] B.B.Bista, K.Takahashi and N.Shiratori, "A Compositional Approach for Constructing Communication Services and Protocols," IEICE Trans. Fundamentals, Vol.E82-A, No.11, pp.2546-2557, 1999.

[4] G.Bochmann and R.Gotzhein, "Deriving Protocol Specification from Service Specification," Proc. SIGCOMM'86, Vol.14, pp.144-156, 1986.

[5] C.Kant, T.Higashino and G.V.Bochmann, "Deriving Protocol Specifications from Service Specifications Written in LOTOS," Distributed Computing, Vol.10, No.1, pp.29-47, 1996.

[6] A.Khoumsi and K.Saleh, "Two Formal Methods for the Synthesis of Discrete Event Systems," Computer Networks and ISDN Systems, Vol.29, No.7, pp.759-780, 1997.

[7] H.-A.Lin and C.L.Tang, "An Improved Method for Constructing Multiphase Communications Protocols," IEEE Transactions on Computers, Vol.42, No.1, pp.15-26, 1993.

[8] M.Nakamura, Y.Kakuda and T.Kikuno, "On Constructing Communication Protocols from Component-Based Service Specifications," Computer Communications, Vol.19,, pp.1200-1215, 1996.

Specification and Verification of Communication Systems Based on State Machine Model with Time Constraint

*Noppadol Maneerat**,†, Purich Chaochanaphun*, Ruttikorn Varakulsiripunth*, Kaoru Takahashi**, and Yasushi Kato***

*Faculty of Engineering, and Research Center for Communications and Information Technology (ReCCIT), and †Computer Research and Service Center, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand. (email: kvruttik@kmitl.ac.th)

** Sendai National College of Technology 4-16-1, Chuo, Ayashi, Aoba-ku, Sendai, 989-3128 Japan. (email: kato@info.sendai-ct.ac.jp)

Abstract

An approach of design and analysis of communication systems is proposed by using a specification method based on the state machine model. The time constraint is introduced into the model in order to make it possible to specify the time sequential processing of communication systems. We have also constructed design and analysis tool of this method that is very helpful for investigating and verifying the specified system before real implementation.

1. Introduction

Most of communication procedure is a concurrent system that consists of a number of communicating entities. To specify such systems it needs to consider their concurrency together with communication. The *process calculi* such as CCS [1] and CSP [2] have been proposed as specification methods by modeling the entities as process represented by algebraic expressions. Though it can deal with the concurrency, it is an abstract one. Several specification methods based on a *state machine model* have been proposed [3~11] that are not as abstract as the *process calculi* and are closer to implementation.

This paper deals with specification methods based on state machine models by adding time constraint into the models. This makes it possible to specify sequential processing of real-time communication systems.

First, we introduce a single basic state machine and modeled it so that it can interact with the environment via channels. By defining a communication system as a collection of such state machines, the whole behavior of the communication system is given in terms of interactions among the state machines and their individual internal activities. Next, we introduce constrained state machines that can adapt to various specification purposes. Finally, we have developed a design and analysis tool based on our specification method. This tool can be used to analyze the communication system in state diagram form. By using this tool, the behaviors of a specified system can be investigated and verified before the implementation.

2. Basic state machine

A (*basic*) *state machine* has *variables* in addition to the usual (control) states. Each variable has its own *type* like *integer* or *string*. Let V be a set of variables, then a *valuation* of V is a mapping that associates to each variable of V, a value in its domain. We write V for the set of valuations of V.

A state machine can perform interaction with the environment or other state machines. For a state machine, this interaction appears as an input or output of data via a *channel*. We denote the set of channels accepting an input set I as C^{in} and the set of channels accepting an output set O as C^{out}. In the following definition, a data input/output of a state machine via a channel is formulated as an *event* that causes a state transition of state machine.

Definition 1: A *state machine* M is a 5-tuple:

$$M = \langle Q, V, \Sigma, T, \theta \rangle$$

- Q is a finite set of *control states*.
- V is a finite set of *variables*.
- Σ is a set of three *events*, those are Σ^{in} ($\Sigma^{in} \subseteq C^{in} \times I$), Σ^{out} ($\Sigma^{out} \subseteq C^{out} \times O$) and $\{\tau\}$ which are a set of input event from environment, a set of output event to the environment and a set of *internal* event, respectively.
- $T \subseteq (Q \times \mathrm{V}) \times \Sigma \times (Q \times \mathrm{V})$ is a *transition relation*.
- $\theta \subseteq Q \times \mathrm{V}$ is a set of *initial state*.

We say $s \in Q \times \mathrm{V}$ is a *state* of the state machine M. Since there may be a case where the domain of a variable is infinite, the number of states of M may not be finite.

We write $s \xrightarrow{\sigma} s'$ for a $(s, \sigma, s') \in T$. If there exists a state s' such that $s \xrightarrow{\sigma} s'$ for a state s and an event σ, we write $s \xrightarrow{\sigma}$.

Notation 1: We use a usual state transition diagram for a visual representation of a state machine. If an explicit specification of variable values is needed, it is attached to the control state. A basic structure of state transition is represented as follows:

current_state $\xrightarrow{guard \rightarrow event}$ *next_state*

The *guard* is a Boolean expression that qualifies the values of the variables related with the transition. If it is not necessary or it is always *true*, it can be omitted.

The *event* is written as follows if it is an input.

channel_name?*variable_name* "value"

This represents that the state machine inputs data with the value specified by "value" via the channel specified by *channel_name*, and assigns it to the variable specified by *variable_name*. If data is structured, for example, a protocol message whose *type* is defined and which has some parameters, it is written as follows.

channel_name?(*parameter*, ..., *parameter*)

Each *parameter* is represented as the above described *variable_name*:*type* "value".

The *event* is written as follows if it is an output.

channel_name!*expression*

This represents that the state machine outputs data with the value obtained by interpreting the specified *expression* via the channel specified by *channel_name*. For a structured data, its form conforms to the above description as follows.

channel_name!(*parameter*, ..., *parameter*)

An expression is written in each *parameter* part.

If the *event* is internal, the processing statements that may change the variable should be written. We use the symbol τ to represent the internal event.

Example 1: Figure 1 shows a notation example of state machines for data transmission in a communication system in which Sender transmits data to Receiver via Medium. The control states of Sender are $S0$ and $S1$. Sender inputs data into the variable *din* via the channel *p* from the environment and delivers it to Medium via the channel *s*. Sender's initial state is $S0$. The control states of Medium are $M0$, $M1$ and $M2$, and its initial control state is $M0$. There are four variables: *no*, *i*, *o* and *data*. *data* is a buffer for storing data. *no* represents the number of data in the buffer, *i* indicates the position in the buffer into which the next data is input, and *o* indicates the position in the buffer from which the next data is output. The initial values of these three variables are all 0's. *MAX* is a constant that represents the maximum of the number of data capable of storing. Within this range, data can be input into the buffer via the channel *s*. If there are one or more data in buffer, Medium can output data to Receiver via the channel *r*. While inputting and/or outputting data, the necessary update of the variable values is performed, with internal transitions. The control states of Receiver are $R0$ and $R1$, and its initial control state is $R0$. Receiver inputs data into the variable *dout* via the channel *r*, and outputs it to the environment via the channel *c*.

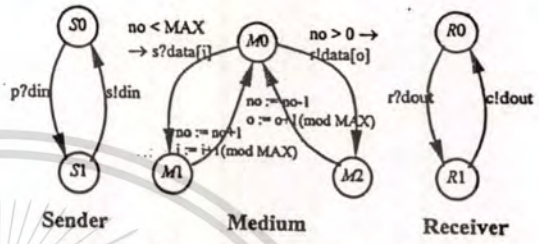


Figure 1. Data Transfer in a Communication System.

Definition 2: A *computation* of a state machine M is an alternating sequence of states and events

$\alpha = s_0\sigma_0 s_1\sigma_1 s_2\sigma_2 \ldots$

which satisfies the following conditions:

1. $s_0 \in \theta$.
2. For all $i \geq 0$, $s_i \xrightarrow{\sigma_i} s_{i+1}$.

The set of all computations of a state machine M is denoted as $Comp(M)$.

If $MAX = 2$ in Medium of Example 1, then the following sequence α is a computation of Medium:

α = ($M0$, {*no*=0, *i*=0, *o*=0, *data*[]=*null*}) *s*?"*d*1"
($M1$, {*no*=0, *i*=0, *o*=0, *data*[]="*d*1"}) τ
($M0$, {*no*=1, *i*=1, *o*=0, *data*[]="*d*1"}) *s*?"*d*2"
($M1$, {*no*=1, *i*=1, *o*=0, *data*[]="*d*1" · "*d*2"}) τ
($M0$, {*no*=2, *i*=0, *o*=0, *data*[]="*d*1" · "*d*2"}) *r*!"*d*1"
($M2$, {*no*=2, *i*=0, *o*=0, *data*[]="*d*1" · "*d*2"}) τ
($M0$, {*no*=1, *i*=0, *o*=1, *data*[]="*d*1" · "*d*2"}) *s*?"*d*3"
($M1$, {*no*=1, *i*=0, *o*=1, *data*[]="*d*3" · "*d*2"}) ....

where "*d*1", "*d*2", "*d*3" ..... represent data values.

Definition 3: Let $\alpha = s_0\sigma_0 s_1\sigma_1 s_2\sigma_2 \ldots$ be a computation of a state machine M. The sequence obtained by removing all states and all τ-events from α is said to be a *trace* of M.

We can think of a trace as an observation of a computation of a state machine from the outside (environment). The set of all traces of state machine M is denoted as $Trace(M)$.

From the above mentioned example, a trace of Medium is written as follows.

$$tr = s?"d1"\ s?"d2"\ r!"d1"\ s?"d3"\ \cdots$$

3. State machine with time constraint

Basic state machines described above do not have the concept of time. Here, we have introduced a time constraint into the transitions of a state machine to deal with time-dependent computations and traces. Consequently, it becomes possible to specify real-time systems, communication protocols and so on.

Here, a *time domain* Ψ is assumed to contain the minimum element 0 representing the starting time and have a usual +-operation over its elements. Ψ is also assumed to be totally ordered. In this paper, we think of the non-negative real numbers $R \geq 0$ as Ψ. We extend Ψ to $\Psi^{\infty} = \Psi \cup \{\infty\}$ where $\infty \geq t$ for all $t \in \Psi^{\infty}$. Then the time sequence ψ is defined as follows.

Definition 4: A *time sequence* ψ is a sequence of time represented by,

$$\psi = t_0\, t_1\, t_2\, t_3\, t_4 \cdots\cdots$$

which satisfies the following requirements.

- $t_i \in \Psi$ for all $i \geq 0$.
- $t_i \leq t_{i+1}$ for all $i \geq 0$.
- if ψ is an infinite sequence, there exists $i \geq 0$ such that $t_i > t$ for all $t \in \Psi$.

