## NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

## AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

# SYNTHESIS METHODS FOR THE DESIGN AND VALIDATION OF COMMUNICATION PROTOCOLS

By

Kassem Afif Saleh

Thesis

Submitted to the School of Graduate Studies
and Research

in Partial Fulfillment of the Requirements for the
Degree of
Doctor of Engineering

in
Electrical Engineering
(Computer Science)

University of Ottawa

January 1991

ISBN 0-315-68083-0

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

وَقُلْ رَبِّ زِدْنِي عِلْمًا

## ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my thesis supervisor Professor Robert L. Probert for his support, encouragement and fruitful discussions throughout my thesis research. Many thanks also go to Professors Sunil Das, Hasan Ural and Murray Woodside of my thesis advisory committee for many helpful remarks and suggestions during the progress of my work. Special thanks go to my wife Maha for her support and patience that were very valuable to finish this work, and to my little daughter May who was generally comprehensive in not interrupting my short nights after long and tiring work days. My deepest respect and love to my father Afif who inspired me and gave me the moral support and encouragement to further my education.

Finally, I gratefully acknowledge partial financial support by the Telecommunications Research Institute of Ontario (TRIO) and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

## ABSTRACT

Communication protocol design consists essentially of the construction of interacting protocol entities which cooperate to provide a set of specified services to the service users. Protocol validation or design verification is a pre-implementation phase in the protocol development process which is essential for the detection of design errors such as deadlocks and incompleteness (unspecified receptions). The conventional analytic approach to protocol design consists of iteratively specifying a protocol in an informal manner, then validating it by analytic methods and correcting it until it becomes error-free. The synthetic approach consists of the automatic or semi-automatic (interactive) construction of the communicating protocol entities using a synthesis method which guarantees the correctness of the resulting protocol with respect to freedom from design errors and provision of the specified service.

In this thesis, we develop service-oriented synthesis methods for the design and validation of communications protocols. First, we introduce an efficient, localized, synthetic technique for the validation of an existing protocol design. The technique can also be extended to complete an erroneous protocol design so as to satisfy the service specification.

Then, after providing the rationale for designing protocols starting from service specifications, we introduce an automatic, service-oriented synthesis method for the design of communication protocols. Given the services a protocol is supposed to provide to a

number of service users, our method automatically derives the specification of the corresponding communicating protocol entities. Both service and protocol entity specifications are modeled by the finite state machine (FSM) specification model. The interactions among the derived entities through a reliable FIFO communication medium combine to provide the set of specified services. Moreover, the synthesis method guarantees both semantic and syntactic correctness of the resulting protocol specifications.

Furthermore, to enhance the expressive power of the specification model and the functionality of the protocol, the FSM model is extended to allow the specification to include concurrent behaviors of service users at the distributed service access points. The synthesis method is modified accordingly to accomodate this extended FSM specification model of communication services and protocols. This provides a potentially useful link to state-oriented FDTs (Formal Description Techniques) such as SDL [CCITT 88] and Estelle [ISO 9074]. Moreover, the method is extended to synthesize error-recovery mechanisms in the derived protocol specifications to handle the case of an unreliable communication medium.

Finally, the feasibility and usefulness of the synthesis method is demonstrated by its application to ISO's association control service specification. Interestingly, the method exactly reproduces the corresponding ISO protocol, thus providing evidence of the value of our approach for real protocol design processes.

# TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1. Context and Motivation for the Thesis

Reliable computer communication protocols play a critical role in providing effective communication services. A *communication protocol* consists of a set of rules which govern the orderly exchange of information among network components in order to provide a specified set of services to service users located at different access points. The *communication service* specification describes the distributed functions that a communication system must provide to its service users. The specification of a communication protocol includes the specification of the communicating protocol entities, each servicing a particular service access point.

The increasing complexity of communication protocols requires more expressive and precise techniques to specify and develop reliable communication systems. Experience has shown that the development process for communication software is tedious, delicate and complicated. We hope that judicious formalization in the early phases of the process may avoid the occurrence of design problems in the final product. In this way, re-design costs can be minimized and communication software may be developed in the most cost-effective and productive way.

The use of natural language, English-like descriptions of increasingly complex protocols and services has unnecessarily complicated the work of designers and implementors. Such

descriptions are not rigorous and in fact, are often incomprehensible and ambiguous (allowing different interpretations).

In recent years, a great deal of research has been done in the area of formal description and validation techniques for distributed systems in general, and communication software in particular [FDT 88-89, IFIP 80-90]. In particular, a wide range of specification formalisms and techniques have been introduced. Each of these techniques attempts to offer desirable features for formal description techniques (FDTs), such as readability, conciseness, modularity and expressiveness. These formal techniques can be classified into seven categories: (i) the transition-based models which include the finite state machine (FSM) model [BOCH 78, DANT 80], and Petri-net models [BERT 82, BILL 82, DANT 77, DIAZ 82, MERL 76], (ii) the programming language-based models [BOCH 77, BRAN 78, BUHR 83, LOGR 83, SIDH 83, URAL 90], (iii) the formal language-based models [HAAS 85, HARA 78, TENG 78], (iv) the temporal logic-based models [HAIL 80, SCHW 82], (v) the behavior-based models which include the communicating sequential processes (CSP) [BROO 84, HOAR 85], and the calculus of communicating systems (CCS) [MILN 80, MILN 89], (vi) the trace-based models [BART 77, HOFF 85, MCLE 84], and (vii) the standard formal description techniques which include LOTOS [BOLO 87, ISO 8807], ESTELLE [BUDK 87, ISO 9074] and SDL [BELI 88, CCITT 88, SARA 87].

Among these formal specification models, the FSM model describes a protocol entity as a labelled directed graph. This model is very natural in its description of the sequence of behaviors of protocol entities. Therefore, it is relatively simple to model and analyze protocols using the FSM model. Moreover, the FSM model, as opposed to other models, has been extensively used for designing and specifying real-life protocols. Finally, FSM descriptions are familiar to probably the largest audience of communication software

engineers. For these reasons, we used the FSM model for the research reported in this thesis.

Because of their complex nature, communication protocols are very hard to design properly in an informal and non-methodical fashion [BOCH 80, RUDI 88]. As a result, design errors are not uncommon. Design errors will eventually result in an erroneous protocol implementation unless they are detected at an early stage. To eliminate these errors, we have to rely on the validation (design verification) phase: a post-design activity of the protocol development process which is intended to detect design errors. Moreover, the sequence of re-design, analysis, error detection and correction is applied iteratively until the protocol design becomes error-free.

These design errors are of two types:

1) general errors related to the *safety* of the protocol, often called *logical* or *syntactic* errors, such as deadlocks and unspecified receptions, and

2) specific errors related to the correctness of the functions or *liveness* of the protocol and are often called *semantic* errors.

Typically, syntactic errors can be detected by state-space search techniques, whereas semantic errors can be extremely difficult to detect. Various validation techniques have been developed to support the many formal specification models for communication software. Validation of FSM protocol specifications is based on a widely used, but inefficient technique called "*reachability analysis*" [BOCH 78, WEST 78]. To reduce the complexity of this technique, various other FSM-based validation techniques have been developed and are being widely used [LINC 88, YUAN 88].

A protocol design process consists essentially of the construction of interacting protocol entities which cooperate to provide a set of (possibly informally) specified services to the service users, while guaranteeing that no errors will be manifested. Two approaches to protocol design are known, *analysis and synthesis* [ZAFI 80].

The informal protocol design approach combined iteratively with a validation procedure, is often called the *analysis* approach to protocol design. The second approach is the *synthesis* approach, in which various methods are used to construct or complete a partially specified protocol such that the interactions among the constructed or completed protocol entities are free from both types of design errors. The synthesis is based on an incremental approach in which both the construction and the validation of the protocol are performed simultaneously. Therefore, the correctness of the synthesized protocol is often a direct by-product of the synthesis method. This thesis will focus on the synthetic design approach.

A related task is the verification of the protocol implementation against its protocol specification, called *conformance testing* (or *implementation verification*) [ISO 9646] [PROB 89]. We believe that specification and design validation tools should be designed keeping in mind the subsequent requirements for design and validation of conformance tests. This is an integral part of software design-for-testability, but is beyond the scope of this thesis.

In this thesis, we assume a layered architecture of communication software. As well, we choose to emphasize the importance of the service concept by making a clear distinction between protocols and services. At a high level of abstraction, we view a communication system as a service provider offering a specified set of communication services to some service users accessing the system through many distributed service access points (SAPs).

However, at a more refined level of abstraction, the communication services are provided to the service users by a number of interacting protocol entities which exchange protocol messages that are not observable at the SAPs. A protocol entity uses its lower layer service functions to relay protocol messages to another protocol entity. This viewpoint is made clear in Chapter 2.

The main motivation for the research done in this thesis is the urgent need for a synthetic protocol design methodology which would avoid the need for a time-consuming validation procedure since in principle, the validation of the protocol is a by-product of the protocol synthesis method itself. In the context of the layered architecture of the OSI reference model, the synthesis problem we need to solve is "the design of an (N)-protocol from both (N)- and (N-1)- service specifications" [VISS 85].

## 1.2. Objectives and Overview of Main Contributions

The primary objective of this thesis is to define a service-oriented method for the synthesis of protocol entity specifications starting from the service specification. The method must be flexible and efficient enough to be applied to the synthesis of multi-party protocols in both synchronous and asynchronous communication environments. Furthermore, the proof of correctness of the synthesized protocol must be directly derivable, so that no further validation of the resulting protocol would be required.

The main contributions of this thesis are:

(1) The introduction of an efficient localized and synthetic validation technique, which, in addition to validating a protocol, can be extended to synthesize or complete a protocol specification in order to satisfy the service specification.

(2) Direct evidence for the need for and direct benefits from using a precise service specification as a starting point for the automatic synthesis of reliable communication protocols.

(3) The introduction of a new service-oriented synthesis method for the design of communication protocols starting from the service specification. This method is more general than other existing synthesis methods in its application to synthesize multi-party protocols. Extensions to the method to synthesize error-recoverable protocols and to handle colliding service primitives are also proposed.

## 1.3. Outline of the Thesis

The thesis is organized as follows.

In Chapter 2, we present specification, design and validation issues, and a survey and assess synthetic protocol design methods.

In Chapter 3, we introduce a synthetic and localized protocol validation technique, together with its proof of correctness.

In Chapter 4, we elaborate on the concept of services in communication systems and the relationship between services and protocols. We also present the rationale for designing protocols starting from the service specification.

In Chapter 5, we present a new method for the automatic synthesis of communication protocols starting from the service specification, together with its proof of correctness. Both protocol and service specifications are modeled by the finite state machine specification model. Finally, we apply the method to an OSI service specification.

In Chapter 6, we propose some extensions to the specification model and the synthesis method to allow the handling of concurrent behaviors of service users, and the synthesis of error-recoverable protocols. Finally, we summarize the results of the thesis and conclude the thesis by presenting some directions for future research.

# CHAPTER 2

# SURVEY OF RELATED WORK

In this chapter, we review some issues related to protocol specification, validation and design activities of the protocol engineering process. In Section 2.1, we briefly introduce issues related to specification models for communication protocols. In Section 2.2, we define the problems that may exist in a protocol design. In Section 2.3, we present issues related to protocol validation. In Section 2.4, we describe two different approaches to protocol design, namely, analysis and synthesis. In Section 2.5, we survey and assess various synthesis methods for the design of protocols. Finally, in Section 2.6, we conclude the chapter.

## 2.1. Protocol Specification Models

In this section, we briefly review the layered architecture of ISO's open system interconnection (OSI) Reference Model and the related terminology that we refer to in this thesis. Then, we introduce some basic concepts, definitions and issues related to specification models for communication protocols and services.

### 2.1.1. Open System Interconnection Reference Model and Terminology

Because of the rapid growth in the computer communication and networking industry, the need arose for the standardization of rules for message exchanges (protocols) between machines and networks introduced by different manufacturers.

Within the International Standardization Organization (ISO), an architecture for open system interconnection (OSI), called the '*Open System Interconnection Reference Model* (OSI RM), has been developed [IEEE 83], [ISO 7498], [ZIMM 80]. The basic objective of the model is that an '*open system* ' from one manufacturer would be able to communicate via a network, with other '*open systems*' from different manufacturers.

To reduce the design complexity of computer communication protocols, networks are organized as a composition of seven layers. Figure 2.1 shows the seven layers as defined in the OSI reference model. The major principles used to arrive at this layered design are:

1) Each layer should perform a well-defined function.
2) A layer is created when a different level of abstraction is needed.
3) The layer boundaries should be chosen to minimize the information flow across the interfaces.
4) Each layer should offer certain specified services to the layer above it.

A layer N entity in one *open system* carries on a conversation with a layer N entity in another *open system*. The rules and conventions used in the conversation are collectively known as the layer N protocol. In reality, no data are directly transferred from layer N on one open system to layer N on another one (except at the physical layer). Instead, each layer passes data and control to the layer immediately below it, until the lowest layer is reached. At the lowest layer, there is a physical communication between open systems, as opposed to the virtual communication used by the higher layers. In summary, layer N uses its own functions and the services provided by layer N-1, to provide its own services to layer N+1 through its service access points. Figure 2.2 shows the interactions of layer N with layer N-1 below it and layer N+1 above it.

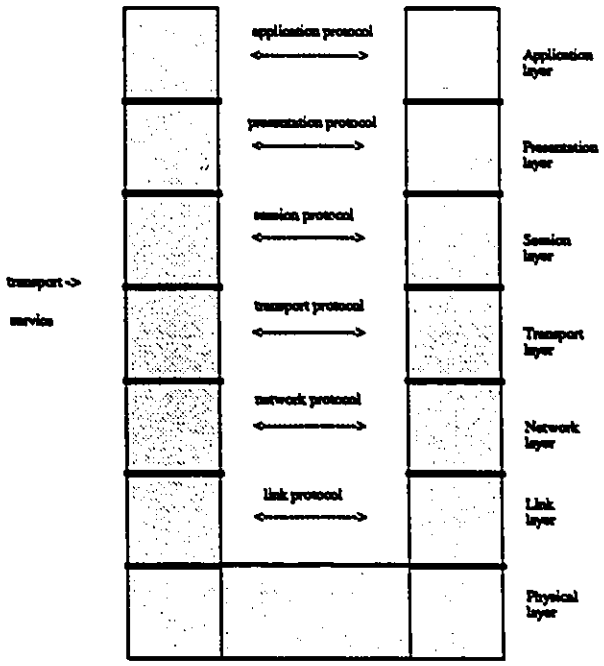



Figure 2.1. The Open System Interconnection (OSI) Reference Model

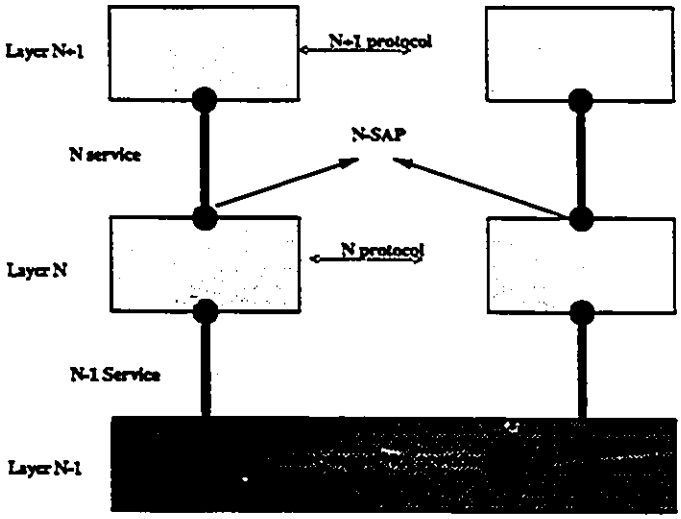



Figure 2.2. Interactions between layers of the OSI model.