In our state machine model without time concept, the firing (enabling) condition of a transition is that the current state of the state machine is at the source of the transition and the corresponding event is possible to occur. Considering the time constraint however, the *minimal delay* l and the *maximal delay* u until its firing are also taken into account in addition to that firing condition. Let φ be a function that gives transitions their minimal delays and maximal delays, as a time constraint. Then we denote the minimal delay of a transition μ as $\varphi_l(\mu)$ and the maximal delay as $\varphi_u(\mu)$, where $\varphi_l(\mu) \leq \varphi_u(\mu)$. We assume that the firing of each transition is instantaneous (i.e., takes no time).

Definition 5: Let $M = \langle Q, V, \Sigma, T, \theta \rangle$ be a state machine, and $\varphi\,(\varphi : T \rightarrow [\Psi \times \Psi^{\infty}])$ be a total function which gives each $\mu \in T$ a pair of $\varphi_l(\mu)$ and $\varphi_u(\mu)$, where $\varphi_l(\mu) \leq \varphi_u(\mu)$. Then a pair $\langle M, \varphi \rangle$ is said to be a *state machine with a time constrain* φ.

We assume that each state machine with a time constraint has a special, implicit channel *CLK* which can provide the state machine with the *current time* at any timing. And this channel is not affected by other constraints.

Notation 2: When describing the minimal delay l and maximal delay u of a transition in a state diagram, it takes the following form:

$$current\_state \xrightarrow{guard \rightarrow event[l,\,u]} next\_state$$

The part of $[l, u]$ can be omitted if $l = 0$ and $u = \infty$.

Definition 6: Let s be a state of a state machine with a time constraint. For some transitions whose source states are s and events are internal, the minimum of their maximal delays is denoted as $mm(s)$. If the transitions whose events are internal do not exist, it is defined to be $mm(s) = \infty$. If μ is a transition from s and satisfies the following conditions, then μ is said to be *firable at (relative) time t*:

1. $\varphi_l(\mu) \leq t \leq \varphi_u(\mu)$.
2. $t \leq mm(s)$.

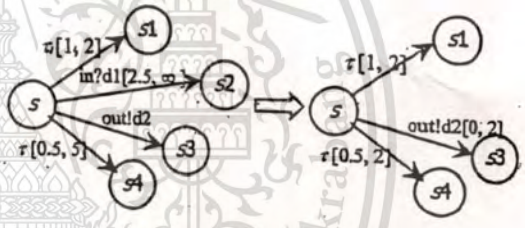


Figure 2. Firability of transitions.

Figure 2 shows an example of firability of transitions. In this case, since $mm(s) = 2$, the firability of each transition from state s is described as follows.

$s \xrightarrow{\tau} s_1$ is firable at any time between 1 and 2.

$s \xrightarrow{in?d1} s_2$ is not firable at any time.

$s \xrightarrow{out!d2} s_3$ is firable at any time between 0 and 2.

$s \xrightarrow{\tau} s_4$ is firable at any time between 0.5 and 2.

Thus the transitions on the left part can be interpreted by the right part as shown in the Figure 2.

We model a computation of a state machine with a time constraint by using *time-stamped events* and a *time sequence*. A time-stamped event takes the form of $\langle \sigma, t \rangle$ and it represents that an event σ occurs at time t.

Definition 7: Let $\langle M, \varphi \rangle$ be a state machine with a time constraint. An alternating sequence of states and time-stamped events

$$\alpha = s_0 \langle \sigma_0, t_0 \rangle s_1 \langle \sigma_1, t_1 \rangle s_2 \langle \sigma_2, t_2 \rangle \cdots$$

is a *computation* of $\langle M, \varphi \rangle$ if α satisfies the following conditions:

1. $s_0 \sigma_0 s_1 \sigma_1 s_2 \sigma_2 \ldots$ is a computation of M.
2. $t_0 t_1 t_2 \ldots$ is a time sequence.
3. $s_0 \xrightarrow{\sigma_0} s_1$ is firable at t_0. And for all $i \geq 1$, $s_i \xrightarrow{\sigma_i} s_{i+1}$ is firable at $t_i - t_{i-1}$.

The set of all computations of $\langle M, \varphi \rangle$ is denoted as $Comp(\langle M, \varphi \rangle)$.

Definition 8: Let $\alpha = s_0 \langle \sigma_0, t_0 \rangle s_1 \langle \sigma_1, t_1 \rangle s_2 \langle \sigma_2, t_2 \rangle \cdots$ be a computation of a state machine with time constraint $\langle M, \varphi \rangle$. The sequence obtained by removing all states and all time-stamped τ-events from α is said to be a *trace* of $\langle M, \varphi \rangle$. The set of all traces of $\langle M, \varphi \rangle$ is denote as $Trace(\langle M, \varphi \rangle)$.

4. Design and analysis tool

In this section we will present design and analysis tool that we have developed based on the concept of the basic and constrained state machine described in sections 2 and 3. This tool, *"MgraphGen"*, can be used to design and specify the specification of communication systems. The developers can construct specification obviously by the state machine model with a time constraint. They can investigate and analyze the system behaviors easily and simply in the form of graphical design. The detail description is explained as follows.

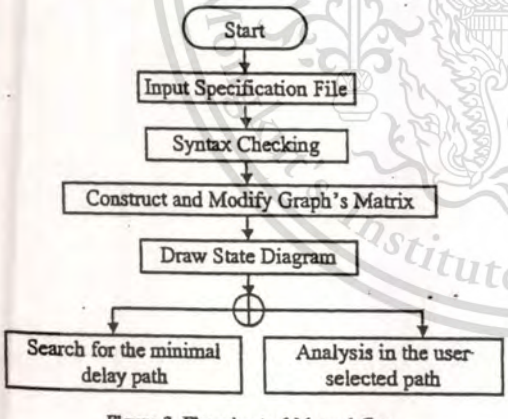


Figure 3. Flowchart of *MgraphGen*.

4.1 General features of *MgraphGen*

MGraphGen can show a state machine model in the form of state diagram corresponding to the specification of the communication system. As shown in Figure 2, firstly, the input specification in the text format will be checked for the correctly specified syntax. Then, the tool will construct and modify the *graph's matrix* in order to show its state diagram as the first result. From this step, user can investigate the processes of the specified system through this state diagram by selecting the following modes. Those are (i) the search for the shortest path or minimal delay path by choosing source and destination states, and (ii) the analysis in the user selected path.

4.2 Operation modules in *MgraphGen*

There are 5 main operation modules in this tool operated sequentially as shown in Figure 3. They are *Syntax Checking Module, Matrix Generating Module, Display Sate Diagram Module, Search for the minimal delay path Module, and Analysis in the user selected path Module.*

4.2.1 Syntax checking module

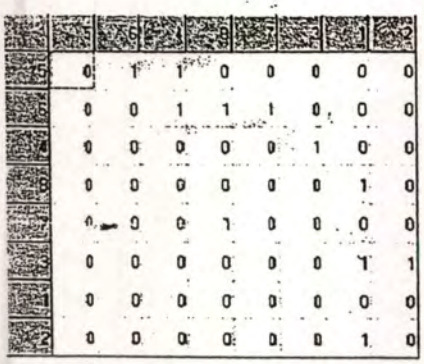
This module is used for checking the specification syntax. The syntax checking process occurs after the input specification file is loaded. If the input file has any syntax errors, it will create the error log file that contains the error or warning messages as an error report. User must correct those errors and re-input the corrected file again. Figure 4 shows a sample of specification file.

Line		Line	
1	5{Tau[2.5,5]}6	7	3{ch3!(2)[8,16]}2
2	5{ch5?a:str"2"[15,30]}4	8	3{Tau[10,20]}1
3	6{ch6?a:int(2)[10,20]}4	9	2{ch2!Acdf("23")[3,6]}1
4	6{ch6?a:int(2)[14,28]}8	10	7{ch7!Acdf((12),"23")}8
5	6{Tau[9,18]}7	11	8{Tau[17,34]}1
6	4{ch4!ab12[12,24]}3		

Figure 4. A sample of specification file.

4.2.2 Matrix generating module

This module is operated after the syntax checking has completed. First, it constructs a graph's matrix from the initial state based on the corrected specification. Then this matrix is adjusted by matrix operation until each state is posted in the proper position. Figure 5 shows modified graph's matrix of specification file shown in Figure 4. The values of this matrix show the number of path from state *i* to state *j*. For example, there is one path from state *5* to state *6*.



0	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0

Figure 5: Graph's Matrix of specification file in Figure 4.

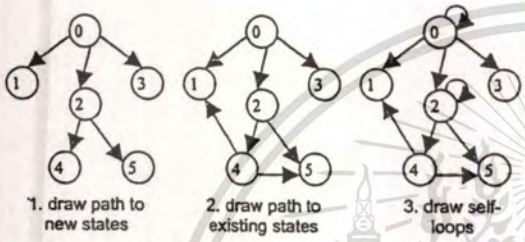


Figure 6. Sample of drawing procedure for state diagram.

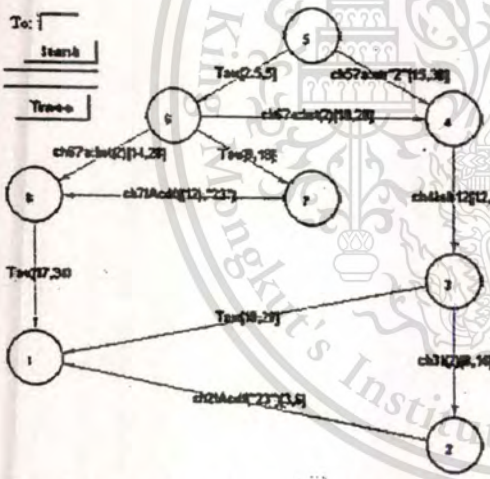


Figure 7. A sample of state diagram of the specification file.

4.2.3 Display state diagram module

This module uses to draw state diagram of the specified system. It is operated after graph's matrix has been constructed and adjusted. This module starts to draw from the initial state to a next state and from the present state to a next state. Then it finds and draws the rest of transition paths until the diagram has been completed. Finally, it draws all of the self-loop transitions. The module will record the position of all the paths for the routing of the next path. It has an ability to avoid the crossing path by finding the existing path and drawing the suitable detour path. Figure 6 shows the drawing procedure of state diagram. Figure 7 shows a sample of state diagram of the specification file in Figure 4.

4.2.4 Search for the minimal delay path module

User chooses this module in order to find the minimal delay path. The Dijkstra's algorithm is introduced into this module for searching the shortest path between specified *source* and *destination states* by the assumption that every link (or event) between two nodes (or states) has a certain minimum delay cost. Then the algorithm will search the path between the two given nodes with the lowest delay cost. The result shows the shortest path as a sequence of states and its total time consumption. Figure 8 shows a shortest path and total delay from *state 5* to *state 1* of state diagram in Figure 7.

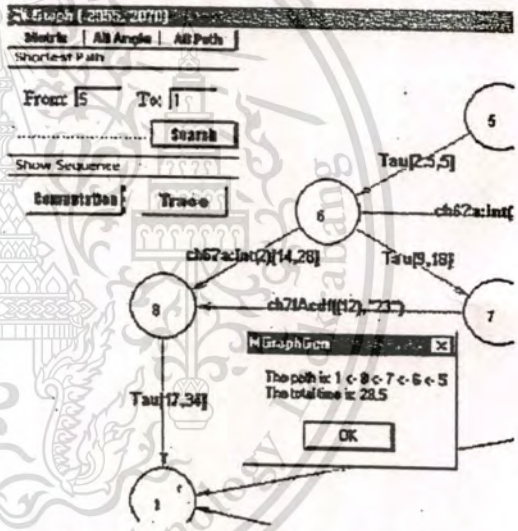


Figure 8. Result of searching for the minimal delay path.

4.2.5 Analysis in the user selected path module

User chooses this module in order to find the computation or the trace the selected path. By clicking on the expected event, the input box will ask user to enter the delay time for enabling that event (see Figure 9). The delay time must be numerical value between minimal and maximal delay of that event. Figure 10 and 11 show a sample of computation and trace results, respectively.

5. Conclusions

In this paper, we have discussed the specification method for a communicatfon system based on the state machine model with a time constraint. Several definitions and notations used in this specification method are described in definite details. Finally, the design and analysis tool, *MgraphGen*, has developed based on those specified definitions and notations.

In the future work, we will consider on the application of our method to a mobile system modeling and specification. In this case, the geographical constraints have to be introduced in the state machine models. And we will also extend the *MGraphGen* to be able to describe the behavior of a communication system in the form of component state machine.

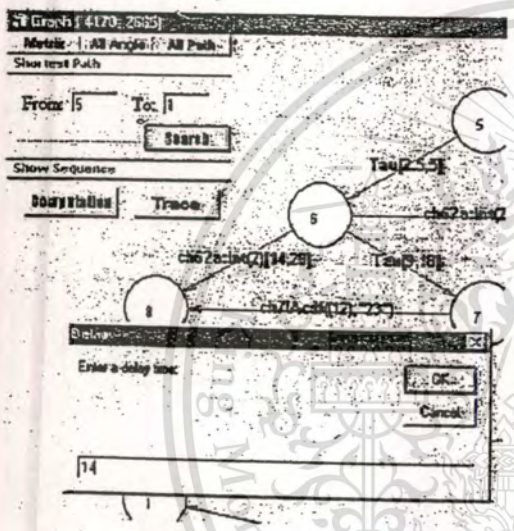


Figure 9. Delay time setting for selected path.

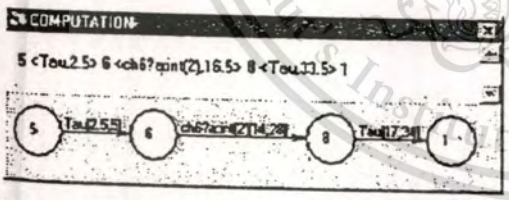


Figure 10. A sample of computation of the user selected path.

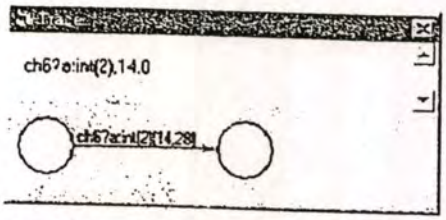


Figure 11. A sample of trace of the user selected path.

6. References

[1] R. Milner, "Communication and Concurrency," Prentice-Hall, 1989.

[2] C. A. R. Hoare, "Communication Sequential Process," Prentice-Hall, 1985.

[3] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. S. Trauring and M. Trakhtenbrot, "Statemate: a Working Environment for the Development of Complex Reactive System," IEEE Trans. Software Eng., Vol.16, No.4, pp.403-414, 1990.

[4] D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," ACM Trans. Software Engineering and Methodology, Vol.5, No.4, pp.293-333, Oct. 1996.

[5] E. Battiston, F. D. Cindio and G. Mauli, "Modular Algebraic Nets to Specify Concurrent Syatems," IEEE Trans. Software Eng., Vol.22, No.10, pp.689-705, 1996.

[6] A. Gabrielian and M. Franklin, "Multilevel Specification of Real-Time Systems," Commun. ACM, Vol.34, No.5, pp.50-60, May. 1991.

[7] F. Jahanian and A. K. Mok, "Modechart: A Specification Language for Real-Time Systems," IEEE Trans. Software Eng., Vol.20, No.12, pp.933-947, Dec. 1994.

[8] A. Coen-Porisini, C. Ghezzi and R. A. Kemmerer, "Specification of Realtime Systems Using ASTRAL," IEEE Trans. Software Eng., Vol.23, No.9, pp.572-598, Sept. 1997.

[9] S. Yamane, "Specification and Verification of Real-time Concurrent Software," (in Japanese) Trans. Information Processing Society of Japan, Vol.37, No.2, pp.188-203, Feb. 1996.

[10] R. Alur and D. Dill, "Automata for Modeling Real-Time Systems," Lecture Notes in Computer Science 443, pp.322-335, 1990.

[11] A. C. Shaw, "Communication Real-Time State Machines," IEEE Trans. Software Eng., Vol.18, No.9, pp.805-816, Sept. 1992.

2306 IEICE TRANS. INF. & SYST., VOL.E87–D, NO.10 OCTOBER 2004

PAPER

Composition of Service and Protocol Specifications in Asynchronous Communication System

Noppadol MANEERAT[†,††], Ruttikorn VARAKULSIRIPUNTH[†a)], Bhed Bahadur BISTA[†††], *Nonmembers*, Kaoru TAKAHASHI[††††], Yasushi KATO[††††], *Members*, *and* Norio SHIRATORI[†††††], *Fellow*

SUMMARY One of the important techniques in communication system design is the composition of service and protocol specifications. In this paper, we have presented a new approach to the composition technique based on the weak bisimulation concept. The main objective is to combine service specifications and protocol specifications individually and simultaneously. The composition technique can maintain the equivalence between the composed service and protocol specifications. LOTOS language terms are utilized to describe the communication specifications. The application on the asynchronous model is presented. Moreover, a support system of the composition technique is developed and presented in this paper.

key words: LOTOS, LTS, weak bisimulation, asynchronous model

1. Introduction

Communication system consists of two elements, i.e., protocol and service. Protocol is a set of processes (entities) communicating with each other under defined rules to accomplish a common task. Service is a set of requested tasks that are processed by protocol. Therefore, the proper design of service and protocol specifications becomes important. A service is specified based on the temporal ordering of actions that may occur at different Service Access Points (*SAPs*). A protocol is specified so that each communicating process provides service at its *SAP* by exchanging messages with each other.

In previous studies related to communication system design, the technique to derive protocol specifications from service specifications was presented [2], [4], [5], and the technique of partial functional protocol specification was discussed [6]–[8]. Bista et al. [3] proposed a compositional approach for constructing communication service and protocol simultaneously. They considered service and protocol as parallel elements. The advantage of this approach is that it is easy to design and verify the specifications because the composed protocols were defined in terms of subfunctions. They used Langerak's algorithm [2] to decompose service and to construct the equivalent protocol specifications. Four major operators in LOTOS [1], i.e., enabling, choice, parallel, and disabling were used in composition method. They applied this technique to synchronous communication model to show its availability.

In this paper, we have modified and extended the composition technique of Bista et al., in order to apply it to the *asynchronous* communication model. We use the weak bisimulation concept to combine service specifications and protocol specifications individually and simultaneously while maintaining the equivalence between composed service and composed protocol specifications. Finally, we have developed its support system.

2. Definition of Transition and Bisimulation

LOTOS (Language Of Temporal Ordering Specification) [1] is an FDT (Formal Description Technique) developed by ISO for the formal description of distributed systems. The process behavior is expressed in terms of temporal ordering of its actions. Operators used in this paper to represent a process are shown in Table 1.

"stop" denotes the inaction of the specified process.

"exit" denotes a process which performs the successful termination action δ and becomes stop.

"a; P" and "i; P" denote process P preceded by the action a and the internal or unobservable action i, respectively.

"P" denotes the invocation of process P.

"$P1$ [] $P2$" denotes a process with the alternative composition of processes $P1$ and $P2$ where it is ready to behave as $P1$ or as $P2$.

"$P1$ >> $P2$" denotes a process with the sequential com-

Table 1 Basic LOTOS operators.

Name	Operator
Inaction	stop
Successful termination	exit
Action prefix	a; P, i; P
Process Instantiation	P
Choice	$P1$ [] $P2$
Enabling	$P1$ >> $P2$
Parallel composition (Synchronization)	$P1$ |[G]| $P2$
Parallel composition (Interleaving)	$P1$ ||| $P2$
Disabling	$P1$ [> $P2$

Manuscript received April 22, 2002.
Manuscript revised April 12, 2004.

[†]The authors are with the Faculty of Engineering & Research Center for Communications and Information Technology, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand.
[††]The author is with Computer Research and Service Center, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand.
[†††]The author is with the Faculty of Software and Information Science, Iwate Prefectural University, Iwate-ken, 020–0173 Japan.
[††††]The authors are with Sendai National College of Technology, Sendai-shi, 989–3128 Japan.
[†††††]The author is with Research Institute of Electrical Communication, Tohoku University, Sendai-shi, 980–8577 Japan.
a) E-mail: kvruttik@kmitl.ac.th

position of processes $P1$ and $P2$ where execution of actions of $P2$ starts only after $P1$ successfully terminates.

"$P1$ |[G]| $P2$" denotes a process with parallel composition of processes $P1$ and $P2$ where G specifies the list of synchronization actions. An action listed in G or δ can be executed as a common action of $P1$ and $P2$ while other actions are executed independently. When G is empty, i.e. |[]|, it is usually represented by |||.

"$P1$ [> $P2$" denotes a process with the interrupted composition of processes $P1$ and $P2$ where $P1$ is interrupted by $P2$.

Here, our technique is mainly concerned with a labeled transition system to describe the formal semantics of a process. We define Act as a set of actions, Act^* as a set of sequences of actions in Act and i* as zero or more i (internal) actions. $Act(P)$ is defined as the set of actions which process P can execute.

[Definition 1] A Labeled Transition System (LTS) L is a quadruple $< S, A, T, s_0 >$, where S is a nonempty set of states, A is a subset of Act, $T \subseteq S \times A \times S$ is a transition relation and $s_0 \in S$ is the initial state of L. □

Using the operational semantics given in the above definitions, a bisimulation relation is defined:

[Definition 2] Let $L_1 =< S_1, A_1, T_1, s_{10} >$ and $L_2 = < S_2, A_2, T_2, s_{20} >$ be labeled transition systems. A binary relation $R \subseteq S_1 \times S_2$ is a weak bisimulation relation if $(s_1, s_2) \in R$ implies that, for all $t \in (Act - \{i\})^*$,

1. if $s_1 \stackrel{t}{\Rightarrow} s_1'$ for some s_1', then $s_2 \stackrel{t}{\Rightarrow} s_2'$ and $(s_1', s_2') \in R$ for some s_2'
2. if $s_2 \stackrel{t}{\Rightarrow} s_2'$ for some s_2', then $s_1 \stackrel{t}{\Rightarrow} s_1'$ and $(s_1', s_2') \in R$ for some s_1'. □

In this definition, if $t = a_1 \ldots a_n \in (Act - \{i\})^*$, then $s \stackrel{t}{\Rightarrow} s'$ stands for

$$s \stackrel{i^*}{\longrightarrow} \stackrel{a_1}{\longrightarrow} \stackrel{i^*}{\longrightarrow} \stackrel{a_2}{\longrightarrow} \stackrel{i^*}{\longrightarrow} \ldots \stackrel{i^*}{\longrightarrow} \stackrel{a_n}{\longrightarrow} \stackrel{i^*}{\longrightarrow} s'$$

Here, the notation $s_1 \stackrel{a}{\rightarrow} s_2$ represents the transition from state s_1 to state s_2 by the execution of the action a.