From a user's point of view, a protocol provides a set of services allowing the interaction with other users at the same or equivalent layer. The input/output description of a protocol layer constitutes a *service specification* of the protocol layer. Normally, the user is concerned with *'what'* set of services a protocol layer provides, but not with *'how'* it provides it. Therefore, a protocol layer is viewed as a service provider whose service is fully described by sequences of inputs from and outputs to a user. The protocol designer is concerned with the internal behavior of a protocol layer.

### 2.1.2. Specification Models for Protocols and Services

Reliable computer communication protocols play a critical role in providing effective communication services. A communication protocol consists of a set of rules which govern the orderly exchange of information among network components in order to provide a specified set of services to service users located at different service access points.

The increasing complexity of protocols requires more powerful and precise techniques to specify them. These techniques are intended to facilitate the development of efficient and reliable communication systems. Formalization in the early phases of the process may avoid the occurrence of design problems in the final product. For example, for the specification of conventional (non real-time) software, an English text may be sufficient to describe its functions, but for real-time software (e.g., communication protocols), a more sophisticated and formal specification technique is required to facilitate the understanding of the system and to avoid misinterpretation by implementors.

Therefore, in recent years, a great deal of research has been done in the area of formal specification models for distributed systems in general, and communication software in particular [FDT 88-89] [IFIP 80-90].

#### 2.1.2.1. Desirable Features of Formal Specification Models for Protocols and Services

A protocol specification is a description of the requirements and functions that the protocol must provide.

Several desirable features should be considered when developing a formal specification technique for distributed systems in general, and protocols and services in particular. These features are:

(i) **Generality:** The technique should be general, that is: 1) be able to describe both control and data flow aspects of the protocol, and 2) be used not only for the specification, but also as a formal and well-defined basis for the design, validation and conformance testing of the protocol.

(ii) **Formality:** The technique should be based on a powerful semantic and mathematical model with a well-defined set of rules and operations. The existence of such a formal model would also ease the validation of the protocol design.

(iii) **Expressiveness:** The technique should be powerful enough to express (clearly and correctly) complex time-related features of protocols, such as concurrency and communication synchronization. Also, it should be able to model spontaneous actions, non-deterministic and unpredicatable behaviors that may occur in protocols.

(iv) **Readability:** Because a different interpretation of the same specification often leads to a service different than what is intended, the technique must provide a set of constructs and unambiguous syntax rules to enhance the readability and ease the understandability of the specification.

(v) **Modularity:** The specification must be able to describe a large and complex protocol without affecting its readability. This is achieved by providing some specification constructs that allow the modular expansion (or incremental expansion) of the protocol without modifying the structure of the existing description.

(vi) **Adequate Degree of Abstraction:** The technique should be abstract and implementation independent. This implies that it should describe all the requirements and functions the protocol provides, without describing how it provides them. This criterion guarantees that the technique is not concerned about implementation details, which would greatly affect the readability of the specification.

(vii) **Executability:** The technique should be capable of generating a running prototype, which consequently allows for the validation to start at an early stage of the protocol development process.This feature will also increase the confidence of the designer.

(viii) **Testability:** The technique should be able to assist the designer at the implementation verification (or testing) stage. This feature includes aspects of i), ii), iii), iv), v) and vii) above.

A wide range of specification formalisms and techniques have been introduced in the literature. Each of these techniques satisfies some combinations of the features listed above. These formalisms and techniques can be classified under seven models [SALE 88]: (i) the transition-based models which include the finite state machine (FSM) model and Petri-net models, (ii) the programming language-based models, (iii) the formal language-based models, (iv) the temporal logic-based models, (v) the behavior-based models which include the communicating sequential processes (CSP) and the calculus of communicating systems (CCS), (vi) the trace-based models, and (vii) the standard formal description techniques.

Presently, three standard formal description techniques (FDTs) are being used by researchers and practitioners in the communication industry [ISO 3252]. SDL (Specification and Description Language) [BELI 88, CCITT 88, SARA 87] has been introduced by CCITT, LOTOS (Language Of Temporal Ordering Specification) [BOLO 87, ISO 8807] and ESTELLE [BUDK 87, ISO 9074] have been introduced by the International Organization for Standardization (ISO) and are now International Standards. These FDTs are being applied to the specification of telephony software, and to the specification of OSI protocols and services [FDT 88-89, IFIP 80-90].

FDTs are characterized by strict rules which allow the unambiguous description of distributed systems. The main reasons for using FDTs are that:

a) FDTs enable the specifier to write unambiguous, clear and concise specifications
b) FDTs allow the analysis of specifications for correctness with respect to user requirements so that errors can be detected earlier in the design
c) FDTs ease the derivation of designs for systems providing the functionality expressed in the specifications
d) FDTs facilitate the automated generation of partial implementations (fast prototyping)
e) FDTs allow the application of formal test generation and selection methods

In the following section, we introduce the FSM specification model for communication protocols and services since this is the most universally familiar model and the one we will use in this thesis.

### 2.1.2.2. FSM Model for Protocols and Services

Communication protocols can be modeled by communicating FSMs each representing a protocol entity, and channels linking those entities. Channels can also be represented by FSMs. This model assumes underlying first-in-first-out (FIFO), reliable communication channels.

A formal definition of the FSM model of protocols is given below.

**Definition 2.1.** A protocol ($\mathcal{P}$) is defined as a finite set of FSMs { *fsm*1, *fsm*2, *fsm*3, ..., *fsmn*} where **n** is the number of entities involved. In $\mathcal{P}$, each FSM *fsmi* is defined by a quadruple ($S_i$, $\Sigma_i$, $T_i$, $\sigma_i$), where:

$S_i$ is a non-empty finite set of states of protocol entity **i**.

$\Sigma_i$ is a non-empty set of protocol messages sent or received by protocol entity **i**.

$T_i$ is a partial transition function between protocol states (it is a subset of the cartesian product $S_i \times \Sigma_i \times S_i$).

$\sigma_i \in S_i$ is the initial state of protocol entity **i**.

Also, note that each transition is associated with only one message (received or sent). $T_i$ is a subset of the cartesian product $S_i \times (\Sigma_i, \Delta) \times S_i$, where $\Delta$ is only used when a message transmission is involved and it represents the destination set of protocol entities to which the message is transmitted. For simplicity, $\Delta$ can be omitted when only two protocol entities are involved.

In this model, each protocol entity is represented by a directed labeled graph whose nodes and edges (see Example 2.1 in Figure 2.3) represent states and transitions, respectively.

Each transition is either a message transmission, or a message reception, and each pair of entities is linked by a reliable full-duplex FIFO channel. Furthermore, no assumption is made about the time a process spends in a state before sending a message, and about the time a message spends in a channel before being delivered to its destination. In other words, the model lacks an inherent mechanism for representing of complex control flow aspects of protocols such as timing, synchronization and concurrent behaviors aspects. FSMs are most suitable for modelling sequential communication behaviors expressed as a set of linear sequences of transitions.

**Definition 2.2.** A *(initial) global protocol* state is defined as a feasible combination of the (initial) states of the interacting protocol entities and the states of the channels linking them.

**Definition 2.3.** A *stable global protocol* state is a global state reachable from the initial global state with all channels empty.

**Definition 2.4.** An *enabled* transition in a protocol entity is either a transmission transition or a reception transition for a message which is the first message in the incoming FIFO slot for that entity.

**Definition 2.5.** A channel is said to be *bounded* if, for every reachable global state, the number of messages in this channel is less than or equal to a pre-specified constant. Furthermore, a protocol is said to be bounded if all of its channels are bounded.

Figure 2.3 shows a two-party protocol example consisting of protocol entities PE1 and PE2. The transmission of a message m is represented by "!m", and the reception of a message m is represented by "?m". In this example, $\mathcal{P}$ = {PE1, PE2}, where:

PE1 = {{s1, s2, s3, s4}, {a,b,c,d}, {(s1,!b,s2), (s2, ?a, s3), (s3, ?a, s4), (s4, !d, s3), (s3, ?c, s1), (s1, ?a, s3) }, s1 }, and

PE2 = {{s1, s2, s3, s4}, {a,b,c,d}, {(s1, !a, s2), (s1, ?b, s2), (s2, ?d, s4), (s4, ?a, s1), (s2, ?b, s3), (s3, ?b, s2), (s2, !c, s3)}, s1 }.

The initial global state of the protocol is (s1, s1) and a possible stable global state is (s3, s3). An alternative simple representation for these protocol entities is via state transition tables.

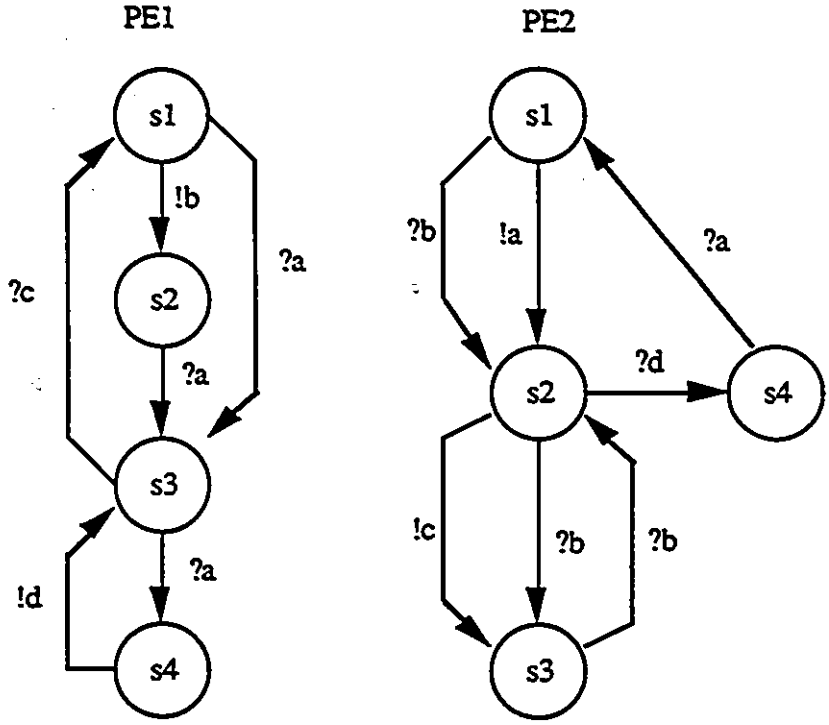



Figure 2.3. FSM protocol specification for Example 2.1.

## 2.2. Potential Protocol Design Problems

A protocol design process consists essentially of the construction of interacting protocol entities which cooperate to provide a set of specified services to the service users, while guaranteeing that no errors will be manifested.

Because of the complex nature of protocol specifications, design errors often occur. These errors will be propagated to the protocol implementation phase unless they are detected at early stages using an appropriate protocol validation technique.

Two types of properties must be guaranteed in a protocol design [HANS 89] [SAJK 84]:

(1) the *safety properties* of the protocol which ensure that the protocol never enters an undesirable global state, i.e. something bad never happens (the protocol is safe). These general properties include freeness from deadlocks, freeness from livelocks, and completeness (i.e. absence of unspecified reception errors).

(2) the *liveness properties* of the protocol which ensure that the protocol will eventually enter a desirable state, i.e. something good will eventually happen (the protocol is live), meaning that the protocol will perform its intended functions with respect to the service specification.

These properties can be guaranteed by verifying the absence of *syntactic* and *semantic* design errors. These two types of errors are described below:

### 2.2.1. Syntactic or Logical Design Errors

The syntactic correctness of a protocol design is concerned with the logical structure of the exchange of messages among the protocol entities. Syntactic design errors include deadlocks, unspecified receptions, instabilities, livelocks, overspecifications, and channel overflows (defined below). The abscence of such design errors guarantees the safety of the protocol. In general, this type of error can be defined generically and independently of the service provided. These design errors are defined below:

*i) Deadlock*

A protocol is in a *deadlock* state if it enters a global state (except for the desirable final sate) in which no message transmissions are possible and all the channels are empty, i.e. no further progress is possible.

*ii) Incompleteness*

A protocol is said to be *incomplete*, if at a certain state, one entity is offered a message the reaction to which is unspecified. This error is also called *unspecified reception*.

*iii) Instability or state ambiguity*

A protocol is said to be *unstable* if an initial state of one protocol entity may coexist with different states of other cooperating entity (with empty channels). The result of such an error is the potential loss of synchronization between the cooperating entities.

*iv) Livelock or dynamic deadlock*

A protocol is in a *livelock* state if it enters a state at which infinite non-productive cycles are possible.

*v) Non-executable transitions or overspecification*

A protocol is said to be *overspecified* if it contains *non-executable* transitions, that is transitions that can never be traversed. This error is analogous to dead or unreachable code in computer software.

*vi) Unboundedness or Overflow*

A protocol design is *unbounded* if a protocol state can be reached such that the channel linking any pair of communicating protocol entities contains a number of messages exceeding the predefined capacity for that channel.

*vii) Non-termination or Acyclicity*

A protocol design has a termination problem (respectively is acyclic) if after a number of interactions, it will never reach the final (respectively initial) state when started from the initial state.

*viii) Unfairness*

A protocol design is *unfair* if a state in the protocol can never be reached even if the transitions leading to it are feasible and executable according to the specifications. In general, non-terminating protocols with at least one final state are unfair protocols.

For the protocol example shown in Figure 2.3, the global protocol state (s3, s3) is a deadlock state of the protocol. This state is reachable from the initial global state (s1, s1) by executing !b?a in PE1 and !a?b in PE2. Furthermore, at the global protocol state (s2, s3), reachable from the initial global state by executing !b in PE1 and ?b!c in PE2, an unspecified reception error will occur. This is because a reception transition ?c is missing at state s2 in PE1.

### 2.2.2. Semantic Design Errors

The semantic correctness of a protocol is concerned with the correctness of the functions to be provided by the protocol. A semantic design error is mainly related to the abnormal functioning of the protocol and its inability to meet its intended purpose (i.e. providing specified services). Moreover, this type of design error cannot be classified generically because it depends on the specific service specification.

Verifying the total correctness of the protocol is based on proving both liveness and safety properties of the protocol design, that is freeness from both syntactic and semantic design errors.

## 2.3. Protocol Validation Techniques

Because of their refined nature, protocol specifications are much more complex than service specifications. Often there are many ways to design a protocol entity to meet its formal specification. Therefore, the protocol design obtained may contain design errors that will prevent the protocol from providing its specified services.

*Protocol validation* (or *design verification*) consists of guaranteeing that the interactions among the protocol entities will provide the specified services. This involves the verification of both the syntactic and semantic correctness of the designed protocol. The verification of the latter involves the service specification. Furthermore, protocol verification depends on the properties of the lower layer service provider. Therefore, to verify that a layer N protocol meets its service specification, it will be necessary to assume the correctness of the layer N-1 services.

A related task is the verification of the protocol implementation against its protocol specification, called *conformance testing* (or *implementation verification*). We believe that specification and design validation tools should be designed keeping in mind the subsequent requirements for design and validation of conformance tests. Others have also recommended bringing protocol validation techniques and conformance testing techniques closer together [MILL 90].

Validation techniques allow designers to detect errors in protocol designs at an early stage in the protocol development process, provided a formal definition of the protocol is available. The early detection of errors would consequently reduce development cost and increase the reliability of distributed communication systems.