L_1 is weakly bisimilar to L_2 (or L_1 is weakly bisimulation equivalent to L_2) written as $L_1 \approx L_2$, if $(s_{10}, s_{20}) \in R$ for some weak bisimulation R. A process P is weakly bisimilar to a process Q (or P is weakly bisimulation equivalent to Q), written as $P \approx Q$, if the labeled transition systems of P and Q are weakly bisimulation equivalent.

3. Service and Protocol Models

In this section, the models of service and protocol in asynchronous communication system are described first. Next, a polling mechanism is introduced to preserve the order of actions and is used to let the behaviors of service and protocol be equivalent. Finally, the decomposition algorithm that is used to derive the equivalent protocol specification from a service specification automatically is summarized.

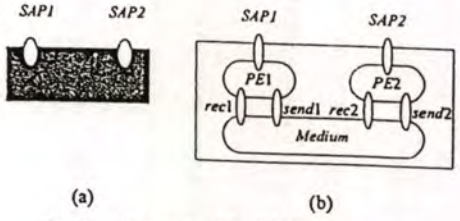


Fig. 1 (a) Service model and (b) Protocol model.

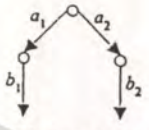


Fig. 2 LTS of service specification.

3.1 Service Model

A service is modeled as a black box as shown in Fig. 1 (a). Here, we know only what is provided at each *SAP* but we do not know how. The black box represents all of the lower layers including local node, network and remote node. A service is described by the temporal ordering of actions that occur at *SAPs*. In this paper, we assume that there are two *SAPs*, where *SAP*1 is at node 1 and *SAP*2 is at node 2. The service specification is in an *action-prefix form*, e.g., $S = \sum\{a_i; A_i | i \in I\}$ for some finite index set I where each A_i is either a process identifier or an expression in an action-prefix form. The behavior of a specification is represented by an LTS. We define $\sum$ as the generalized choice among behavior expressions distinguished by the index set I. For example, if $S = \sum\{a_i; A_i | i \in I\}$ for some finite index set I where each A_i is either a process or an expression in an action-prefix form, and when $I = \{1, 2\}$, we have $S = a_1; A_1 [] a_2; A_2$ and furthermore if $A_1 = b_1$; stop and $A_2 = b_2$; stop then $S = a_1; b_1;$ stop[]$a_2; b_2;$ stop. In this case, the LTS of this specification S is shown in Fig. 2.

3.2 Protocol Model

In contrast to service, a protocol is modeled as a white box shown in Fig. 1 (b). A protocol is a specification of communication entities that communicate with each other to provide a service at *SAPs*. We also assume that the protocol model has two *SAPs*, (*SAP*1 and *SAP*2), with two protocol entities, *PE*1 and *PE*2. The protocol specification specifies the behavior of the protocol entities mediated by *Medium* which is thought of as the whole of the lower layers. *PE*1 sends messages to *PE*2 through synchronized gate *send*1 and receives messages from *PE*2 through synchronized gate *rec*1. On the other hand, *PE*2 sends and receives messages to and from *PE*1 via synchronized gates *send*2 and *rec*2, respectively.

According to Fig. 1 (b), a protocol is expressed by the following expression in LOTOS where *Medium* is defined in Sect. 3.4.

$(PE1|||PE2)|[send1, rec1, send2, rec2]|$ *Medium*

The service and the protocol specifications must be weakly bisimulation equivalent when internal actions, which occurred except that *SAPs*, are hidden from the environment.

The weak bisimulation equivalence between the service and the protocol specifications basically means that, from an external observer's point of view, the actions that occur at *SAPs* of the service specification are indistinguishable from the actions that occur at *SAPs* of the protocol specification.

3.3 Polling Mechanism

In general, protocol entities communicate in parallel via *Medium* as shown in Fig. 1 (b). Then, the order of actions in protocol and service specifications may be different when the actions at different *SAPs* were chosen by the environment. The service and the protocol may not be bisimilar, because the temporal order may not be maintained. Therefore, we have applied a mechanism to exchange a polling message between entities in order to maintain the order. Figure 3 (a) shows the LTS of a simple service specification where action a_1 occurs at *SAP*1 and action b_2 occurs at *SAP*2. Figure 3 (b) shows the LTS of the corresponding protocol specification using the polling mechanism for each node. Nodes 1 and 2 exchange the polling messages ($send1!poll$, $rec2!poll$, $send2!poll$ and $rec1!poll$)† with each other at the point where actions at different *SAPs* are possible to occur. It is easily shown that Figs. 3 (a) and 3 (b) are weakly bisimulation equivalent when we hide the polling messages. The exchange of polling messages is thus a simple but a powerful way of accomplishing the weak bisimulation equivalence. Figure 4 shows the LTS of protocol specification without polling mechanism. In this case, each action occurs at each *SAP* independently. So the LTSs shown in Figs. 3 (a) and 4 are not weakly bisimulation equivalent because the temporal order is different.

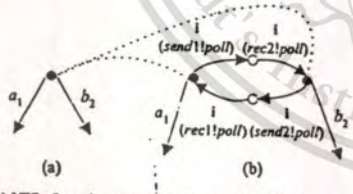


Fig. 3 (a) LTS of service specification and (b) LTS of protocol specification with polling mechanism.

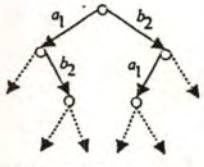


Fig. 4 LTS of protocol specification without polling mechanism.

We use the polling mechanism when our composition technique requires one to preserve the weak bisimulation equivalence between service and protocol specifications as well as when a service is decomposed into the corresponding protocol.

3.4 Summary of the Decomposition Algorithm

The basic idea of the decomposition algorithm [2] is to send either a sent or a received action signal after an observable action. We then decompose a service specification S. We assume that *Med*1 and *Med*2 are buffers within *Medium* which is able to hold one message simultaneously. A message from *PE*1 to *PE*2 is sent through synchronized gate *send*1 and received at synchronized gate *rec*2. A message from *PE*2 to *PE*1 is sent via synchronized gate *send*2 and received at synchronized gate *rec*1. *Medium* is defined as follows,

$Medium = Med1|||Med2$

where $Med1 = \sum\{send1!m; rec2!m; Med1 | m \in M\}$

$Med2 = \sum\{send2!m; rec1!m; Med2 | m \in M\}$

where M is the universe of messages.

In this algorithm, we identify specifications of protocol entities by T1act(S), T1pas(S), T2act(S), and T2pas(S). Here, T1act(S) and T1pas(S) correspond to *PE*1, T2act(S) and T2pas(S) correspond to *PE*2. T1act(S) and T1pas(S) are not independent of each other. If an action in T1act(S) has been executed, T1pas(S) should be notified in order to produce the appropriate behavior after the action. This notification is done by a message on synchronization via a synchronization gate. T1act(S) is a process that contains sending actions and T1pas(S) is a process that contains receiving actions. T2act(S) and T2pas(S) are similarly explained.

As we do not treat any internal actions in this paper, we omit the decomposition that contains the internal action i. The decomposition algorithm is shown as follows. We assume that an expression S in an action-prefix form has been given. This algorithm decomposes S into T1act(S), T1pas(S), T2act(S), and T2pas(S), where actions of node 1 are a_i ($i \in$ I) and actions of node 2 are b_j ($j \in$ J).

Decomposition Algorithm

$S = \sum\{a_i; A_i | i \in I\} [] \sum\{b_j; B_j | j \in J\}$

Then,

$\mathbf{T1act}(S) = \sum\{a_i; send1!m_i; \mathbf{T1pas}(A_i) | i \in I\}$
$[]send1!poll; (\sum\{rec1!m_j; \mathbf{T1act}(B_j) | j \in J\}$
$[]rec1!poll; \mathbf{T1act}(S))$

$\mathbf{T1pas}(S) = \sum\{rec1!m_j; \mathbf{T1act}(B_j) | j \in J\}$
$[]rec1!poll; (\sum\{a_i; send1!m_i; \mathbf{T1pas}(A_i) | i \in I\}$

†In LOTOS, when processes exchange message, *message*, by synchronizing at the gate, *gate*, it is represented as *gate!message*. This is a LOTOS syntax.

$[]send1!poll; \mathrm{T1pas}(S))$

$\mathrm{T2act}(S) = \sum\{b_j; send2!m_j; \mathrm{T2pas}(B_j) | j \in J\}$

$[]send2!poll; (\sum\{rec2!m_i; \mathrm{T2act}(A_i) | i \in I\}$

$[]rec2!poll; \mathrm{T2act}(S))$

$\mathrm{T2pas}(S) = \sum\{rec2!m_i; \mathrm{T2act}(A_i) | i \in I\}$

$[]rec2!poll; (\sum\{b_j; send2!m_j; \mathrm{T2pas}(B_j) | j \in J\}$

$[]send2!poll; \mathrm{T2pas}(S))$ □

[Theorem 1] Let S be a process in an action-prefix form and $\mathrm{T1act}(S)$ and $\mathrm{T2pas}(S)$ defined by the decomposition algorithm. Then

$S \approx$ **hide** $send1, rec1, send2, rec2$ **in**

$(\mathrm{T1act}(S) \;|||\; \mathrm{T2pas}(S))$

$|[send1, rec1, send2, rec2]| Medium$ □

<Example of decomposition>

We apply the decomposition algorithm to a simple service specification as described below.

Given $S = a_1; b_2;$ stop, $S' = b_2;$ stop, and $S'' =$ stop

The result of decomposition of the service is shown as follows.

- $\mathrm{T1act}(S) = a_1; send1!a_1; \mathrm{T1pas}(S')$

  $[]send1!poll; rec1!poll; \mathrm{T1act}(S)$
- $\mathrm{T1pas}(S') = rec1!b_2; \mathrm{T1act}(S'')$

  $[]rec1!poll; send1!poll; \mathrm{T1pas}(S')$

$\mathrm{T1act}(S'') = send1!poll; rec1!poll; \mathrm{T1act}(S'')$

$\mathrm{T2pas}(S) = rec2!a_1; \mathrm{T2act}(S')$

$[]rec2!poll; send2!poll; \mathrm{T2pas}(S)$

$\mathrm{T2act}(S') = b_2; send2!b_2; \mathrm{T2pas}(S'')$

$[]send2!poll; rec2!poll; \mathrm{T2act}(S')$

$\mathrm{T2pas}(S'') = rec2!poll; send2!poll; \mathrm{T2pas}(S'')$ □

4. Composition Method

In this section, we propose our composition method for service and protocol specifications. First, outline of composition of specifications is described. Next, the characteristics of the composition method are given followed by the composition methods.