When developing a validation technique, several criteria must be considered:

a) the technique must be efficient and not resource-expensive (with respect to memory and CPU time).
b) the technique should be easily automated and easily used by designers.
c) the technique must be powerful enough to detect different kinds of design errors, both semantic and syntactic errors.
d) the technique must be general enough to be applied to multi-party protocols as well as to two-party protocols.

A large number of validation techniques have been developed to support the many formal specification models for communication software. Protocol validation techniques have also been surveyed in [SAJK 84] [YUAN 88].

In the following, we briefly discuss validation techniques supporting the FSM specification model.

### 2.3.1. Validation of FSM Specifications of Protocols

Since FSMs consist of nodes (states) and edges (transitions), their manipulation is easily automated. Moreover, the representation of a protocol entity as a graph allows the application of algorithms and methods developed in graph theory and automata theory [BARR 79, EILE 74, HOPC 69].

Using the FSM model, a common validation technique, called *global state exploration* or *reachability analysis*, was proposed to provide mechanical and automated protocol validation [BOCH 78, WEST 78, ZAFI 78]. The state exploration method starts with the initial global protocol state and then constructs a *reachability graph*. First, all global states reachable from the initial global state and appropriate transitions are added to the graph. Then, for every newly added reachable global state, the same process is applied (addition of new states and transitions). The procedure stops at a global state (a leaf node in the reachability graph) when a deadlock is detected at that state or if the same state was previously reached. This validation method identifies all deadlock states and livelocks, verifies some properties such as stability, and detects unspecified receptions.

Although the reachability analysis technique is widely used, it has many serious problems. One of the problems is the undecidability of termination of the reachability process [WEST 78]. Moreover, for complex multi-party protocols. This type of validation also faces the problem of combinatorial explosion in the size of the state space. To reduce the complexity and seriousness of this size problem, various techniques have been proposed. These validation techniques are classified into five categories [YUAN 88]: closed covers [GOUD

84c], reachability analysis [WEST 78, ZAFI 78], divide-and-conquer [CHOI 86b, CHOW 85, LAMS 84, VUON 82], partial state exploration [LINC 87, MAXE 87, WEST 86,89], and localized [BRAN 83, KAKU 86, SALE 90a, YUAN 89].

## 2.4. Protocol Design Approaches

A lot of research has been done into the development of formal methods for the design of communication protocols. These methods tend to follow one of two types of design approaches, namely, *analysis* or *synthesis* [ZAFI 80]. In this section, we first present the analysis approaches [BLUM 86, BOCH 78, WEST 78, ZAFI 78] and the synthesis approaches [BOCH 86, CHOI 86a, CHUL 88a, GOUD 84b, KAKU 87, MERL 83, RAMA 85, SIDH 82, ZAFI 80, ZHAN 88]. Then, we identify several features that we use to classify the synthesis design methods approaches.

### 2.4.1. Analysis Approach to Protocol Design

In the ***analysis*** approach, the protocol designer starts with a preliminary version of the protocol in which the syntactic and semantic validation aspects are often overlooked. The preliminary version is usually obtained by defining messages (Protocol Data Units) and the effect of message exchanges on the protocol entities under design. This approach often results in an incomplete and erroneous design. Therefore, a design verification and analysis process is then performed in a separate post-design activity and is based on analysis techniques intended tc detect errors and omissions in the protocol design. The sequence of re-design, analysis, error detection and correction is applied iteratively until the protocol design becomes error-free. This approach is time-consuming because of its iterative, trial-and-error nature. Another shortcoming is that the reformulation or modification of an existing design is often a very difficult and time-consuming task. We believe that the lack

of emphasis on a well-defined service specification is at the core of difficulties with this approach. Design validation and analysis techniques have been surveyed in [SAJK 84] and [YUAN 88]. We only address synthesis approaches in this thesis.

### 2.4.2. Synthesis Approach to Protocol Design

In the ***synthesis*** approach, a partially specified protocol design is constructed or completed such that the interactions between its protocol entities proceed without manifesting any logical error and (ideally) provide the set of specified services. The synthesis is either performed interactively in an incremental fashion or automatically. In both cases, the construction and the validation of the protocol are performed simultaneously. The correctness of the synthesized protocol is a direct by-product of the synthesis process itself. Therefore, no further verification of the protocol design is needed as was the case for the analysis approach.

#### 2.4.2.1. Classification of Protocol Synthesis Methods

In order to classify and compare the protocol synthesis methods published in the literature, we identify and describe some general features that can be used to characterize those methods. These features are: the starting point of the method, the modeling formalism used to describe the protocol, the constraints on the communication model, the mode of interaction with the designer, the protocol properties guaranteed by the method, the protocol functions provided in the design, and the complexity of the method.

- the ***starting point*** of the method: The synthesis process can start: (a) from scratch with only informal specifications of requirements, (b) from a complete specification of one protocol entity, (c) from partial specifications of the protocol entities

involved, or (d) from a complete service specification. Although other combinations are possible, all methods, except one [MERL 83], surveyed fell into one of these categories.

- the ***modeling formalism*** used to describe the protocol: Various formalisms can be used to model the synthesized protocol. This includes models like finite state machines, Petri-nets, LOTOS-like and trace models.

- the ***constraints on the communication model***: One of the constraints on the communication model is related to the interactions among the communicating entities and the service users: either rendezvous type interactions, in which the exchange of messages is tightly coupled, or a loosely coupled interactions to model the asynchronous behavior of the service users. Another constraint is related to the number of protocol entities that can be synthesized by the method. Other constraints include the features of the communication medium such as the ordering of messages in the medium (e.g. first-in-first-out) and the degree of reliability of the medium.

- the ***mode of interaction*** with the designer indicates whether the protocol design is generated *automatically* or constructed *interactively* by allowing the designer to intervene and make design choices whenever requested to do so.

- the ***protocol properties guaranteed*** by the method: This includes the liveness and safety properties of the protocol.

- ***specific protocol functions*** included in the design: In addition to the synchronization function (protocol message exchanges and service primitive interactions), a protocol design must provide other (protocol-specific) functions

such as: (a) error recovery when the underlying communication medium is assumed to be unreliable, (b) throughput (performance) enhancement functions, i.e., speeding up the protocol, and (c) testability, i.e. making the protocol implementation easily testable.

- ***complexity*** of the method: Some methods generate global system states (cross product of protocol entity states and channel contents) during the synthesis process, and other methods consider each of the entities separately. Also, some methods generate more protocol interactions than required (over-specification).

Protocol synthesis methods can be divided into two broad categories based on the starting point or reference point for the design:

*Category 1:*

Methods starting from a given service definition [BOCH 86, CHUL 88a]. A mixed method, described in [MERL 83], starts from a given service definition and the specification of all but one of the protocol entities involved, and synthesizes the missing entity.

*Category 2:*

Methods that do not consider the service as the authoratative reference point [CHOI 86a, GOUD 84b, KAKU 87, RAMA 85, SIDH 82, ZAFI 80, ZHAN 88]. These methods start either from a partial protocol description or from a complete description of one protocol entity, and either automatically or interactively synthesize the complete protocol.

In the second category, further design verification should be applied in order to ensure that the synthesized protocol provides the specified service, that is, the semantic or liveness properties must be satisfied. Furthermore, these methods are not suitable for the design of protocols in a layered architecture, such as the OSI reference model [ISO 7498], since no reference is made to the provided service. Therefore, those methods in the second category are not purely synthesis methods, in the sense that freedom from syntactic and semantic errors is not guaranteed by the construction itself.

Figure 2.4 shows a broad classification of ten existing synthesis methods based on their reference to the service and the modeling formalism used.

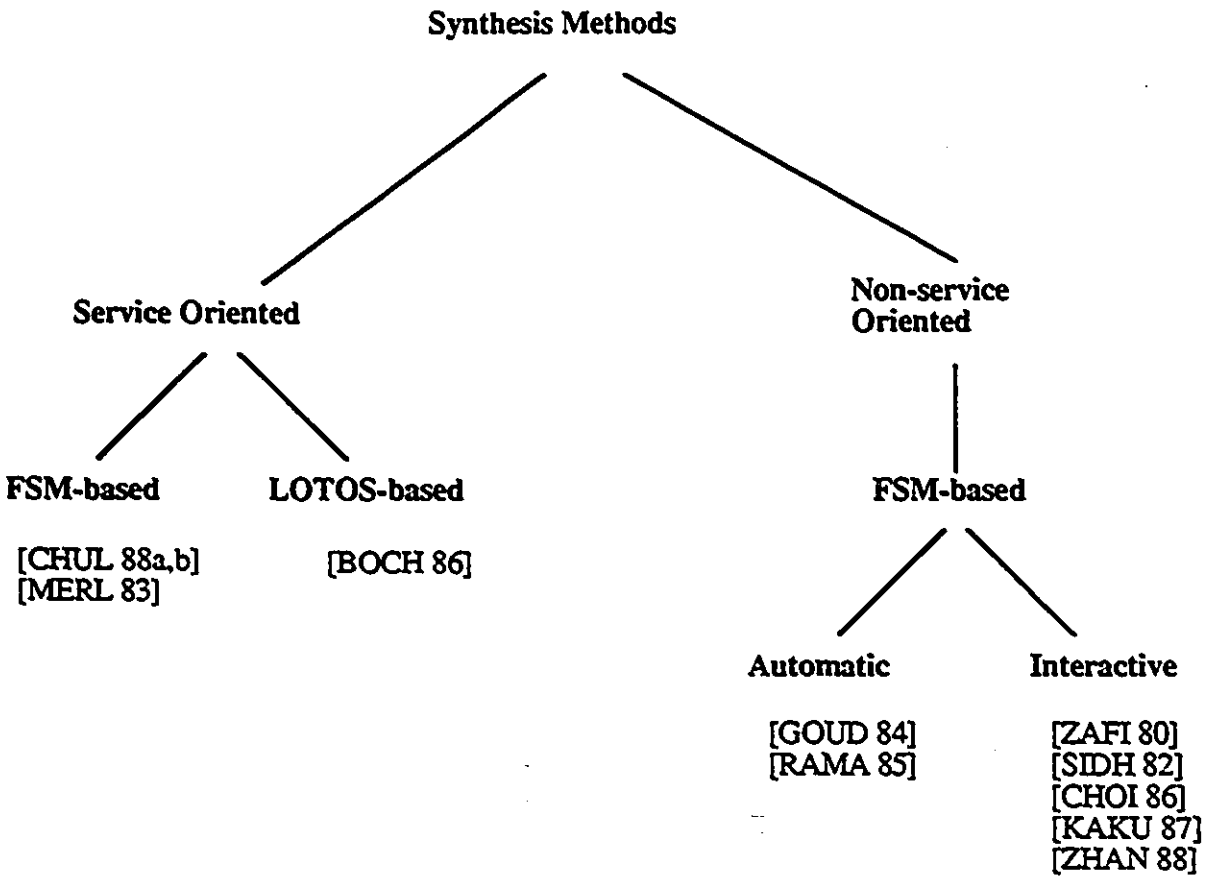



**Figure 2.4.** Classification of synthesis methods according to orientation and formalism.

## 2.5. Protocol Synthesis Methods

In this section, we briefly survey and assess ten synthesis methods. First, we evaluate three service-oriented synthesis methods. Then, we survey seven non-service-oriented synthesis methods. The assessment of these methods will be mainly based on the classification features described in Section 2.4.

### 2.5.1. Service-Oriented Synthesis Methods

Two methods [BOCH 86, CHUL 88a] refer to the service specification as the starting point in the protocol synthesis process. Another method [MERL 83] starts with the service specification and the specification of all but one of the protocol entities. Although it is not purely service-oriented, it is the first published synthesis method to refer to the service specification. These methods are briefly described below.

*1) Submodule construction method:*

Merlin et al. [MERL 83] proposed a method for constructing a finite state machine representation of a protocol entity by solving the following problem: "Given n-1 communicating finite state machines ($n \geq 2$) (one of which may be the underlying medium), and a service specification, it is required to synthesize an nth communicating machine such that the service performed by the n communicating machines is defined by the given service specification". A special assumption is made on the model: the communication between the entities is tightly coupled, which implies that the interaction consisting of the sending and subsequent receiving of a message is an atomic operation and the delay between the sending and receiving of an event is ignored. This method is the first one to refer to the

service specification; however it is not purely service-oriented, since the knowledge of at least one protocol entity is required to derive the peer entity in a two-party communication service provider.

The major drawbacks of this method are as follows. The method is based on a formula which produces an overspecification of the missing nth machine by including a large number of infeasible execution sequences. Therefore, the resulting communicating machine contains redundant or non-executable transitions and may reach a deadlock when communicating with the (n-1) machines. To solve this problem, the method must be complemented by a time-consuming reachability analysis-based validation procedure to remove useless and error-prone transitions. Furthermore, the method is too restrictive in assuming a close synchronization between the communicating entities. Message collisions, for example, are not allowed; Thus, the limited expressive power of the method and its limited scope of application exclude a wide range of asynchronous and asymmetric real protocols.

### *2) Derivation of protocol specifications from service specification in LOTOS:*

This method was introduced in [BOCH 86]. The communication model consists of several protocol entities which interact through their respective service access points (SAPs) in order to provide the set of specified services. The communication medium is assumed to be a full duplex reliable FIFO channel, and the message interactions between the different components of the model are based on the rendezvous concept. The service specification is described by a subset of LOTOS [ISO 8807] operators which includes ';' for sequencing, '|||' for independent parallelism, and '[]' for choice. The synthesized protocol is also described in the same specification language.

The derivation algorithm considers a syntactic tree representation of the service specification, and based on some production rules, synthesizes the protocol entities automatically. The derived protocol specifications are claimed to be free from syntactic errors, such as deadlocks, and to satisfy the service requirements. However, the original derivation algorithm was found to involve an unnecessarily large number of synchronization messages. This inefficiency was later addressed and the derivation algorithm was optimized in [KHEN 89]. In addition, the algorithm was modified to include recursive service specifications, hence extending the specification language to allow recursive definitions. The method was also extended to include the synthesis of parameterized specifications [GOTZ 90].

The major weaknesses of this method are: (1) the model is restricted to synchronous communication, which unfortunately oversimplifies the synthesis problem and therefore excludes a large class of asynchronous protocols and services, (2) the service specification language can describe concurrent activities at the SAPs, however the method cannot handle such activities, and (3) the method does not allow for distributed choices, that is, a choice involving actions occurring at different SAPs, and therefore cannot even be applied to very simple real-life service examples.

Recently, a new synthesis method has been developed which starts from a basic LOTOS service specification and synthesizes the protocol specifications [LANG 90]. Although this method is limited to two-party services, it does eliminate the restriction on distributed choices and provides a proof of correctness for the synthesized protocol.

***3) Synthesis of protocol specifications from service specification in the FSM model:***

This method, introduced in [CHUL 88a], starts from a given service specification modeled by one or more finite state machines which describe the interleaving of the interactions at two SAPs. The two synthesized protocol machines communicate over reliable communication channels to provide the specified services. The constructed protocol is claimed to be free from deadlocks, unspecified receptions, non-executable transitions and overflows. Furthermore, the protocol entities are claimed to provide the specified services. The method is also complemented by an algorithm to construct an error detecting and error-recoverable protocol from the generated protocol. In [CHUL 88b], the service specification language has been extended to allow more flexible specification of concurrent activities at both SAPs.

The method starts by transforming each state in the FSM service specification into four different states. Transitions are then added to link those states. After thus transforming all service states, a more complex FSM is obtained, which is then decomposed into two protocol FSMs.