4.1 Outline of Composition

In our composition method, we assume that protocol specifications $P1$ and $P2$ are derived from the service specifications $S1$ and $S2$, respectively by applying the decomposition algorithm described in Sect. 3.4. Under this assumption, $S1$ and $S2$, as well as $P1$ and $P2$ are composed. Then, the composition of the service and the protocol specifications would be weakly bisimilar and is described as follows (see Fig. 5).

$S1 * S2 \approx$ **hide** $send1, rec1, send2, rec2$ **in**

$((\mathrm{T1act}(S1) * \mathrm{T1act}(S2)) |||(\mathrm{T2pas}(S1) *$

$\mathrm{T2pas}(S2)))\ |[send1, rec1, send2, rec2]|\ Medium$

Here, the symbol * represents the LOTOS operator which can be any of the following: enabling, choice, parallel, or disabling. The enabling ">>" can be used for serializing the phases, the choice "[]" for selecting features, the parallel "|[G]|" for combining functions executed in parallel, and the disabling "[>" for disconnection or interruption.

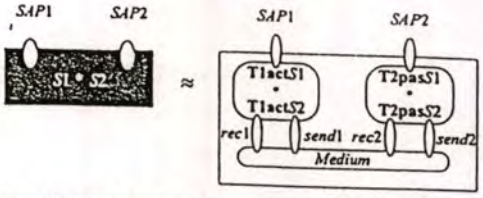


Fig. 5 Relation between the composed service and protocol specifications.

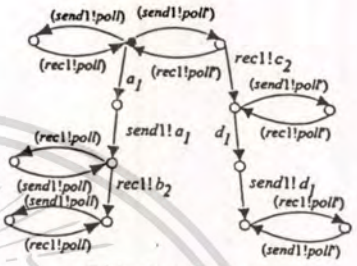


Fig. 6 Protocol entity 1.

4.2 Characteristics of Composition Method

In this section, we explain the preconditions and the characteristics of composition and decomposition of the specifications. In order to manifest that the composition of the protocols can be done without any contradiction, it is important to keep the weak bisimulation equivalence between the composition of the service specifications which represent the external behaviors of the system and the composition of the protocol specifications which represent how to provide the services in the system. The protocol specifications are derived by the decomposition of the service specifications using the Langerak's decomposition algorithm. In case of decomposition in the asynchronous model, maintaining the order of actions is controlled by sending and receiving the synchronization actions through the medium between the entities.

The expression S as the object of the decomposition means the service which may contain some choices between different nodes. Therefore the polling messages should be added to the derived protocol specification to maintain the weak bisimulation equivalence with the service specification. Two service specifications, which correspond to the protocol specifications $P1$ and $P2$, have been considered. The polling messages exchanged between entities of $P1$ and the polling messages exchanged between entities of $P2$ are distinguished as *poll* and *poll'*, respectively, as shown in Figs. 6, 7 and 8. And by doing so, the composition method becomes more simple.

The basic idea of the composition of service and proto-

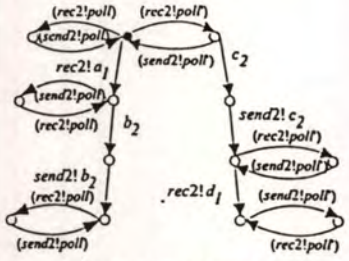


Fig. 7 Protocol entity 2.

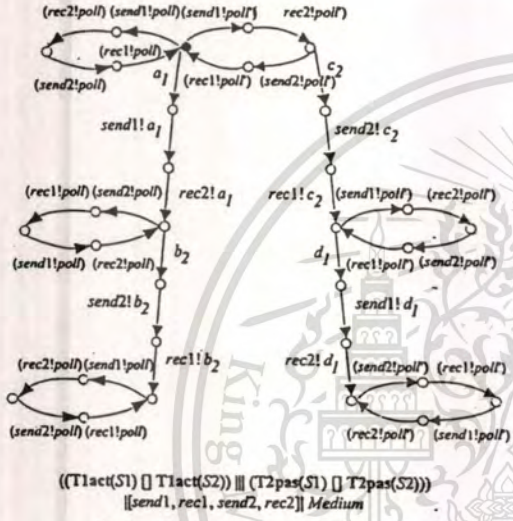


Fig. 8 LTS of resultant protocol specification.

col specifications is described as follows.

Given the service specifications $S1$ and $S2$ corresponding to the following protocol specifications $P1$ and $P2$ respectively, we have;

$P1$ = (T1act($S1$)|||T2pas($S1$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

$P2$ = (T1act($S2$)|||T2pas($S2$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

$S1$ and $S2$ are weakly bisimulation equivalent to $P1$ and $P2$, respectively when we hide the notification actions *send*1, *rec*1, *send*2 and *rec*2, and are shown as follows.

$S1 \approx$ hide *send*1, *rec*1, *send*2, *rec*2 in $P1$

$S2 \approx$ hide *send*1, *rec*1, *send*2, *rec*2 in $P2$

In order to compose the protocol specifications corresponding to the service specifications under the preconditions mentioned above, we compose the entities that correspond to the same node (*SAP*) in $P1$ and $P2$. Composition of service and protocol specification has been done as follows.

Service specification:

$S1 * S2$

Protocol specification:

((T1act($S1$) * T1act($S2$))|||(T2pas($S1$)*
T2pas($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The composition of the services should become weakly bisimulation equivalent to the composition of the protocols as shown below when the notification actions *send*1, *rec*1, *send*2, and *rec*2 are hidden.

$S1 * S2 \approx$ hide *send*1, *rec*1, *send*2, *rec*2 in
((T1act($S1$) * T1act($S2$))|||(T2pas($S1$)*
T2pas($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The symbol "*"means "[]", ">>", or "|[G]|" in the LOTOS operators. The protocol composition methods which are choice, enabling, and parallel compositions agree with the expression mentioned above. The polling messages, which have been kept during the decomposing service specification, work well in maintaining equivalency when the choice between different nodes exists.

<Example of composition>

We show an example where the composition operator is the choice. Service specifications $S1$ and $S2$ are given as follows.

$S1 = a_1; b_2;$ stop

$S2 = c_2; d_1;$ stop

After decomposition of services into protocol entities, we compose the entities at the same node together. The entities of each node, node 1 and node 2, are shown in Figs. 6 and 7, respectively. The resultant protocol specification is shown in Fig. 8.

Note that an action subscript i ($i = 1, 2$), such as a_1 and c_2, represents the action executed at the SAP i (node i). This notation is used throughout the rest of the paper. □

As shown in this example, in order to make the composition methods more general and simpler, several polling messages are introduced in protocol specification. However, unnecessary polling messages can be removed without any problem. For instance, in Fig. 6, polling messages can be removed without any effect to the system except polling messages between actions a_1 and $rec1!c_2$. If the polling messages are not for the choice between different nodes, we can remove them.

In case of disabling, the composition of the services and the protocols generally does not become weakly bisimulation equivalent by using the above expression.

Disabling is the composition that a service can interrupt every state of another service. The polling messages, which have been kept during the decomposition of the service specification, work to maintain the equivalency only in the case of choice between the initial actions. Therefore, the other composition method needs to be applied. We convert the disabling expression into an expression that contains choices. The derivation process of disabling method is described in Sect. 4.3.4. Intermediate behavior expressions P and Q, which correspond to node 1 and node 2, respectively, have been created for disabling composition as follows.

$S1$ [> $S2 \approx$ hide *send*1, *rec*1, *send*2, *rec*2 in
(P ||| Q) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

4.3 Composition Methods

In this section, four types of the composition methods corresponding to the LOTOS operators, ">>", "|[G]|", "[]", and "[>" are shown.

Due to the introduction of polling messages, the composition methods ensure that the protocol specification and the service specification are weakly bisimilar. The proof of the correctness of the composition methods can be done by the expansion technique [2]. In the expansion technique, the protocol specification which consists of parallel composition of protocol entities and the medium is expanded (flattened) using the inference rules of the LOTOS parallel operator until the protocol specification cannot be expanded [3].

4.3.1 Enabling

Enabling composition between two service specifications $S1$ and $S2$ is shown as "$S1$ >> $S2$" using the LOTOS operator ">>". This means that after $S1$ normally completed, $S2$ occurs. So we directly compose the protocol entities at the same node of $P1$ and $P2$, and the equivalency is maintained.

[Method 1]

The following protocol specifications $P1$ and $P2$ are derived from the service specifications $S1$ and $S2$ by the decomposition algorithm.

$P1$ = (T1act($S1$)|||T2pas($S1$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*
$P2$ = (T1act($S2$)|||T2pas($S2$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

$S1$ and $S2$ are weakly bisimulation equivalent to $P1$ and $P2$, respectively when the notification actions *send*1, *rec*1, *send*2, and *rec*2 are hidden and are shown below.

$S1$ ≈ hide *send*1, *rec*1, *send*2, *rec*2 in $P1$
$S2$ ≈ hide *send*1, *rec*1, *send*2, *rec*2 in $P2$

The compositions of the services and the protocols are executed as follows.

Service: $S1$ >> $S2$
Protocol: ((T1act($S1$) >> T1act($S2$)) |||
(T2pas($S1$) >> T2pas($S2$)))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium* □

<Example of Method 1>

Service specifications $S1$ and $S2$ are given as follows.

$S1 = a_1; b_2;$ exit, $S1' = b_2;$ exit, $S1''$ = exit
$S2 = c_2; d_1;$ stop, $S2' = d_1;$ stop, $S2''$ = stop

We decompose them and get the protocol specifications, $P1$ and $P2$.

$P1$ = (T1act($S1$)|||T2pas($S1$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*
T1act($S1$) = a_1; *send*1!a_1; T1pas($S1'$)
[]*send*1!*poll*; *rec*1!*poll*; T1act($S1$)
T1pas($S1'$) = *rec*1!b_2; T1act($S1''$)
[]*rec*1!*poll*; *send*1!*poll*; T1pas($S1'$)
T1act($S1''$) = *send*1!*poll*; *rec*1!*poll*; T1act($S1''$)
T2pas($S1$) = *rec*2!a_1; T2act($S1'$)
[]*rec*2!*poll*; *send*2!*poll*; T2pas($S1$)
T2act($S1'$) = b_2; *send*2!b_2; T2pas($S1''$)
[]*send*2!*poll*; *rec*2!*poll*; T2act($S1'$)
T2pas($S1''$) = *rec*2!*poll*; *send*2!*poll*; T2pas($S1''$)

$P2$ = (T1act($S2$)|||T2pas($S2$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*
T1act($S2$) = *send*1!*poll'*; (*rec*1!c_2; T1act($S2'$)
[]*rec*1!*poll'*; T1act($S2$))
T1act($S2'$) = d_1; *send*1!d_1; T1pas($S2''$)
[]*send*1!*poll'*; *rec*1!*poll'*; T1act($S2'$)
T1pas($S2''$) = *rec*1!*poll'*; *send*1!*poll'*; T1pas($S2''$)
T2pas($S2$) = *rec*2!*poll'*; (c_2; *send*2!c_2; T2pas($S2'$)
[]*send*2!*poll'*; T2pas($S2$))
T2pas($S2'$) = *rec*2!d_1; T2act($S2''$)
[]*rec*2!*poll'*; *send*2!*poll'*; T2pas($S2'$)
T2act($S2''$) = *send*2!*poll'*; *rec*2!*poll'*; T2act($S2''$)

Services and protocols are weakly bisimilar when we hide the notification actions.

$S1$ ≈ hide *send*1, *rec*1, *send*2, *rec*2 in $P1$
$S2$ ≈ hide *send*1, *rec*1, *send*2, *rec*2 in $P2$

Then, we get the compositions of the specifications by applying Method 1 as follows.

Service specification:
$S1$ >> $S2$ = $a_1; b_2;$ exit >> $c_2; d_1;$ stop
Protocol specification:
((T1act($S1$) >> T1act($S2$))|||(T2pas($S1$) >>
T2pas($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The compositions of the services and protocols are weakly bisimilar when we hide the notification actions.

$S1$ >> $S2$ ≈ hide *send*1, *rec*1, *send*2, *rec*2 in
((T1act($S1$) >> T1act($S2$))|||(T2pas($S1$) >>
T2pas($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

□

4.3.2 Parallel

If G is a set of the synchronization actions at $S1$ and $S2$, the parallel composition between two service specifications $S1$ and $S2$ is shown as "$S1$ |[G]| $S2$" using the LOTOS operator "|[G]|" and $G = Act(S1) \cap Act(S2)$. If $G = \emptyset$, $S1$ and $S2$ become an asynchronous parallel composition, and if $G \neq \emptyset$, they become a synchronous parallel composition synchronized at the actions in G. In case that $S1$ and $S2$ have no synchronization actions, the actions of $S1$ and $S2$ can occur independently with any order. On the other hand, in case that $S1$ and $S2$ have some synchronization actions, these actions must occur synchronously at $S1$ and $S2$, and the other actions occur independently. In the case of protocol composition, we can compose the protocol entities in parallel at the same node of $P1$ and $P2$, respectively. If $P1$ and $P2$ have some synchronization actions, they will be synchronously composed in their actions and notification actions.

[Method 2]

The precondition is the same as Method 1. When we express the synchronization actions as $G = Act(S1) \cap Act(S2)$, $G1 = Act(\mathrm{T1act}(S1)) \cap Act(\mathrm{T1act}(S2))$ and $G2 = Act(\mathrm{T2pas}(S1)) \cap Act(\mathrm{T2pas}(S2))$, we execute the parallel composition of the services and the protocols as follows.

Service: $S1$ |[G]| $S2$

Protocol: ((T1act($S1$) |[$G1$]| T1act($S2$)) |||
(T2pas($S1$) |[$G2$]| T2pas($S2$)))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium* □

<Example of Method 2>

Service specifications $S1$ and $S2$ are given as follows.

$S1 = a_1; b_2;$ stop, $S1' = b_2;$ stop, $S1'' =$ stop

$S2 = c_2; a_1;$ stop, $S2' = a_1;$ stop, $S2'' =$ stop

We decompose them and get the protocol specifications, P1 and P2.

$P1$ = (T1act($S1$)|||T2pas($S1$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*
T1act($S1$) = a_1; *send*1!a_1; T1pas($S1'$)
[]*send*1!*poll*; *rec*1!*poll*; T1act($S1$)
T1pas($S1'$) = *rec*1!b_2; T1act($S1''$)
[]*rec*1!*poll*; *send*1!*poll*; T1pas($S1'$)
T1act($S1''$) = *send*1!*poll*; *rec*1!*poll*; T1act($S1''$)
T2pas($S1$) = *rec*2!a_1; T2act($S1'$)
[]*rec*2!*poll*; *send*2!*poll*; T2pas($S1$)
T2act($S1'$) = b_2; *send*2!b_2; T2pas($S1''$)
[]*send*2!*poll*; *rec*2!*poll*; T2act($S1'$)
T2pas($S1''$) = *rec*2!*poll*; *send*2!*poll*; T2pas($S1''$)

$P2$ = (T1act($S2$)|||T2pas($S2$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*
T1act($S2$) = *send*1!*poll'*; (*rec*1!c_2; T1act($S2'$)
[]*rec*1!*poll'*; T1act($S2$))
T1act($S2'$) = a_1; *send*1!a_1; T1pas($S2''$)
[]*send*1!*poll'*; *rec*1!*poll'*; T1act($S2'$)
T1pas($S2''$) = *rec*1!*poll'*; *send*1!*poll'*; T1pas($S2''$)
T2pas($S2$) = *rec*2!*poll'*; (c_2; *send*2!c_2; T2pas($S2'$)
[]*send*2!*poll'*; T2pas($S2$))
T2pas($S2'$) = *rec*2!a_1; T2act($S2''$)
[]*rec*2!*poll'*; *send*2!*poll'*; T2pas($S2'$)
T2act($S2''$) = *send*2!*poll'*; *rec*2!*poll'*; T2act($S2''$)

Services and protocols are weakly bisimilar when we hide the notification actions.

$S1 \approx$ hide *send*1, *rec*1, *send*2, *rec*2 in $P1$
$S2 \approx$ hide *send*1, *rec*1, *send*2, *rec*2 in $P2$

Then, we get the compositions of the specifications by applying Method 2 as follows.

Service specification:

$S1$ |[a_1]| $S2 = a_1; b_2;$ stop |[a_1]| $c_2; a_1;$ stop

Protocol specification:

((T1act($S1$) |[a_1, *send*1!a_1]| T1act($S2$)) |||
(T2pas($S1$) |[*rec*2!a_1]| T2pas($S2$)))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The compositions of the services and protocols are weakly bisimilar when we hide the notification actions.

$S1$ |[a_1]| $S2 \approx$ hide *send*1, *rec*1, *send*2, *rec*2 in
((T1act($S1$) |[a_1, *send*1!a_1]| T1act(S_2)) |||
(T2pas($S1$) |[*rec*2!$a1$]| T2pas($S2$)))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium* □

4.3.3 Choice

Choice composition between two service specifications $S1$ and $S2$ is shown as "$S1$[]$S2$" by using the LOTOS operator "[]". This means that either $S1$ or $S2$ is selected. The polling messages, which have been kept when decomposing the service specifications, work to maintain the equivalency.

[Method 3]

The precondition is the same as Method 1. We execute the choice composition of the services and the protocols as follows.

Service: $S1$ [] $S2$

Protocol: ((T1act($S1$)[]T1act($S2$))|||(T2pas($S1$)[]
T2pas($S2$))) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

□

<Example of Method 3>

Service specifications $S1$ and $S2$ are given as follows.

$S1 = a_1; b_2;$ stop, $S1' = b_2;$ stop, $S1'' =$ stop

$S2 = c_2; d_1;$ stop, $S2' = d_1;$ stop, $S2'' =$ stop

We decompose them and get the protocol specifications, $P1$ and $P2$.

$P1$ = (T1act($S1$)|||T2pas($S1$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*
T1act($S1$) = a_1; *send*1!a_1; T1pas($S1'$)
[]*send*1!*poll*; *rec*1!*poll*; T1act($S1$)
T1pas($S1'$) = *rec*1!b_2; T1act($S1''$)
[]*rec*1!*poll*; *send*1!*poll*; T1pas($S1'$)
T1act($S1''$) = *send*1!*poll*; *rec*1!*poll*; T1act($S1''$)
T2pas($S1$) = *rec*2!a_1; T2act($S1'$)
[]*rec*2!*poll*; *send*2!*poll*; T2pas($S1$)
T2act($S1'$) = b_2; *send*2!b_2; T2pas($S1''$)
[]*send*2!*poll*; *rec*2!*poll*; T2act($S1'$)
T2pas($S1''$) = *rec*2!*poll*; *send*2!*poll*; T2pas($S1''$)

$P2$ = (T1act($S2$)|||T2pas($S2$))
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*
T1act($S2$) = *send*1!*poll'*; (*rec*1!c_2; T1act($S2'$)
[]*rec*1!*poll'*; T1act($S2$))
T1act($S2'$) = d_1; *send*1!d_1; T1pas($S2''$)
[]*send*1!*poll'*; *rec*1!*poll'*; T1act($S2'$)
T1pas($S2''$) = *rec*1!*poll'*; *send*1!*poll'*; T1pas($S2''$)
T2pas($S2$) = *rec*2!*poll'*; (c_2; *send*2!c_2; T2pas($S2'$)
[]*send*2!*poll'*; T2pas($S2$))
T2pas($S2'$) = *rec*2!d_1; T2act($S2''$)
[]*rec*2!*poll'*; *send*2!*poll'*; T2pas($S2'$)
T2act($S2''$) = *send*2!*poll'*; *rec*2!*poll'*; T2act($S2''$)

Services and protocols are weakly bisimilar when we hide the notification actions.

$S1 \approx$ hide *send*1, *rec*1, *send*2, *rec*2 in $P1$
$S2 \approx$ hide *send*1, *rec*1, *send*2, *rec*2 in $P2$

Then, we get the compositions of the specifications by applying Method 3 as follows.

Service specification:

$S1[]S2 = a_1; b_2;$ stop$[]c_2; d_1;$ stop

Protocol specification: (See in Figs. 6, 7 and 8)

((T1act($S1$)[]T1act($S2$))|||(T2pas($S1$)[]
T2pas($S2$)))|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

The compositions of the services and protocols are weakly bisimilar when we hide the notification actions.

$S1[]S2$ = hide *send*1, *rec*1, *send*2, *rec*2 in
((T1act($S1$)[]T1act($S2$))|||(T2pas($S1$)[]
T2pas($S2$)))|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

□

4.3.4 Disabling

Disabling composition between two service specifications $S1$ and $S2$ is shown as "$S1$ [> $S2$" by using the LOTOS operator "[>". The choice composition method cannot be applied to disabling directly, because the equivalency is not maintained. Then we need to consider another method. We convert the expression of disabling into another one consisting of only the choice, and consider the disabling method as a repetition of choice. Before going to the method, we show the process of developing the disabling method with an example.

[Derivation Process of Disabling Method]

Assume the following service specifications $S1$ and $S2$.

$S1 = a_1; b_2;$ stop
$S2 = c_2; d_1;$ stop
$B = d_1;$ stop

Let S be the following service composed by disabling.

$S = S1$ [> $S2 = a_1; b_2;$ stop [> $c_2; B$

We can convert the above expression into another one consisting of only the choice shown below.

$S = a_1; (b_2; ($stop$[]c_2; B)[]c_2; B)[]c_2; B$

We divide the above expression into each choice expression as follows.

$A_0 = a_1; A_1[]c_2; B$
$A_1 = b_2; A_2[]c_2; B$
$A_2 =$ stop$[]c_2; B = c_2; B$

The protocol specifications of $S1$ and $S2$ correspond to the following $P1$ and $P2$ which have a style without the polling messages.

$P1 = (E1|||E2)$ |[*send*1, *rec*1, *send*2, *rec*2]|
Medium
$P2 = (F1|||F2)$ |[*send*1, *rec*1, *send*2, *rec*2]|
Medium

where entity correspondence is as follows.

$E1$ and $F1$ are entities at node1.
$E2$ and $F2$ are entities at node2.