The major limitations of this method are: (1) the synthesized protocol entities are always closely synchronized, therefore, the application of this method excludes a wide range of asynchronous protocol entities and services, and (2) a generalization to the synthesis of multi-party protocols is not straightforward. Furthermore, after a careful examination of the synthesis algorithm we found that the synthesized protocol may contain a livelock.

**Overall Comparison of Service-Oriented Methods:**

After careful study of the three service-oriented synthesis methods, we can draw the following comparisons:

- Although the methods of Bochmann [BOCH 86] and Chu [CHUL 88a,b] claim to synthesize correct protocol specifications, no proofs for the semantic and syntactic correctness of the resulting protocols are given. As stated in [BOCH 86], a proof of correctness for the method is one of the outstanding issues.

- The method of Chu [CHUL 88a,b] mainly synthesizes the control flow part of the protocol. Bochmann's [BOCH 86] method was recently extended to deal with parameterized specifications, but it is still far from generating the data part of the protocol.

- The method of Chu [CHUL 88a,b] synthesizes error-recoverable protocol specifications. Due to the possible complexity of the service specification, the extension of Bochmann's method to add error-recovery mechanisms is inherently complex.

- None of the methods considers the case of asynchronous and concurrent service users' behaviors at the distributed SAPs.

Table 2.1 summarizes the main attributes of the three synthesis methods described in this section.

| | Submodule Construction [MERL 83] | Derivation of protocol specs from service specs [BOCH 86, KHEN 89] | Synthesis of protocol specs in the FSM model [CHUL 88a,b] |
|---|---|---|---|
| **Starting Point** | service specification and n-1 communicating finite state machines | Service specification | Service specification |
| **Constraints on Communication Medium** | Synchronous | Synchronous<br>Reliable medium<br>N protocol entities | Synchronous<br>Reliable or unreliable medium<br>2 protocol entities |
| **Modeling Formalism** | FSM model | Lotos-like | FSM model |
| **Mode of Interaction** | Automatic<br>Few fixed comm. patterns | Automatic<br>Few fixed comm. patterns | Automatic<br>Few fixed comm. patterns |
| **Protocol Properties** | Liveness properties satisfied, but not safety properties (no proofs) | Liveness and safety properties satisfied (no proofs) | Liveness and safety properties satisfied (no proofs) |
| **Specific Functions** | None | None | Error-recovery |
| **Complexity** | Overspecification | Localized | Localized |

Table 2.1. Summary of the main features of service-oriented synthesis methods.

### 2.5.2. Non-Service-Oriented Synthesis Methods

In this section, we describe seven synthesis methods which start from partial protocol specifications in order to synthesize the protocol. These partial specifications can be either a partial specification of each of the protocol entities or a complete specification of one of the protocol entities, and the synthesis of the protocol can be either interactive or automatic.

*1) Symmetric Synthesis and Analysis:*

Zafiropulo et al. [ZAFI 80] proposed some production rules for the interactive synthesis of protocol entities modeled by finite state machines which exchange messages over reliable FIFO channels. These rules are based on a cause and effect relationship in message interactions between two communicating machines. The synthesis is performed interactively using a tool which executes the following design steps: 1) the designer adds one sending transition to one of the two incomplete FSMs, 2) a tracking algorithm uses the production rules to add the corresponding receiving transitions in the other machine such that unspecified reception errors will not occur, and 3) the designer checks whether or not the added transitions can lead to a deadlock or channel overflow. The design process terminates when the designer chooses not to add any further sending transitions to either of the two machines. The application of the production rules guarantees that the synthesized protocol will be deadlock-free, complete, free from non-executable transitions and from state ambiguities. In [BRAN 80], the method was extended to cover communication between more than two FSMs, and new production rules were introduced for this purpose.

This method has the following drawbacks: (1) the design steps are time consuming because they are based on a trial and error procedure in which new transitions are first added, then verified, (2) although the production rules refer to each local entity separately, the detection of deadlocks and overflows in step 3) is based on a costly exhaustive global reachability analysis technique, and (3) the termination of the method is not guaranteed to produce an error-free design.

***2) Protocol design rules:***

Sidhu [SIDH 82] proposed some protocol design rules for the interactive and incremental construction of communication protocols. During protocol synthesis, the designer deals with three components: an informal specification of the protocol, a global reachability tree and the construction of trees representing the protocol entities. The basic idea is to generate a new global system state by perturbation from the global states already reached by previous perturbations. The perturbation of global states and the synthesis procedure continues as long as the informal specification has some messages that the designer has not yet considered or some communication channels have messages to be received by some entity. At the end, we have a global reachability tree, whose nodes are NxN matrices representing the states of the system, and N entity trees, one for each of the N protocol entities built during the protocol synthesis. The author suggests some extensions on the method to permit semantic validation of the constructed protocol using a version of temporal logic called computational tree logic (CTL).

This method has the following drawbacks: (1) the global reachability graph becomes unmanageable for complex protocols, (2) the termination of the synthesis process is not formalized, and (3) the design rules are only suitable for tightly coupled synchronously communicating machines.

*3) Synthesis of communicating finite state machines:*

Gouda et al. [GOUD 84b] introduced a method for the synthesis of two communicating finite state machines which exchange messages over two unidirectional FIFO channels. Given an incomplete communicating FSM specification M, the method constructs two FSMs, M' and N' that are guaranteed to satisfy the general syntactic properties of two communicating FSMs.

The method consists of two algorithms. The first, called the 'machine synthesis algorithm', takes one machine M and constructs two communicating machines M' and N'. M' is constructed from M by adding receiving edges to it and the algorithm guarantees that the communication between M' and N' is free from syntactic errors, such as deadlocks, unspecified receptions, state ambiguities and nonexecutable transitions. The second algorithm, called the 'channel capacity algorithm', takes as an input M' and N', and computes the smallest channel capacity between them.

The communication between the two constructed communicating machines proceeds until a loss of synchronization caused by a message collision occurs at mixed states (both receiving and sending states). Upon detection of the loss of synchronization, one machine (the loser) stops its current progress and rejoins the second machine, while the second machine (the winner) discards all messages sent by the first machine during the loss of synchronization. Then a harmonious communication between the two machines is resumed.

This construction method has several drawbacks, of which: (1) the concept of a 'loser' and a 'winner' machine offers only one fixed communication pattern between the two FSMs,

hence an asymmetry exists in the model which may affect its suitability for some real life applications in which no messages should be discarded, and (2) the generalization to more than two communicating FSMs is not straightforward.

*4) An automated protocol synthesizer:*

Ramamoorthy et al. [RAMA 85] developed a synthesis method which automatically generates a peer entity starting from a given local entity modeled by a Petri-net. The protocol entities exchange messages over two reliable unidirectional channels.

An algorithm is proposed to automatically translate the Petri-net representation into a finite state machine and to check the correctness and completeness of the local machine. Another algorithm uses six transformation rules to automatically generate the FSM representation of the peer entity by producing some fixed communication patterns based on the patterns existing in the local entity. Finally, the FSM representation of the peer entity is translated into a Petri-net. The method has been extended in [RAMA 86] to include error recovery mechanisms when unreliable communication channels are assumed.

The method has several drawbacks, of which: (1) a complete and correct Petri-net representation of the local entity must be guaranteed by the designer; therefore the correctness of the resulting protocol depends on the correctness of the local entity, (2) the rules produce fixed communication patterns, and (3) the generalization of the method to consider more than two communicating entities is straightforward.

*5) Sequence method for protocol construction:*

Choi [CHOI 86a] proposed an interactive method for constructing communicating finite state machines using protocol sequences. The method consists of invoking two algorithms for the generation of well-formed protocol sequences for cyclic or acyclic protocols. Subsequently, a third algorithm is used to transform the protocol sequences of each protocol entity into a finite state machine. The method is limited to specific protocol structures in which the level of nesting of iterative subsequences is limited to zero or one.

The method has the following drawbacks: (1) it considers only some fixed communication patterns thereby limiting the applicability of the method, and (2) the generalization to more than two communicating FSMs is not straightforward.

***6) Component-based synthesis of protocols:***

Kakuda et al. [KAKU 87] proposed a component-based algorithm for protocol synthesis. A component is a protocol specification primitive consisting of a pair of fundamental process behavior diagrams. Protocol specifications are synthesized by combining some of the components provided by the algorithm as fundamental parts of such specifications. The synthesized protocol specifications are free from unspecified receptions, non-executable code, deadlocks and channel overflow. The synthesis starts from a complete process interaction diagram (a higher level of modules or process message exchanges) and a partial process behavior diagram entered interactively by the designer in order to enhance the flexibility of designing protocols by allowing the existence of several communication patterns. 22 components are identified and they consist of: a) the six transformation rules identified by Ramamoorthy [RAMA 85], b) ten patterns for looping receptions, and c) six patterns for the generalization to n ($n > 2$) communicating processes.The correct combination of these components guarantees the progress of the synthesized protocol.

Although the paper refers to the 'process interaction diagram' as an input to the synthesis algorithm, it is not clear how this diagram is used. Elaboration on the concept of 'process interaction' as a higher level of abstraction hiding the process behaviors would lead to the service concept and requirements. This diagram is similar to the service specification machine introduced in [CHUL 88a,b].

This method has the following drawbacks: 1) it lacks a formal proof of correctness for the combination of components, and 2) it is not optimal since it generates unnecessary sending transitions. Subsequently an optimization method can be applied to delete such transitions.

*7) Interactive protocol synthesis using a global state transition graph:*

Zhang et al. [ZHAN 88] proposed an interactive synthesis method to construct two communicating finite state machines which exchange messages over two unidirectional FIFO channels. The method consists of an algorithm which, depending on the user's input, first constructs a global state transition graph (GSTG) of the protocol to be synthesized and then produces the protocol by decomposing the GSTG into two machines.

The algorithm is based on a set of production rules, a set of deadlock avoidance rules and a termination mechanism which together guarantee the avoidance of deadlocks, unspecified receptions and channel overflows in the synthesized protocol. In [ZHAN 89], the method was integrated into a knowledge-base system for the design of protocols.

This method has the following drawbacks: (1) the GSTG becomes very complex and unmanageable for the synthesis of large and real-life protocols, (2) the synthesis algorithm cannot be easily generalized to synthesize more than two communicating entities, and (3)

the synthesis process would become much more space and time-consuming for multiple entities.

Table 2.2 summarizes the attributes of the non-service-oriented synthesis methods described in this section. Note that all these techniques, by their nature, require the application of subsequent verification methods to verify that the intended service functions will be provided.

| | Analyzing and synthesizing protocols [ZAFI 80] | Protocol design rules [SIDH 82] | Synthesis of CFSMs [GOUD 84] | Automated protocol synthesizer [RAMA 85] |
|---|---|---|---|---|
| **Starting Point** | N incomplete protocol entities | N incomplete protocol entities | Two incomplete protocol entities | One correct and complete protocol entity |
| **Constraints on Communication Model** | Asynchronous<br>Reliable medium | Synchronous<br>Reliable medium | Asynchronous<br>Reliable medium | Asynchronous<br>Reliable or unreliable medium<br>2 protocol entities |
| **Modeling Formalism** | FSM model | FSM model | FSM model | FSM/Petri Net model |
| **Mode of Interaction** | Interactive | Interactive | Automatic<br>Fixed comm. pattern | Automatic<br>Fixed comm. pattern |
| **Protocol Properties** | Safety properties satisfied | Safety properties satisfied. | Safety properties satisfied. | Safety properties satisfied. |
| **Specific Functions** | None | None | None | Error-recovery |
| **Complexity** | Global reachability | Global reachability | Localized | Localized |

Table 2.2. Summary of the main features of non-service-oriented synthesis methods.

| | Sequence method [CHOI 86] | Component-based synthesis [KAKU 87] | Interactive protocol synthesis [ZHAN 88] | |
|---|---|---|---|---|
| **Starting Point** | Informal specifications (scratch) | Partial protocol specifications | Informal specifications | |
| **Constraints on Communication Model** | Asynchronous<br>Reliable medium<br>2 protocol entities | Asynchronous<br>Reliable medium<br>N protocol entities | Asynchronous<br>Reliable medium<br>2 protocol entities | |
| **Modeling Formalism** | FSM model | FSM model | FSM model | |
| **Mode of Interaction** | Interactive | Interactive | Interactive | |
| **Protocol Properties** | Safety properties satisfied. | Safety properties satisfied. | Safety properties satisfied. | |
| **Specific Functions** | None | None | None | |
| **Complexity** | Global reachability | Localized | Globalized | |

Table 2.2. Summary of the main features of non-service-oriented synthesis methods (continued).

## 2.6. Discussions and Conclusions

In the first part of this chapter, we have presented some issues related to the specification and validation of protocols. We have also emphasized the importance of formal description techniques for a rigourous, unambiguous and expressive specification of complex distributed systems, and of protocols and services, in particular. Furthermore, we have introduced the FSM specification model in detail because our research is mainly based on this model. Moreover, protocol design issues were discussed and are related to the FSM model.

In the second part, we presented a survey and assessment of several existing synthesis methods for protocol design. Each method was briefly described in terms of some specific classification features for synthesis methods.

Tables 2.1 and 2.2 summarizes the features of the synthesis methods described in the previous section. In general, we found that most methods do not make a clear distinction between a protocol and a service in a layered communication model. Consequently, most methods (except for [BOCH 86, CHUL 88a, MERL 83]) do not even refer to the service specification during the synthesis of the protocol.

By comparing the various methods with respect to the specific attributes discussed in Section 2.4.2.1, we can make the following general observations:

- Interactive methods allow flexibility in the design process. As a result, communication patterns need not be pre-specified, but may be constructed interactively.

- Methods that only consider the synchronous mode of behavior of communicating entities exclude a wide range of real-life protocols.

- Methods that make no reference to service requirements, do not guarantee the semantic correctness (liveness properties) of the synthesized protocol, and therefore require the application of a semantic verification procedure.

- Methods that generate a global reachability graph during the synthesis process suffer from state explosion and termination problems, and consequently are complex and inefficient.

- Most methods concentrate on the synthesis of the control part of the protocol entities which mainly consists of the exchange of synchronization messages. The data part is not adequately treated by any of the synthesis methods.

- Most methods are limited to the synthesis of two-party protocols , and therefore a wide range of real-life application protocols is not explicitly addressed.

- Other than the exchange of synchronization messages, some methods have been extended to deal with unreliable media by synthesizing error-recovery patterns. This is one example of a means of exploiting the power of the synthesis process in order to provide key protocol-specific functions.

In summary, it is clear that most existing methods do not adequately address two important aspects of protocol design:

a. **Semantic correctness** of the synthesized protocol (with respect to the service specifications). This is due to the fact that no reference has been made to a service specification, the ultimate point of reference of a protocol. This leads to the inability of most methods to prove that the designed protocol satisfies the service user requirements. Only Bochmann [BOCH 86] and Chu [CHUL 88a] start the protocol synthesis process from a complete service specification. However, the service specification models they use are very restrictive and are unable to express concurrent service behaviors existing in asynchronous communication systems. Furthermore, proofs for the semantic and syntactic correctness of the synthesized protocols are not provided.

b. **Protocol-specific functionalities** such as error recovery. This is possibly due to an oversimplification of the synthesis problem. Error recoverable protocol designs are only synthesized in the methods introduced by Ramamoorthy [RAMA 86] and Chu [CHUL 88a].

These two shortcomings of current protocol design techniques motivated us to propose and develop a new service-oriented protocol synthesis method.

# CHAPTER 3

# A SYNTHETIC LOCALIZED APPROACH TO PROTOCOL VALIDATION

In this chapter, we propose a simple and efficient validation technique for the detection and correction of logical design (syntactic) errors in communication protocols. The communication model consists of two entities asynchronously exchanging messages via lower modules modeled by a reliable full-duplex FIFO channel. The communicating entities are represented by acceptance trees, traces and sets (ATTS), a tree-based model which is general enough to express any labeled transition-based model, including the finite state machine model.

This chapter is organized as follows. In Section 3.1, we briefly introduce our localized validation technique and its main features. In Section 3.2, a new model, denoted the Acceptance Tree, Traces and Sets (ATTS) model, is introduced and operations on the model and its components are defined. A brief overview of the techniques used for the transformation of FSM specifications into ATTS-like specifications is also provided. In Section 3.3, the validation technique and its proof of correctness are presented. In Section 3.4, the technique is applied to examples of protocols. Finally, in Section 3.5, we conclude the chapter.

## 3.1. Introduction

In Section 2.3.1, we mentioned several approaches to the validation of FSM-based specifications of communication protocols, namely, closed covers [GOUD 84c],

reachability analysis [WEST 78, ZAFI 78], divide-and-conquer [CHOI 86b, LAMS 84, VUON 82], partial state space exploration [LINC 87, MAXE 87, WEST 86,89], and localized [BRAN 83, KAKU 86, YUAN 89]. These approaches addressed the explosion problems caused by the reachability analysis and the global protocol state exploration. However, in this chapter, we favor the localized approach to validation for the following reasons: (i) the localized validation procedure is not based on a global reachability analysis; therefore, the global state explosion and termination problems are avoided. (ii) since protocol entities can be validated independently from each other, the validation of protocol entities can be executed in parallel.

Our validation technique uses the concept of synthesized and inherited traces at nodes of an ATTS. The validation is performed in a bottom-up fashion which implies an early detection of complex erroneous behaviors. Very limited coupling between the two entities during the validation process is required; therefore this technique is denoted a localized approach to validation. In addition, the validation of the entities can be executed in parallel and the computations that are performed during the validation are very simple. Moreover, this technique can be integrated into an incremental synthetic protocol design methodology which guarantees freedom from logical errors and satisfaction of the service requirements. Therefore, this technique can be used for the validation of existing designs, and can be extended for the synthesis of new designs.

The advantages of this technique over other localized validation techniques are:

i) the global information required at each local entity is minimal and very simple to obtain and maintain.
ii) the computations involving local and global information are very simple.

iii) the technique is general enough to be applied for the validation of any labeled transition system-based specifications, including the FSM specifications of protocols.

iv) our technique can be extended into a synthesis method for the generation of new protocol behaviors.

For simplicity of the presentation of the validation technique, the communication model we consider is based on two communicating entities asynchronously exchanging messages via lower modules modeled by a reliable full-duplex FIFO channels. This involves no loss of generality for layered communications protocols (for example, OSI protocols). As well, the model and the technique are easily extended to handle more than two communicating entities.

## 3.2. The Specification Model

In this section, we first introduce the ATTS model for the representation of communicating entities. We also define some operations to manipulate the model, and we derive some key properties of the model. Finally, we briefly survey the work done on the transformation of labeled transition system-based specifications and specifically FSM specifications to the ATTS model.

### 3.2.1. Acceptance Trees, Traces and Sets (ATTS)

The ATTS model is based on the concept of *acceptance trees* introduced in [HENN 85]. The semantics of the model is quite similar to that of the acceptance trees in the case of deterministic systems. However, a minor modification is introduced for non-deterministic

systems. In fact, the ATTS model we introduce below is an adaptation of acceptance trees to communication protocol design and validation activities.

**Definition 3.1.** An *ATTS* is a tree that carries semantic information useful to perform computations on the behavior of an entity it describes. The nodes of an ATTS tree are labeled by an *acceptance set* and an *acceptance trace*. An acceptance set at node *n*, denoted by *A(n)*, includes all events that the process can perform at *n*. An acceptance trace at node *n*, denoted by *t(n)*, is the sequence of event occurrences that leads to *n* starting from the root node of the ATTS. This trace uniquely identifies a given node in the ATTS tree. This uniqueness is guaranteed by the the fact that the acceptance set at node *n* *(A(n))* cannot contain multiple occurrences of the same event. Therefore, a node *n* is uniquely characterized by the tuple $<t(n), A(n)>$.

Note that *t(n)* of the root node is a null trace ($\varepsilon$) and *A(n)*s of the leaves are empty sets ($\Phi$). Each root node identifier is a tree name; however, each leaf node identifier may be either: i) own tree name (recursive or cyclic behavior), ii) a new tree name (enabling a new phase), or iii) *end* or *exit*.

Internal non-deterministic behaviors can also be expressed in the ATTS model. For a node *n*, at which the process can proceed non-deterministically to node n', *A(n)* will include events of the form *ie* for every *e* in *A*(n'). Moreover, *t*(n') will be equal to t(*n*). The ATTS model is general enough to express deterministic or non-deterministic behaviors, synchronous or asynchronous and terminating or non-terminating (recursive) communicating systems. Furthermore, any labeled transition-based systems can be transformed into the ATTS model.

Figure 3.1 shows an example of an ATTS representation of a communicating entity P.

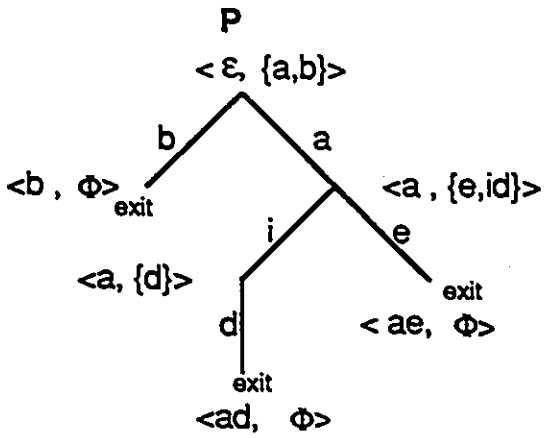



**Figure 3.1.** ATTS representation of P.

**Definition 3.2.** The set of the *final* traces of process P, denoted by $F(P)$, includes the acceptance traces at the *leaf* nodes of the ATTS representation of P.

For the example in Figure 3.1, $F(P) = \{b, ad, ae\}$.

**Definition 3.3.** *Last* (t) is the last event in the trace or event list *t*.

**Definition 3.4.** An event in a trace can be either a *send* event, denoted by *!e*, or a *receive* event, denoted by *?e*. If *e* is a *send (receive)* event, then $\sigma(e)$ is *send (receive)*. If $\sigma(e)$ = *send (receive)*, then $\sigma(\hat{}e)$ = *receive (send)*.

For example, if t1 = !a?b!c, then t2 = ^t1 = ?a!b?c.

**Definition 3.5.** Two events e1 and e2 are said to be *matching* events, denoted by *Matching* (e1, e2), if i) $\sigma$(e1) = *receive* and $\sigma$(e2) is either *receive* or *send* such that e1 = e2, or ii) $\sigma$(e1) = *send* and $\sigma$(e2) = *receive* such that e1 = e2.

**Definition 3.6.** The length of a trace *t*, denoted by *Length* (t) (or *L*(t)), is the number of events in the trace or event list *t*.

**Definition 3.7.** The height of a node *n*, denoted by *Height (n)* (or *H(n)*), is equal to *L(t(n))*.

**Definition 3.8.** The first *x* events of a trace *t* is denoted by (*x*) *t*. *Init(t)* is (*L(t)* -1) *t*..

**Definition 3.9.** The subtrace consisting of the last *x* events of *t* is denoted by *(x-) t*.

**Definition 3.10.** The position of event *e* in the trace *t* is denoted by *Pos (e, t)*.

**Definition 3.11.** The *ith* event in the trace t (with $0 < i \leq L(t)$) is denoted by [i]t.

**Definition 3.12.** The *concatenation* of two traces t1 and t2 is denoted by t1@t2.

**Definition 3.13.** The number of occurrences of an event e in a trace t is denoted by #(e, t).

**Definition 3.14.** An event $e_k$ is said to be *feasibly executable* at process behavior Pi communicating with Pj, that is, $e_k$ is in A(Pi), if:

1. $\sigma(e_k)$ = *send*, or
2. $\sigma(e_k)$ = *receive* and $\#(?e_k, t(Pi)) \leq \#(!e_k, t(Pj))$ and,

   $\forall e_i, e_k \in t(Pj) / \sigma(e_i) = \sigma(e_k) =$ *send*, $Pos(e_i, t(Pj)) < Pos(e_k, t(Pj))$ and

   $Pos(?e_i, t(Pi)) < Pos(?e_k, t(Pi))$.

For example, for t1 = !a?b in P1 and t2 = !b?a!c in P2, event !d is feasibly executable in P2 after executing t1 in P1. Similarly, ?c is feasibly executable in P1. However, ?m is not feasibly executable in P1.

**Definition 3.15.** For two traces t1 and t2 (in distinct processes), the *first-send-not-received* (t1,t2) is the event !e1 such that $\forall$ !e1, !e2 in t1 and Pos(!e1, t1) < *Pos*(!e2, t1), $\exists$ ?e1 in t2.

For example, for t1 = !a?b?c!d in P1 and t2 = ?a!b!c!e in P2, *first-send-not-received* (t1, t2) = !d, and *first-send-not-received* (t2,t1) = !e.

**Definition 3.16.** *Protocol validation* consists of finding and enumerating all protocol design errors (if any). In this work, and without loss of generality, we only consider syntactic errors like deadlocks, unspecified receptions and overflows.

**Definition 3.17.** The characterization of *compatible* traces depends on whether the communication progresses synchronously or asynchronously in the system. This is related to the degree of coupling between the communicating entities: a close or tight coupling implies a synchronous communication, and a loose coupling implies an asynchronous communication. The general case corresponds to asynchronous communication, in which a time delay occurs between the transmission of event *e* and its eventual reception at a peer entity. Synchronous communication corresponds to the special case in which the time delay is null.

In the general case, two traces t1 and t2 are said to be *compatible* if:

1. $\forall$ !e in t1, $\exists$ ?e in t2 / $Pos$(?e,t2) $\geq$ $Pos$(!e, t1), and $\forall$ ?e in t2, $\exists$ !e in t1 / $Pos$(!e,t1) $\leq$ $Pos$(?e, t2).
2. $\forall$ !e in t2, $\exists$ ?e in t1 / $Pos$(?e,t1) $\geq$ $Pos$(!e, t2). and $\forall$ ?e in t1, $\exists$ !e in t2 / $Pos$(!e, t2) $\leq$ $Pos$(?e, t1).
3. If $!e_i$ and $!e_j$ in t1 and $Pos(!e_i,t1) > Pos(!e_j,t1)$ then $Pos(?ei, t2) > Pos(?ej, t2)$.

Moreover, if t1 and t2 are final traces in $F$(P1) and $F$(P2), respectively, they are compatible if they satisfy the three conditions listed above, and if their leaf node identifiers are the same, that is, if $t$(n1) = t1 and $t$(n2) = t2, $Id$(n1) = $Id$(n2).

This general definition corresponds to the case of both synchronous and asynchronous communication between entities P1 and P2. In the case of synchronous communication, the general case is reduced to the following rule:

t1 and t2 are compatible if: t1 = ^t2, that is: $\forall$ $!(?)e_i \in$ t1, $\exists$ $?(!)e_i \in$ t2 / $Pos(e_i, t1) = Pos(e_j, t2)$.

For example, t1 = !e?d!c?k!a!b in P1 and t2 = ?e!d?c!k?a?b are compatible traces. Also, t'2 = !d?e!k?c?a?b is compatible with t1. However, t"2 = ?e!d!k!f?l?m is not compatible with t1.

**Lemma 3.1.** For successful synchronous communication between entities P1 and P2, the following two conditions are always satisfied:

i) $\forall$ t1 $\in$ $F$(P1), $\exists$ t2 $\in$ $F$(P2) / t1 = ^t2, and

ii) $\forall$ t2 $\in$ $F$(P2), $\exists$ t1 $\in$ $F$(P1)/ t2 = ^t1.

**Proof:** The synchronous communication is based on the rendez-vous between a message transmission and its reception. Therefore, for every message transmission (reception) at position *i* in t1, there must exist a message reception (transmission) at the same position in t2. If for example, none of the two conditions is satisfied, that is for !e ∈ t1 such that p = *Pos*(!e, t1), ∃ t2 such that [p]t2 = ?e, then a deadlock will occur in the system.

This lemma imposes a necessary and sufficient condition to avoid deadlocks and unspecified receptions, since message collisions are not possible ■

**Lemma 3.2.** In the case of asynchronous communication between processes P1 and P2, the following two conditions are satisfied:

i) ∀ t1 ∈ *F*(P1) , ∃ t2 ∈ *F*(P2) / t1 and t2 are two compatible traces, and

ii) ∀ t1 ∈ *F*(P2), ∃ t2 ∈ *F*(P1)/ t1 and t2 are two compatible traces.

**Proof:** Lemma 3.2 imposes a necessary (but not sufficient) condition for deadlock and unspecified reception avoidance in the communication system.

We prove the necessity of this Lemma by considering the case in which it is not satisfied.

Suppose t1 ∈ *F*(P1) and ∃ t2 ∈ *F*(P2) such that t1 and t2 are compatible, that is any t2 ∈ F(P2) satisfies either:

(1) Id(t1) ≠ Id(t2) or

(2) t1 and t2 are decomposable as follows: t1 = t'1 @ t"1 and t2 = t'2 @ t"2, t'1 and t'2 are compatible and (1)t"1 and (1)t"2 are not executable. Two possibilities exist:

a. $\sigma((1)t"1)$ is send, $\sigma((1)t"2)$ is receive and $(1)t"1 \neq (1)t"2$, then an unspecified reception exists. Similarly, $\sigma((1)t"2)$ is send, $\sigma((1)t"1)$ is receive and $(1)t"1 \neq (1)t"2$, then an unspecified reception exists.

b. $\sigma((1)t"1)$ and $\sigma((1)t"2)$ are receive, therefore a deadlock exists since t'1 and t'2 are compatible.

It is a non-sufficient condition, since $F$(P1) and $F$(P2) could be both missing some possible executable traces ■

**Note** that Lemma 3.1 is a necessary and sufficient condition to avoid deadlocks and unspecified receptions in the design; however, Lemma 3.2 is only a necessary (not sufficient) condition.

**Definition 3.18.** Two traces, t1 and t2, are said to be simultaneously *executable*, if they can be decomposed into t'1, t"1 and t'2, t"2, respectively, in which, t'1 and t'2 are compatible traces (or empty traces) and every event in t"1 and t"2 is a feasible executable event.

For example, t1 = !a?b!c!d in P1 and t2 = !b?a?c!k in P2 are simultaneously executable since t1 and t2 are decomposable into !a?b!c @ !d (t'1@t"1) and !b?a?c @ !k (t'2@t"2), respectively. t'1 and t'2 are compatible and events !d and !k are feasibly executable in P1 and P2, respectively.

The existence of simultaneously executable traces guarantees the absence of deadlocks: however, unspecified receptions are still possible.

**Definition 3.19.** *Trace decomposition.* From the definitions of compatible and executable traces, we can conclude that traces t1 and t2 of two communicating processes can be decomposed into (t'1@t"1) and (t'2@t"2), respectively, in which the subtraces t'1 and t'2 are compatible and the subtraces t"1 and t"2 are executable. Since the lengths of compatible subtraces must be identical, only three possibilities exist when decomposing t1 and t2:

1. t'1 and t'2 are empty, then t1 and t2 are executable traces but not compatible.
2. t"1 and t"2 are empty, then t1 and t2 are compatible, therefore, executable.
3. no subtrace is empty, then t1 and t2 are said to be executable but not compatible.