The specifications of each entity are as follows.

$E1 = a_1;$ *send*1!$a_1;$ *rec*1!$b_2;$ stop
$E2 =$ *rec*2!$a_1; b_2;$ *send*2!$b_2;$ stop
$F1 =$ *rec*1!$c_2; d_1;$ *send*1!$d_1;$ stop
$F2 = c_2;$ *send*2!$c_2;$ *rec*2!$d_1;$ stop

Generally, the weak bisimulation equivalence is not preserved by a straightforward composition between protocol entities corresponding to the same node of $S1$ and $S2$.

$S1$ [> $S2 \neq$ hide *send*1, *rec*1, *send*2, *rec*2 in
($(E1$ [> $F1)$ ||| $(E2$ [> $F2)$)
|[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

So, we introduce intermediate behavior expressions P and Q which satisfy the next formula.

$S1$ [> $S2 \approx$ hide *send*1, *rec*1, *send*2, *rec*2 in
(P ||| Q) |[*send*1, *rec*1, *send*2, *rec*2]| *Medium*

P and Q correspond to the protocol entities at node 1 and node 2, respectively.

In order to derive P and Q, we consider the decomposition of A_0. A_0 is a choice between a_1 and c_2 at the different nodes. We must add the polling messages. The parts of the protocol specification corresponding to a_1 and c_2 at node 1 are "$a_1;$ *send*1!a_1" and "*rec*1!c_2", respectively and the parts of the protocol specification corresponding to a_1 and c_2 at node 2 are "*rec*2!a_1" and "$c_2;$ *send*2!c_2", respectively. Therefore, the intermediate behavior expressions corresponding to A_0 are the following.

$P = a_1;$ *send*1!$a_1; P_1[]$*send*1!*poll*;
(*rec*1!$c_2; B[]$*rec*1!*poll*; P)
$Q =$ *rec*2!$a_1; Q_1[]$*rec*2!*poll*;
($c_2;$ *send*2!$c_2; B[]$*send*2!*poll*; Q)

Next, we consider the decomposition of A_1. A_1 is a choice between b_2 and c_2 at the same node. We do not need to add the polling message. The parts of the protocol specification corresponding to b_2 and c_2 at node 1 are "*rec*1!b_2" and "*rec*1!c_2", respectively, and the parts of the protocol specification corresponding to b_2 and c_2 at node 2 are "$b_2;$ *send*2!b_2" and "$c_2;$ *send*2!c_2", respectively. Therefore, the intermediate behavior expressions corresponding to A_1 are the following.

$P_1 =$ *rec*1!$b_2; P2[]$*rec*1!$c_2; B$
$Q_1 = b_2;$ *send*2!$b_2; Q_2[]c_2;$ *send*2!$c_2; B$

Next, we consider the decomposition of A_2. A_2 does not include choice. So, the intermediate behavior expressions corresponding to A_2 consist of only the actions related with c_2 as follows.

$P_2 =$ *rec*1!$c_2; B$
$Q_2 = c_2;$ *send*2!$c_2; B$

Finally, we get the intermediate behavior expressions P and Q by substituting P_2 and Q_2 for P_1 and Q_1 respectively, and by substituting P_1 and Q_1 for P and Q respectively. Namely, when we execute the disabling composition, we convert the expression of disabling composition into another one consisting of only the choice operators, then we derive P and Q, and we compose P and Q finally. □

[Method 4]

When we execute the disabling composition, we convert the expression of disabling composition into another one consisting of only the choice operators, then we derive P and Q, and finally compose P and Q.

$P1$ and $P2$ shown below are derived from the service specifications $S1$ and $S2$ by applying the Langerak's decomposition algorithm without polling messages.

$P1 = (E1$ ||| $E2)$ |[*send*1, *rec*1, *send*2, *rec*2]|

$$P2 = (F1 \;|||\; F2) \;|[send1, rec1, send2, rec2]|\; Medium$$

where the correspondence of entities is shown below.

$E1$ and $F1$ are entities at node1.

$E2$ and $F2$ are entities at node2.

Based on the premise mentioned above, we execute the disabling composition of the services and the protocols as follows.

Service: $S1$ [> $S2$

Protocol: $(P \;|||\; Q) \;|[send1, rec1, send2, rec2]|\; Medium$

The intermediate behavior expressions P and Q are derived by repetition of the procedure shown below.

If InitM($E1$) ∪ InitM($F1$) ⊆ ActR($N1$) or
InitM($E1$) ∪ InitM($F1$) ⊆ ActR($N2$)
then P = InitM($E1$); (Der($E1$)[> $F1$)[]$F1$
else P = InitM($E1$); (Der($E1$)[> $F1$)
[]$send1!poll$; ($F1$[]$rec1!poll$; P)
If InitM($E2$) ∪ InitM($F2$) ⊆ ActR($N1$) or
InitM($E2$) ∪ InitM($F2$) ⊆ ActR($N2$)
then Q = InitM($E2$); (Der($E2$)[> $F2$)[]$F1$
else Q = InitM($E2$); (Der($E2$)[> $F2$)
[]$rec2!poll$; ($F2$[]$send2!poll$; Q)

where Der($E1$) [> $F1$ means executing this procedure once again with the substition of $E1$ by Der($E1$), and Der($E2$) [> $F2$ means executing this procedure once again with the substition of $E2$ by Der($E2$).

The functions in this method are defined as follows.

ActR(Ni) is the set of the actions which can occur at node i and its notification actions related to these actions.

InitM(E) is the set of the pair of the actions which can occur at first in entity E and its transmission notification actions, or the set of the receiving notification actions.

Der(E) is the set of the behavior expressions immediately after the execution of the InitM(E). □

<Example of each function>

If $E = a_1; send1!a_1; rec1!b_2;$ stop
then InitM(E) = {$a_1; send1!a_1$}
Der(E) = {$rec1!b_2$; stop}
ActR($N1$) = {$a_1; send1!a_1$}
ActR($N2$) = {$rec1!b_2$} □

5. Support System

The support system has five working stages; those are (i) entering service specifications ($S1$ and $S2$), (ii) the selection of the composition operator, (iii) the decomposition, (iv) the result of composition, and (v) LTS of composition, as shown in Fig. 9.

At the first stage, a user enters service specifications through the service specification windows and may save them as text files and reload them to specify again. One different point from LOTOS in an action prefix form is that a node number is put within "[]". After the user enters $S1$, the support system will check the syntax of expression based on the LOTOS syntax. If $S1$ has syntax errors, the support system will show these errors. If $S1$ has no errors, the support system permits the user to enter $S2$. Entering $S2$ is the same as the case of $S1$.

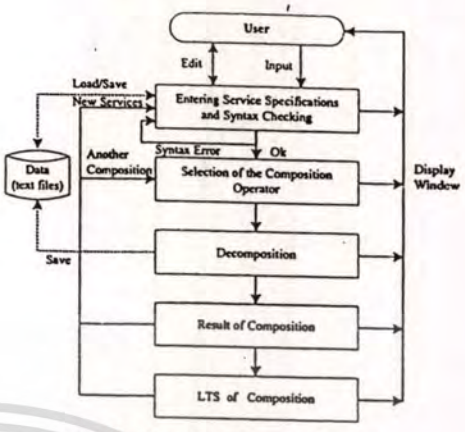


Fig. 9 Flowchart of the support system.

At the second stage, the support system shows the composition method window for the user to select the LOTOS operator, which can be one of four types of composition operators.

At the third stage, the support system decomposes service specifications into protocol specifications automatically and shows the protocol specification window which includes protocol specifications of $S1$ and $S2$.

At the fourth stage, the support system shows that the composition of the services and the composition of protocols are weakly bisimilar when notification actions (internal actions) are hidden. The support system provides the user with the selection of another composition, the composition of new services, or LTS.

At the last stage, if LTS is selected, the support system will show LTS of the composed services.

<Application example>

We apply the proposed composition method and the support system to construct service and protocol specifications for Bulk Data Transfer part of the FTAM (File Transfer, Access and Management) [9], [10] OSI Application Layer as an example for their evaluation.

FTAM has many services such as F-READ, F-WRITE, F-TRANSFER-END, F-DATA, F-CANCEL, etc. For this application, we consider two alternative services of data transfer in FTAM, i.e., F-READ and F-WRITE services. In FTAM service, there are two associated users, *Initiator* and *Responder*. The Initiator requests a service from the Responder, and then the Responder replies to the request. After the Initiator executes an F-READ request (F-WRITE request), the F-READ service (F-WRITE service) begins.

The F-READ service specifies a data transfer from the Responder at a node to the Initiator at another node. The data will be transferred until the transfer is completed. The

Responder executes an F-DATA-END request to inform the Initiator that the data transfer is completed. Then, the Initiator confirms the completion using an F-TRANSFER-END service. After that, the F-READ service is finished.

Similarly, the F-WRITE service specifies a data transfer from the Initiator to the Responder. The data will be transferred until the transfer is completed. The Initiator executes an F-DATA-END request to inform the Responder that the data transfer is completed. Then, the Initiator confirms the completion using an F-TRANSFER-END service. After that, the F-WRITE service is finished.

The LTSs of the F-READ and F-WRITE service specifications are shown in Fig. 10. The F-READ service will start when a read request signal (F-Rd_Req1) is issued, and the F-WRITE service will start when a write request signal (F-Wt_Req1) is issued. At first, the support system shows the asynchronous protocol model to the user as shown in Fig. 11. The user just puts the F-READ and F-WRITE service specifications into the support system as shown in Fig. 12. Then, the support system will ask the user to select one of the four composition operators (choice, enabling, parallel and disabling) as shown in Fig. 13. The user selects the choice operator because the two services are composed alternatively. Then, as shown in Fig. 14, the protocol specification is automatically derived (composed) from the composed service specification, in accordance with the composition method. Finally, it is displayed that the composed service specification and the composed protocol specification are weakly bisimilar when the notification actions are hidden, as shown in Fig. 15.

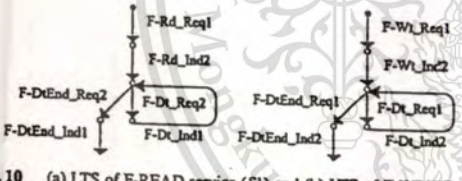


Fig. 10 (a) LTS of F-READ service ($S1$) and (b) LTS of F-WRITE service ($S2$).

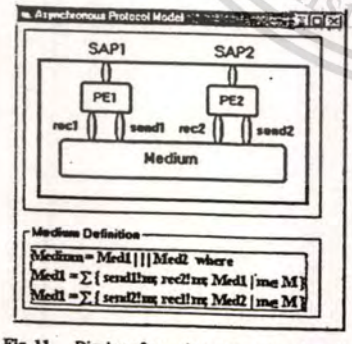


Fig. 11 Display of asynchronous protocol model.