For example, t1 = !d and t2 = !k are executable but not compatible traces. However, !a?b!c and !b?a?c are compatible and executable traces. Also, t1 = !a?b!c!d and t2 = !b?a?c!k are executable but not compatible traces.

**Definition 3.20.** Two traces, t1 and t2, are said to be *colliding*, denoted by *Colliding*(t1,t2), if $\exists$ x (x> 0) / (x) t1 and (x) t2 are not subtraces of compatible traces t'1 and t'2, respectively.

The term 'colliding' derives from the fact that processes executing simultaneous sends can cause collisions in the transmission medium. In the next section, we show how to handle such collisions.

For example, t1 = !a?b!c!d and t2 = !b?a?c!k are incompatible colliding traces, since, for x = 3, subtraces !d and !k are not compatible.

**Lemma 3.3.** For every pair of incompatible colliding traces t1 $\in$ *F*(P1) and t2 $\in$ *F*(P2), synchronizing traces must exist in *F*(P1) and *F*(P2) for handling collisions.

**Proof:** Let t1 and t2 be two incompatible colliding traces decomposable as follows:

t1 = t'1 @ !e1 @ t"1

t2 = t'2 @ !e2 @ t"2

in which t'1 and t'2 are compatible traces (t"1 and t"2 can be empty traces).

We prove this Lemma by enumeration of cases.

For all traces T2 ∈ F(P2) decomposable into T2 = t'2 @ !e2 @ T"2 and (1)T"2 ≠ ?e1, one of two possibilities exists:

a. σ((1)T"2) is receive other than ?e1, and since t'1 and t'2 are compatible, First Sent First Received requirement is not respected, and an unspecified reception exists in P2.
b. σ((1)T"2) is send.

For all traces T1 ∈ *F*(P1) decomposable into T1 = t'1 @ !e1 @ T"1 and (1)T"1 ≠ ?e2, one of two possibilities exists:

a. σ((1)T"1) is receive other than ?e2, and since t'1 and t'2 are compatible, First Sent First Received requirement is not respected, and an unspecified reception exists in P1.
b. σ((1)T"1) is send ■

### 3.2.2. Transformations from the Labelled Transition-Based Models to the ATTS Model

Since the ATTS model is a tree-based model augmented with some information, we can make use of existing techniques for the transformation of labeled transition-based systems into trees.

The execution model for LOTOS specifications can be transformed into a tree representation using an expansion procedure [ISO 8807, MILN 80].

Zafiropulo [ZAFI 80] uses a tree protocol representation of a protocol entity for the synthesis of protocols. Also, Yuang [YUAN 89] developed an algorithm for transforming FSM specifications into an expanded tree representation. The latter technique can be easily adapted to become a transformation procedure from FSM into ATTS representation which is the requirement for the application of our synthetic validation technique.

## 3.3. The Validation Technique and its Proofs of Correctness

In this section, we first describe the features of the validation technique. Then we outline two algorithms for the implementation of the technique. Finally, the proofs for the correctness of the technique are presented.

### 3.3.1. Features of the Validation Technique

The validation technique consists of two phases. In the first phase, the ATTS representations of the processes to validate are augmented with some semantic information in the form of traces of two types: *synthesized* and *inherited*. These terms are derived from attributed grammars [KNUT 68]. In the second phase, the detection and correction of logical design errors are performed.

Given the ATTS representations of two communicating protocol entities P1 and P2, two types of traces are computed for each node of the ATTSs: the inherited trace carries information about the ATTS where the node exists. However, the synthesized traces transfer information about traces from the other ATTS.

The validation procedure is essentially based on the examination of the inherited and the synthesized sets of traces at each of the nodes of the ATTS representing the process under validation. Synthesized traces at one node are propagated upwards to their (unique) parent node depending on some rules for compatibility and executability. The design errors detected by the procedure can be corrected before proceeding with the validation.

Furthermore, the validation is done in a bottom-up fashion which results in an early detection of complex erroneous behaviors and an efficient technique to detect errors since a trace at a leaf node identifies a unique path in the tree. The validation procedure is then applied twice: first, P1 is validated with respect to P2, and then, P2 is validated with respect to P1 (or vice versa). Very limited coupling between the two entities during the validation process is required; therefore this technique is classified under the localized approach to validation. In addition, the validation of the entities can be executed in parallel and the computations that are performed during the validation are very simple. Finally, the technique could be extended to validate against the service requirements in addition to the detection of logical design errors.

### 3.3.2. Algorithms

In the following, we present two algorithms which implement the validation technique: Algorithm-G for the generation of inherited and synthesized traces at nodes of an ATTS

representation of a process, and Algorithm-D for the detection and correction of logical design errors in the given protocol. These algorithms were not implemented or tested, but based on the proofs that follow later in this section, these algorithms are correct.

**Algorithm-G for the generation of inherited and synthesized trace sets:**

**Input:** Labelled transition-based system representation of communicating entities to validate.

**Output:** ATTS representation augmented with inherited and synthesized traces.

**Steps:**

1. Initialization. For each of the communicating entities to validate, generate its ATTS representation.
2. Generate the inherited and synthesized traces at the ***leaf nodes*** of each of the ATTSs.

   For a leaf node $n$ in P1's ATTS:

   . $Ip(n) = t(n)$.

   . $Sp(n) = \{ t / t \in F(P2)$ and $H(n) \geq L(t) \} \cup$

   $\{t / \forall \pi \in F(P2)$ and $H(n) < L(\pi), t = (H(n))\ \pi \}$.
3. Generate the inherited and synthesized traces at other nodes.

   For a non-leaf node $n$ in P1's ATTS:

   . $Ip(n) = t(n)$.

   . $Sp(n) = \{ t \in Sp(n-1) / L(t) < H(n) \} \cup$

   $\{ Init(t), t \in Sp(n-1)$ / $Matching(Last(t), Last(Ip(n-1))$, and

   $Init(Ip(n-1))$ and $t$ are executable$\}$

Where $Sp(n-1)$ denotes the union of the sets of synchronized traces at nodes dependent of $n$.

Algorithm-G shows that the only information transferred from the *ATTS* representing P2 is the synthesized sets of traces at the leaves of P1. These traces correspond to the inherited traces at the leaves of P2. The synthesized sets of traces at nodes other than the leaves are generated using the synthesized sets of dependent nodes propagated upwards in the ATTS.

**Definition 3.21.** *Collision handling synthesis (analysis) rule.* Let t1 ∈ $F$(P1) and t2 ∈ $F$(P2) be two colliding traces. Suppose t1 and t2 are decomposed into t'1 @ !e1 @ t"1 and t'2 @ !e2 @ t"2, respectively, where t'1 and t'2 are compatible traces. The handling of a collision caused by simultaneous transmissions of e1 and e2 can be done by considering (verifying that) one of the three cases:

a. P1 has a higher priority than P2. Add t11 = t'1 @ !e1 @ ?e2 @ t"1 to $F$(P1), and t21 = t'2 @ !e2 @ ?e1 @ ^t"1 to $F$(P2).
b. P2 has a higher priority than P1. Add t11 = t'1 @ !e1 @ ?e2 @ ^t"2 to $F$(P1), and t21 = t'2 @ !e2 @ ?e1 @ t"2 to $F$(P2).
c. P1 and P2 have equal priorities. Add t11 = t'1 @ !e1 @ ?e2 @ t"1 @ ^t"2, t12 = t'1 @ !e1 @ ?e2 @ ^t"2 @ t"1 to $F$(P1), and t21 = t'2 @ !e2 @ ?e1 @ t"2 @ ^t"1, t22 = t'2 @ !e2 @ ?e2 @ ^t"1 @ t"2 to $F$(P2).

For example, Let t1 = !a?b?c!d?e ∈ $F$(P1) and t2 = !b?a!c!f?g ∈ $F$(P2) be two colliding traces decomposable as follows. t1 = t'1 @ !d @ ?e and t2 = t'2 @ !f @ ?g. The collision rules imply that there must exist in $F$(P1) and in $F$(P2) additional colliding and compatible traces depending on the three cases defined above.

a. !a?b?c!d?f?e ∈ $F$(P1) and !b?a!c!f?d!e ∈ $F$(P2)

b. !a?b?c!d?f!g ∈ $F$(P1) and !b?a!c!f?d?g ∈ $F$(P2)

c. !a?b?c!d?f?e!g and !a?b?c!d?f?e!g ∈ $F$(P1), !b?a!c!f?d?g!e and !b?a!c!f?d!e?g ∈ $F$(P2)

Note that cases a and b correspond to the winner/loser approach to synchronization introduced by Gouda [GOUD 84b]. However, case c corresponds to the no-winner/no-loser collision resolution (which is characteristic of multiple concurrent updates in a distributed database application).

**Lemma 3.4.** The application of the collision handling rule to every pair of colliding traces avoids the occurrence of unspecified receptions.

**Proof:** Let F'(P1), F"(P1), F"'(P1) and F'(P2), F"(P2), F"'(P2) be subsets of $F$(P1) and $F$(P2), respectively. Let F'(P1) be the set of traces non-colliding and compatible with traces in F'(P2), F"(P1) be the set of traces colliding and incompatible with traces in F"(P2), and finally, F"'(P1) be the set of traces colliding and compatible with traces in F"'(P2).

The application of the collision handling rule (Definition 3.21) to a pair of incompatible colliding traces in F"(P1) and F"(P2) would generate colliding and compatible final traces which may or may not exist in F"'(P1) and F"'(P2). Eventually, after successive applications of the collision handling rule to pairs of colliding traces, no new traces would be generated, and F"(P1) and F"(P2) become empty sets ■

**Lemma 3.5.** Only incompatible colliding traces can be used to detect unspecified receptions.

**Proof:** We prove this Lemma by showing that any other type of trace can never detect unspecified receptions.

Suppose t1 and t2 are non-colliding or compatible final traces. Then, t1 and t2 satisfy one of the following three cases:

(a) t1 and t2 are compatible and non-colliding, that is t1 = ^t2 (synchronous progress). The simultaneous occurrence of send and receive of an event e guarantees that the occurrence of an unspecified reception is impossible; therefore, no new traces can be generated.

(b) t1 and t2 are compatible and colliding traces (asynchronous progress); then, according to the definition of compatibility, all messages sent are received and hence the occurrence of an unspecified reception is impossible.

(c) t1 and t2 are incompatible and non-colliding; then, according to the definitions of compatibility and collision, t1 and t2 can never synchronize since (1)t1 and (1)t2 are not executable.

Therefore, compatible colliding traces and compatible non-colliding traces are not useful for the detection of unspecified receptions ■

Based on Lemma 3.5, the following optimization rule can be applied for the elimination of useless synthesized traces generated by Algorithm-G at a node $n$ of an ATTS:

$\forall\, t \in Sp(n)$, if *not-Colliding* $(t, Ip(n))$ is true then eliminate $t$ from $Sp(n)$.

**Lemma 3.6.** A protocol design causes a channel overflow if $\exists$ t1 (t2) $\in F(P1) \setminus F(P2)$ / $\forall\, e \in$ t1 (t2), $\sigma(e) =$ *send*.

**Proof.** The presence of a trace consisting of event transmissions only can potentially lead to an overflow when an asynchronous communication is assumed. Only the following

scenario of a trace of event transmissions would not cause such error: $L(t) \leq b$ , where b is the channel capacity, and $Id(n)$ is not own ATTS name, for n whose $A(n) = t$ such that $A(n')$ (where n' is the root node of the enabled ATTS) does not include an event transmission ■

Before proceeding to Algorithm-D, we state a theorem for proving the correctness of the validation technique.

**Theorem 3.1.** The satisfaction of the rules for compatibility (Lemmas 3.1 and 3.2), collision handling (Lemmas 3.3, 3.4, and 3.5) and overflows (Lemma 3.6) guarantees that deadlocks, unspecified receptions and overflows are avoided.

**Algorithm-D for the detection and correction of logical design errors:**

**Input:** ATTS representation augmented with inherited and synthesized traces.

**Output:** ATTS representation free from deadlock and unspecified receptions.

**Steps:**

1. For each t1 in $F$(P1) there must exist t2 in $F$(P2) such that t1 and t2 are compatible. Similarly, for each t2 in $F$(P2), there must exist t1 in $F$(P1) such that t1 and t2 are compatible.
2. Detect the existence of *unspecified receptions* at each node of the ATTS.
   At a node $n$, let $t \in Sp(n)$.
   . If *non Matching*($Last(t)$, $Last(Ip(n))$ ) and *First-sent-not-received*($t$, $Ip(n)$) = !e and ?e $\notin A(n)$ then Unspecified Reception (?e) at $n$.
3. If at node $n$ a reception is missing, add new traces according to Definition 3.21.

Step 1 guarantees that no deadlock is possible for synchronous communication. Step 2 checks if for every pair of colliding traces, all possible traces for handling the collision are present in the behavior of the entity to validate. Step 3 generates all those synchronizing traces absent from the behavior of the entity as a result of the detection of colliding traces and unspecified receptions.

The correction of an unspecified reception (?e) at a node $n$, characterized by $<\iota(n), A(n)>$, is done by the generation of a new final trace $t = \iota(n)$ @ ?e @ t'. The trace $t$ must be compatible with another trace in $F$(P2). Furthermore, t must be a correct trace with respect to the service definition; hence, it is a semantically valid trace. Since the new final trace in $F$(P1) is compatible with another trace in $F$(P2) and because of Lemma 3.5, the re-application of the validation is not necessary.

In the case of non-deterministic ATTSs, all internal events i must be converted to !δ (send dummy event), where δ is not to be received by any other ATTS. Then, the two algorithms described above are applicable for the validation of the non-determinstic communicating entities.

The validation technique for one ATTS can be applied in two ways. The two-phased application in which we first apply Algorithm-G to generate all synthesized traces at all nodes, and then apply Algorithm-D for the detection of logical errors. The interleaved application in which for every node, we first apply Algorithm-G, and then apply algorithm-D. Furthermore, the validation of the ATTSs can be performed concurrently.

Based on Lemma 3.2, if the validation of the ATTSs is applied sequentially, the validation of one ATTS may optimize the validation of the other ATTS. The termination of both

algorithms is also guaranteed since both the number of final nodes and the number of pairwise incompatible colliding traces are finite.

## 3.4. Examples

In this section, we demonstrate the application of the validation technique to three protocol specification examples. The first example is taken from [YUAN 89] and shows the simplicity of our technique compared to the one described in the paper referenced above. The second example shows how multiple occurrences of design errors can be detected and corrected. Finally, the third example shows the application of the technique to the validation of the Call Setup procedure of X.25.

### Example 3.1:

The finite state machine representations of two communicating processes, P1 and P2, are given in Figure 3.2. Their ATTS representations are shown in Figure 3.3. Figure 3.4 shows the synthesized and inherited traces at each node of process P2 to validate.

The application of Algorithm-G starting from node n3 and upwards proceeds as follows:

0. $F$(P1) = { !2?4?3, !2??2?4?3, ??2!4?3 }
1. $I$(n3) = $r$(n3) = {!2?4!3}, $S$(n3) = {!2?4?3, !2??2?4, ??2!4?3 }
   After applying the optimization rule, $S$(n3) becomes {!2?4?3, !2??2?4 }.
2. $I$(n2) = {!2?4 }, $S$(n2) = { !2?4, !2??2 }
3. $I$(n1) = {!2 }, $S$(n1) = {!2 }

After applying Algorithm-D, an unspecified reception is detected at node n1. The event ?2 must be added to the acceptance set $A$(n1). This implies that a new trace starting with !2?2 must be added to the set of final traces $F$(P2). Based on Lemma 3.1, this trace must be

compatible with a trace in $F$(P1). Therefore, the new final trace suggested by the algorithm is !2?2!4!3.