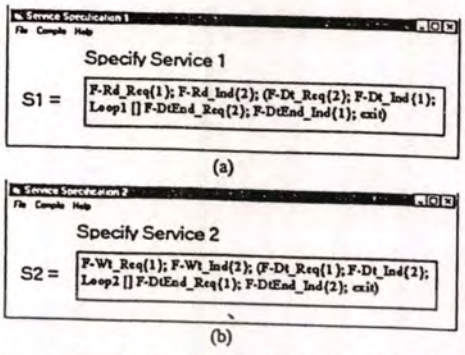


Fig. 12 (a) Input of service specification $S1$ and (b) Input of service specification $S2$.

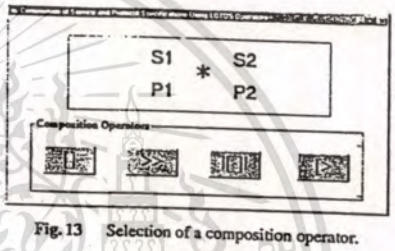


Fig. 13 Selection of a composition operator.

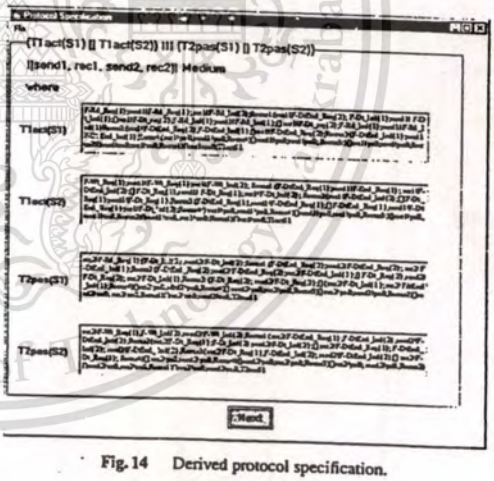


Fig. 14 Derived protocol specification.

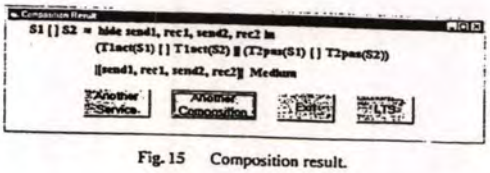


Fig. 15 Composition result.

Similarly, other services such as F-TRANSFER-END and F-CANCEL can be also composed, and the corresponding equivalent protocol specification can be derived, using the proposed composition method and the support system.

Finally, it is noted that protocol specifications derived by applying the proposed method are somewhat complex since they contain some redundant polling messages. However, as suggested in Sect. 4.2, they could be removed (though it is one of our future works), thereby relaxing the problem of efficiency.

6. Related Work

We briefly describe some closely related work in this section.

In [3], service and protocol specifications, which perform a several kind of functions, are combined together. The approach considers synchronous communication model, i.e., the communicating entities reside in the same system. Unlike the present work, they do not consider entities residing in different locations, i.e., the asynchronous communication model.

There are studies similar to the present work, but they differ in approach and purpose. References [4] and [5] propose approaches to derive the specification of n protocol entities from a service specification. This is basically a specification transformation but not specification integration and transformation. Unlike our approach, any addition of a component service specification cannot be handled.

The authors in Ref. [6] propose a method to derive protocol specification from service specification using the decomposition but this approach becomes difficult when many functions and behaviors of service specifications and protocol specifications have to be considered simultaneously.

Reference [7] proposes a compositional approach in which sub-function protocols are designed independently and are combined together to obtain a composite protocol. This approach considers only protocol specifications but not service specifications. What kind of service will be provided is not clearly mentioned. The approach is based on Communicating Finite State Machine (CFSM).

Reference [8] discusses sequential, recursive and alternative composition of component service specifications to obtain an integrated service specification. The protocol is derived from the integrated service specification using transition synthesis rules. The specifications are modeled in CFSM. This approach has two steps: (a) composition of service components and then (b) transformation of the integrated service specification to a protocol specification containing two entities. Our approach is a single step approach; the composition of service specifications only. The protocol specification is automatically derived when the service specifications are combined together. In [8], after obtaining the integrated service specification by combining component service specifications, the integrated service specification is further modified in order to avoid any logical errors (unspecified reception and deadlock) in derived protocol specifications. We believe that the functionalities of service components should be maintained in the integrated (composed) service specification. Any modification, if required, should be introduced in lower level or transformed specifications, e.g., in protocol specifications. In our composition method, the component service specifications and the composed (integrated) service specification are not modified, thus maintaining the functionalities of given component service specifications in the composed service specification.

7. Conclusion

We have proposed composition method of service and protocol specifications, for four composition types called enabling, parallel, choice, and disabling, which are LOTOS operators, for the asynchronous communication model involving two SAPs (two nodes). In our previous method [3], only the synchronous model was used. However, the asynchronous model is more suitable in the practical communication networks. By applying this composition method, a good design of services and protocols has been achieved and we have developed its support system.

Because there are many redundant polling messages in composed (derived) protocol specifications, future work will involve the optimization of redundant polling messages.

Acknowledgments

A part of this research was supported by JGN-2 project and Research Grant of Kiban (B): No.6300011 from Ministry of Education, Science and Culture, Japan.

References

[1] ISO, "Information processing systems—Open systems interconnection-LOTOS-A formal description technique based on the temporal ordering of observational behavior," ISO 8807, 1989.
[2] R. Langerak, "Decomposition of functionality: A correctness preserving LOTOS transformation," in Protocol Specification, Testing and Verification, pp.203–218, 1990.
[3] B.B. Bista, K. Takahashi, and N. Shiratori, "A compositional approach for constructing communication services and protocols," IEICE Trans. Fundamentals, vol.E82-A, no.11, pp.2546–2557, Nov. 1999.
[4] G. Bochmann and R. Gotzhein, "Deriving protocol specification from service specification," Proc. SIGCOMM'86, vol.14, pp.144–156, 1986.
[5] C. Kant, T. Higashino, and G.V. Bochmann, "Deriving protocol specifications from service specifications written in LOTOS," Distributed Computing, vol.10, no.1, pp.29–47, 1996.
[6] A. Khoumsi and K. Saleh, "Two formal methods for the synthesis of discrete event systems," Computer Networks and ISDN Systems, vol.29, no.7, pp.759–780, 1997.
[7] H.A. Lin and C.L. Tarng, "An improved method for constructing multiphase communications protocols," IEEE Trans. Comput., vol.42, no.1, pp.15–26, 1993.
[8] M. Nakamura, Y. Kakuda, and T. Kikuno, "On constructing communication protocols from component-based service specifications," Comput. Commun., vol.19, pp.1200–1215, 1996.

[9] ISO, "File service definition: File transfer, access and management — Part 3," ISO 8571-3, 1989.
[10] ISO, "File protocol definition: File transfer, access and management — Part 4," ISO 8571-4, 1989.

Noppadol Maneerat received the B.Sc. degree in Physics from Srinakharinwirote University, Pitsanuloke, Thailand in 1990, the M.Eng. degree in Electrical Engineering from King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand in 1997. He is now an engineer in the Information System and Quality Assurance Division, Computer Research and Service Center, KMITL. He is currently working toward the D.Eng degree at Faculty of Engineering, KMITL. His current research interests are protocol design, concurrent system design, computer communication network, wireless communication and analog-digital communications.

Ruttikorn Varakulsiripunth received the B.E. degree in Electrical and Electronics from Kyoto University, Kyoto, Japan in 1978. He obtained his M.E. and Ph.D. degrees in Electrical and Communication Engineering from Tohoku University, Sendai, Japan in 1983 and 1986, respectively. He is now an associate professor in the Department of Electronics, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand. His current research interests are concerned with computer communication network including switching system, queueing analysis, flow and congestion control, multimedia communication, wireless communication, image processing and natural language processing.

Ehed Bahadur Bista received the B.Eng. degree in Electronics from University of York, England in 1991 and the M.S. and Ph.D. degrees in Information Science from Tohoku University, Japan in 1994 and 1997 respectively. He worked as a research associate at Miyagi University from April 1997 to March 1998. At present, he is an associate professor in Faculty of Software and Information Science, Iwate Prefectural University, Japan. His current research interests are protocol specification and synthesis, and formal description techniques. He is a member of IPSJ and IEEE.

Kaoru Takahashi received the Ph.D. degree from Tohoku University in 1992. From 1993 to 1995, he was a senior visiting researcher in the Advanced Intelligent Communication Systems Laboratories. He is now a professor in the Department of Information and Communication Engineering, Sendai National College of Technology. Currently, he is doing research on distributed systems design, Semantic Web, network security and so on. He received the 25th Anniversary Memorial Prize-Winning Paper Award of IPSJ (Information Processing Society of Japan) in 1985, the 6th Telecommunication Advancement Foundation Incorporation Award in 1991, and the Information Network Research Award of IEICE in 1997. He is a member of IPSJ.

Yasushi Kato is currently a professor and the chairman in the Department of Information Engineering, Sendai National College of Technology. His research interests include concurrent systems design, Semantic Web, multi-agent system, micro-ITRON based embedded system and education method for digital technology. He received the Ph.D. degree in Electrical and Communication Engineering from Tohoku University, Sendai in 1978. He had been a visiting researcher at Twente University, the Netherlands from 1995 to 1996 and had engaged in the research on formal description techniques. He received the achievements award of JSEE (Japanese Society for Engineering Education) in 1994. He is a member of IPSJ (Information Processing Society of Japan) and JSEE. He is also a senior visiting advisor of Miyagi, Fukushima and Iwate Prefectural Institute of Technologies.

Norio Shiratori was born in Miyagi Prefecture, Japan, on May 11, 1946. He received the B.E. degree from Tokai University, Tokyo, in 1972, and the M.E. and Ph.D. degrees in Electrical and Communication Engineering from Tohoku University, Sendai, in 1974 and 1977, respectively. He has joined RIEC (Research Institute of Electrical Communication), Tohoku University, Sendai, since 1977, and he is now a Professor of Computer Science at RIEC of Tohoku University. Professor Norio Shiratori received the 25th Anniversary of IPSJ Memorial Paper Award in 1985, the 6th Telecommunications Advancement Foundation Incorporation Award in 1991. He was awarded the Best Paper in ICOIN-11 and ICOIN-12 in 1997 and 1998, respectively, and also the IPSJ Best Paper Award in 1997, and the Best Paper Award of ICPADS in 2000. He has been named a Fellow of the IEEE for his contributions to the field of computer communication networks since 1998. He has also received the IPSJ Fellow and IEICE Fellow since 2000 and 2002, respectively.

AUTHOR BIOGRAPHY

Name Surname	Mr.Noppadol Maneerat
Date of Birth	May 11, 1968
Place of Birth	Nakornpathom Province
Education Background	Master of Engineering (Electrical), King Mongkut's Institute of Technology Ladkrabang (KMITL), 1997 Bachelor of Science (Physics), Srinakarinwirot University (Pitsanulok Campus), 1990
Work	Engineer, since 1997 – present
Office	Computer Research and Service Center (CRSC), King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok
Training	- Advance English Language, California State University, USA (March 2003 – May 2003) - Computer Network at Sendai National College of Technology, Japan (March 2002 – September 2002), JICA Scholarship - Computer Network at Sendai National College of Technology, Japan (September 2000 – March 2001), JICA Scholarship
Research Interests	- Protocol Design - Concurrent System Design - Computer Communication Network - Wireless Communication - Analog and Digital Communications