**Note.** In fact, the two communicating processes do not satisfy the condition of Lemma 3.1, which states that for every final trace in $F$(P1) there should exist a compatible final trace in $F$(P2) and vice versa. Therefore, we can see that $F$(P2) is missing a trace compatible with $t$ = !2?2?4?3 $\in$ $F$(P1).

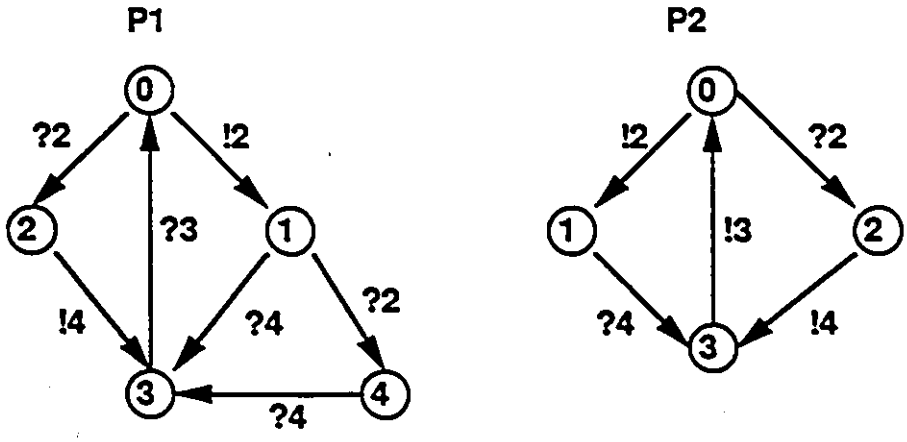



Figure 3.2. Two communicating protocol entities (Example 3.1).

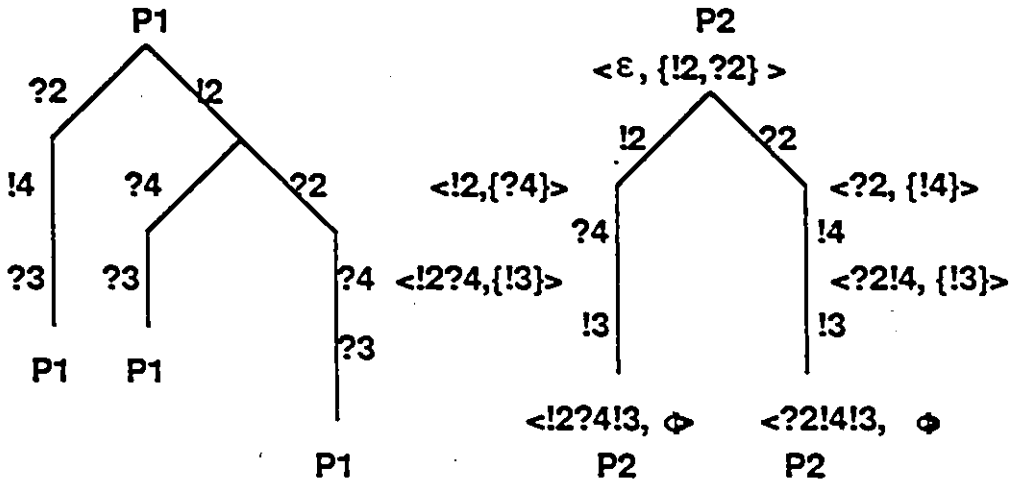



Figure 3.3. ATTS representations of the protocol entities of Figure 3.2.

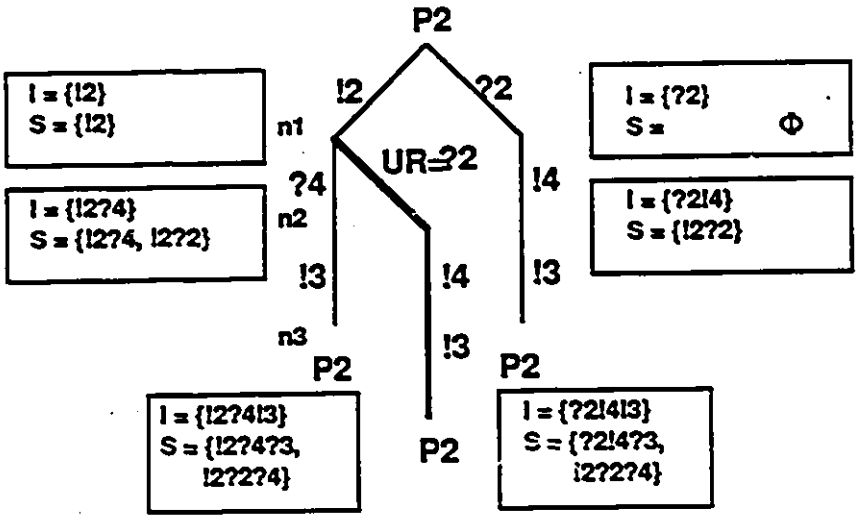



Figure 3.4. Synthesized and inherited traces at nodes of P2's ATTS of Figure 3.3.

**Example 3.2:**

The ATTS representations of the two communicating processes to validate are given in Figure 3.5. The synthesized and inherited traces at each node of the ATTS representing the process to validate are shown in Figure 3.6. This example shows how multiple occurrences of logical design errors can be detected and corrected by the validation technique.

0. $F$(P1) = { ?b?c?e!g, ?b?c!f?h, !a?d }
1. $I$(n4) = $r$(n4) = {!b!c!e?g}, $S$(n4) = { ?b?c?e!g, ?b?c!f?h, !a?d }

   After applying the optimization rule, $S$(n4) becomes { ?b?c!f?h, !a?d }.
2. $I$(n3) = { !b!c!e }, $S$(n3) = {!a?d, ?b?c!f }

   Two unspecified receptions are detected at node n3. ?a and ?f must be added to $A$(n3).
3. $I$(n2) = { !b!c }, $S$(n2) = { !a?d }

   One unspecified reception is detected at node n2. Event ?a must be added to $A$(n2).
4. $I$(n1) = {!b }, $S$(n1) = {!a }

   One unspecified reception is detected at node n1. Event ?a must be added to $A$(n1).

The additions of the unspecified receptions at nodes n1, n2 and n3 implies that new traces must be added to $F$(P2). These traces are: !b?a @ t1, !b!c?a @ t2, !b!c!e?a @ t3 and !b!c!e?f @ t4. Each of these new traces must be compatible with one final trace in $F$(P1).

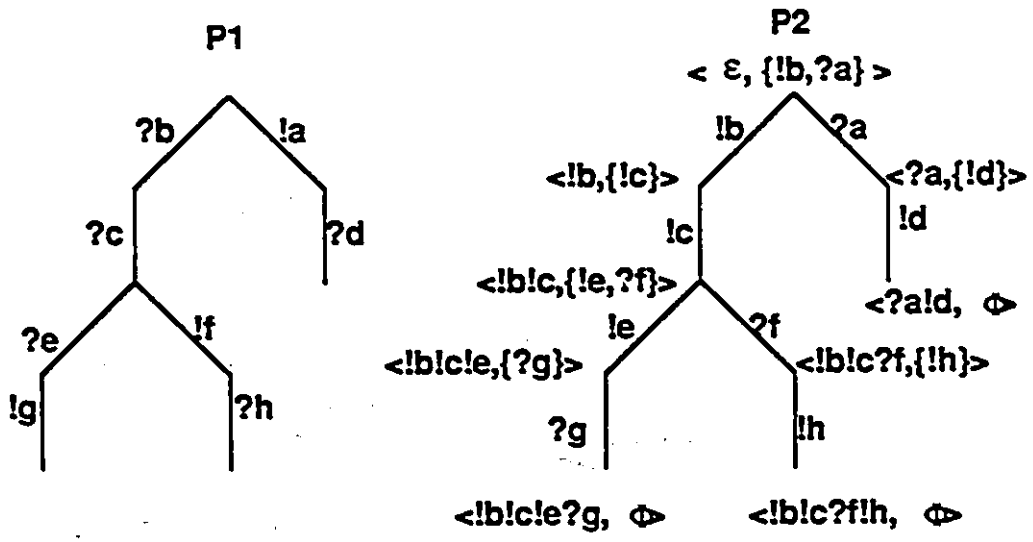



Figure 3.5. ATTS representations of P1 and P2 (Example 3.2).

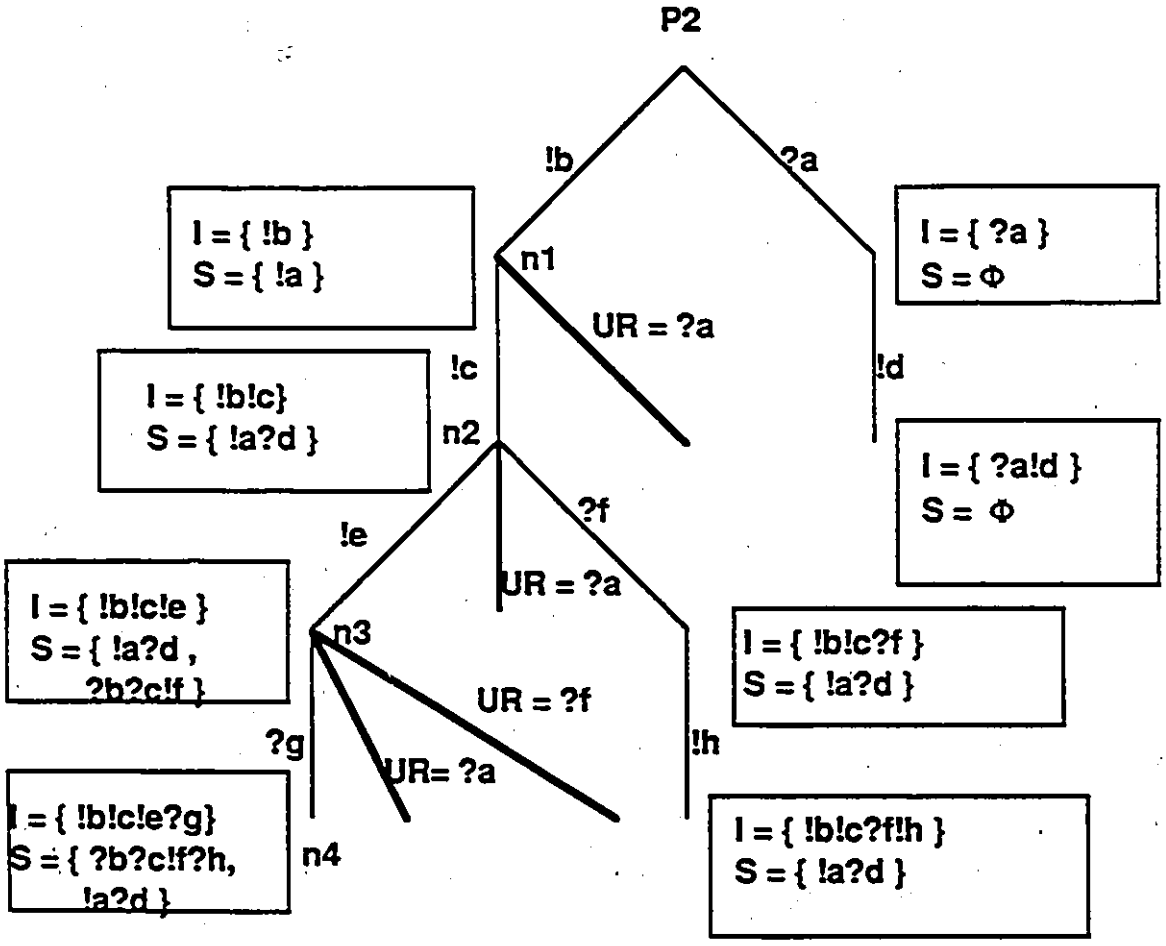



Figure 3.6. Synthesized and inherited traces at nodes of P2's ATTS in Figure 3.5.

**Example 3.3:**

The ATTS representations of an incomplete specification of the DTE and DCE involved in the Call Setup procedure of X.25 is given in Figure 3.7. The application of the technique detects the omission of possible traces and suggests possible corrections, as shown in Figure 3.8. The example shows, informally, how the technique can be applied to the synthesis of new traces or behaviors, while satisfying the service requirements.

The application of Algorithm-D indicates that two unspecified receptions ?IC and ?CR exist in DTE and DCE, respectively. Two new traces must be generated: !CR?IC @ t1 in DTE and !IC?CR @ t2 in DCE. Based on Lemma 3.2, these two traces must be compatible. t1 and t2 are chosen according to the protocol requirements. For example, the reception of an incoming call (?IC) by the DTE after the transmission of a connection request (!CR) must be ignored by the DTE. This corresponds to the (English) procedures for Incoming Call handling specified in the X.25 protocol. Therefore, the new traces should be: !CR?IC?CC in DTE and !IC?CR!CC in DCE.

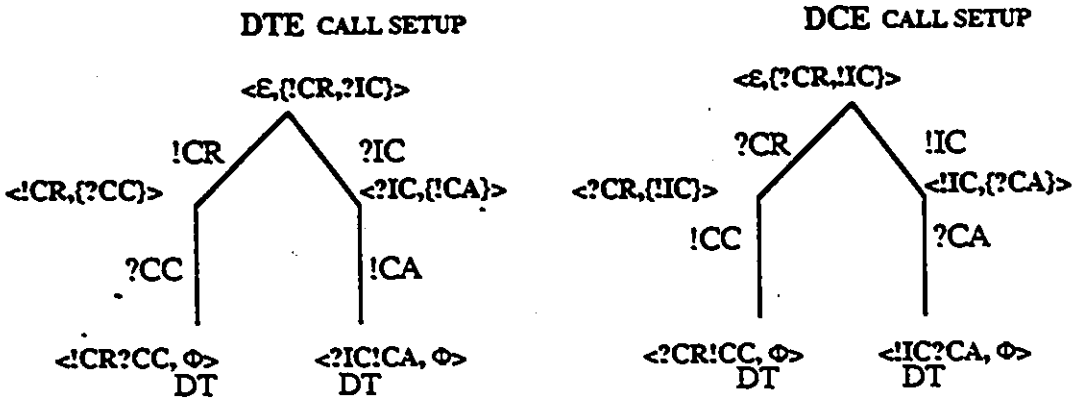



**Figure 3.7.** ATTS representations of DTE and DCE (Example 3.3).

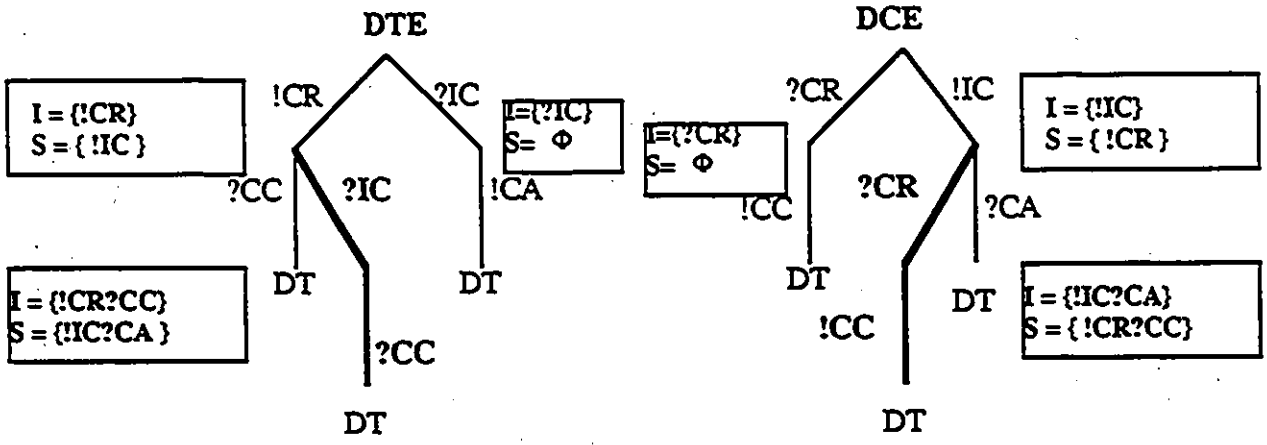



**Figure 3.8.** Synthesized and inherited traces at nodes of DTE and DCE.

## 3.5. Discussions and Conclusions

In this chapter, we presented a synthetic protocol validation technique for the detection of logical design errors such as deadlocks and unspecified receptions. The technique is simple and requires little computation on the model of communicating entities. Very limited coupling between the communicating entities is necessary during validation; therefore, the explosion problems of traditional global validation the reachability technique are avoided. Error correction is based on the generation of new event occurrences, hence the emphasis on generation (or synthesis) of new traces in the model. Therefore, it seems quite natural that the technique be integrated into a synthetic protocol design method. The validation technique can also be applied to non-deterministic behaviors in the communication model and can be generalized to involve more than two communicating entities. Finally, the tree-based modelling formalism (ATTS model) we used can be applied to any labeled transition system.

The advantages of our technique over other global and localized validation techniques are: i) the global information required at each local entity is minimal and simple to obtain and maintain, ii) the computations on local and global information are very minimal, and iii) the technique is general and can be applied to any labeled transition system-based specification.

The main limitation of the technique and the model at their current stages is that no validation against the service requirements is performed. In our opinion, this can be achieved when we integrate the validation technique into a synthetic design method, in which existing and synthesized traces are validated against the service requirements. Other areas to investigate are: i) the validation of communicating systems decomposed into more than one ATTS, and ii) a complexity analysis and comparison with other localized validation techniques.

This concludes the part of the thesis dealing primarily with analysis. However, ideas from this work formed the basis for the development of the service-oriented synthesis method which is described and justified in the remainder of the thesis.

# CHAPTER 4

# THE DUALITY BETWEEN SERVICE AND PROTOCOL SPECIFICATIONS

In this chapter, we present some philosophical, conceptual issues concerning the duality relations between service and protocol specifications. In Section 4.1, we introduce the service concept in communication systems and the rationale for designing protocols starting from the service specification. In Section 4.2, we describe some desirable features for protocol and service specifications. In Section 4.3, we discuss the requirements for protocol-oriented service specifications. In Section 4.4, we discuss issues related to the design of service-oriented protocols. Finally, in Section 4.5, we conclude the chapter.

## 4.1. The Service Concept

The distinction between the service and protocol concepts has acquired an increasing level of attention and interest by communication software designers [VISS 85]. Presently, any new international standard for a specific protocol is accompanied by another standard for the service requirements specification. However, although many protocol design methods have been proposed, relatively few of them make a distinction between the two concepts (Chapter 2). We feel that a good understanding of the differences between these concepts is essential for the development of an effective methodology for designing protocols efficiently and correctly.

The problem of protocol design should be tackled using a top-down refinement approach to communication systems. The primary objective of a communication system is to offer a

service. Therefore, at a high level of abstraction, a communication system can be viewed as a service provider offering some services to users accessing the system through many distributed *Service Access Points* (SAPs) using some service functions called *Service Primitives* (SPs) (Figure 4.1). At a more refined (lower) level of abstraction, the communication services are provided to the service users by a number of cooperating protocol entities which exchange protocol messages that are not observable at the upper SAPs. A protocol entity uses the lower service functions to relay protocol messages to another protocol entity. Furthermore, a protocol entity contributes to the service at the local SAP, and must satisfy the local service constraints. Also, the entity synchronizes with other protocol entities to ensure the global ordering of SPs at all SAPs in order to satisfy the global service constraints.

Two distinct concerns exist when we consider a distributed system: the user's and the designer's concerns. The user's concerns are mainly related to the services or functions to be offered by the system and their conformance to the requirements, whereas the designer's concerns are related to the ways these services or functions should be implemented.

Because of the distributed nature and complexity of distributed systems in general, and of communication systems in particular, two basic architectural concepts must be used to address the two concerns (Figure 4.2) [VISS 89]:

1. The *separation of functionality* concerns (horizontal decomposition): these are the user's concerns which are best described by a "Service Definition", whereas, the designer's concerns are described by a "Protocol Definition". This decomposition is driven by the complexity of the system.

2. The *separation of locality* concerns (vertical decomposition): different users have different interests, and are associated with different locations or sites through which

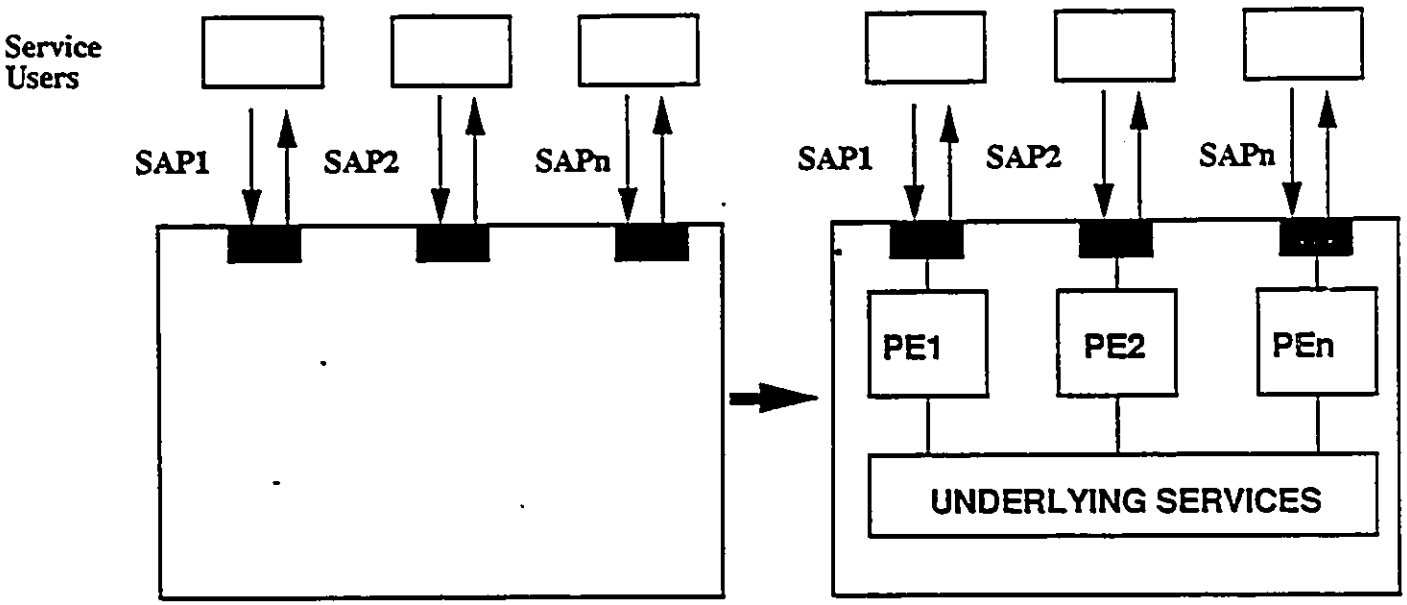



**Figure 4.1.** An abstract view of a distributed communication service and its refinement.

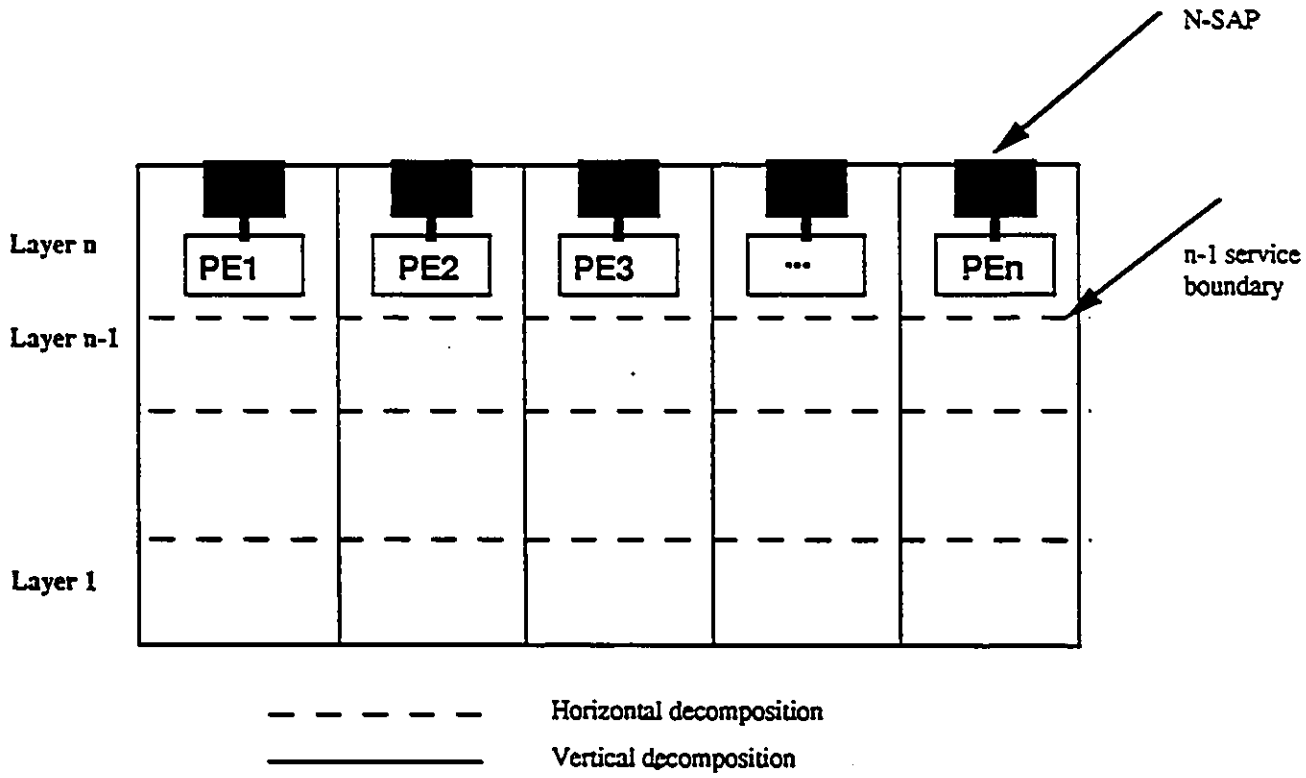



**Figure 4.2.** Horizontal and vertical decompositions of a communication system.

they may access the services offered by the system. This decomposition is driven by the distributed nature of the system.

In fact, the OSI reference model architecture was developed with these two concerns in mind. The different layers of the model correspond to the horizontal decomposition in which each layer is concerned with some specific functions to be provided at the upper horizontal boundaries. On the other hand, the distribution of the access points at each layer's interface are derived from the vertical decomposition in which we are concerned with the location of each service user.

In Chapter 2, we have introduced two approaches to protocol design: the *analytic* and the *synthetic* approach, and we have shown the advantages of the latter. From a synthesis perspective, the service specification should be the starting point for the design of protocols, since a protocol is a refinement of the service and therefore it is normally much complex to specify. Having this refinement point of view in the context of the OSI layered architecture, the *protocol design problem* is transformed into the problem of "designing an (N-)protocol from both (N-) and (N-1) Service Specifications".

Furthermore, a complete and unambiguous service specification helps to eliminate a lot of complex protocol design problems. Therefore it seems quite natural and trivial to start the protocol design process directly from the complete service specification.

Another reason for specifying the service prior to designing the protocol is that once the service is specified according to the (service) user requirements, a service provider prototype can be delivered to the user. Consequently, user applications that use this service can be developed faster, earlier and with less interfacing errors. Moreover, we should be able to design and develop test cases earlier in the development cycle. This is similar to

designing a software product starting from an unambiguous user requirements definition and rigorous functionai specifications.

## 4.2. Desirable Features for Protocol and Service Specifications

For the specification of protocols and services, we assume that we are using a suitable specification model or language with desirable attributes like modular constructs and unambiguous semantics. A list of desirable features for specification techniques was presented in Section 2.1.2.1. However, the existence of such desirable attributes is not enough to guarantee or enforce similar desirable attributes on specifications using those techniques, since designers can use different styles to specify the same object by using the same specification language. For example, although a specification language includes constructs for modularity, a designer can still design in a non-modular fashion.

Therefore, in the following we describe three important features of specification styles for services and protocols, namely modularity, preciseness and completeness.

a. Specifications must be as *modular*, i.e., as needed to enhance their *readability, maintainability, traceability, expandability (open-endedness), testability and verifiability*.

b. Specifications, at a certain level of abstraction, must be *precise*, i.e., so as to avoid any misinterpretations and ambiguity at a more refined (or lower) level. This feature also affects the testability and completeness of the refined specifications.

c. *Completeness* of the specification, i.e., is an important factor for the success of a system, since our goal is to capture and satisfy all system requirements. This feature also affects the testability of specifications from the completeness viewpoint.

It is obvious that these attributes are interrelated, and affect other desirable attributes. For example, the testability of a specification is strongly related to its modularity, conciseness and completeness. We feel that there is a need for further research on how to promote specification styles with such desirable features, and more specifically, how to evaluate the adherence of a specification to a certain desirable style. However, this topic is beyond the scope of this thesis.

## 4.3. Complete and Correct Protocol-Oriented Service Specification

From a synthesis viewpoint, starting from a given service specification, our ultimate goal is to "*synthesize protocol specifications*". Therefore, the starting point is to specify the service, which involves the correct and complete capture of the service requirements. Moreover, the service specification will be the reference for all validation aspects of the protocol synthesis method. The *service specification* includes:

i) the definition of the service functions or primitives available for use by the service users.

ii) the specification of the possible temporal ordering of SP occurrences at the distributed SAPs. This ordering is referred to as the global service constraints. To note here that the local service constraints at a particular SAP can be obtained from the global constraints by simply projecting the latter onto that particular SAP, that is, the local contraints at a particular SAP are obtained by hiding the SP occurrences at other SAPs from the global constraints.

Because of its relative simplicity, the service specification can be described in a monolithic and enumerative (scenario-based) style in which all possible service interactions at the distributed SAPs are listed. Furthermore, the service specification model must be capable of describing concurrent service users behaviors, and therefore the specification must also include all possible concurrent interactions of SPs and their handling. This also requires that the resolutions for SP collisions, due to concurrent interactions, be defined at the service specification level because any incompleteness at this level would undoubtedly lead to an incompleteness error in the protocol design. Furthermore, the service specification must take into consideration the architectural model underlying the service. In some cases, the service specification, as described above by traces of legal sequences of SPs, is not enough to provide an unambiguous interpretation at the protocol design level. For example, for a sequence CONREQ CONREJ observed at one SAP, it is not clear (at the protocol design level) whether the rejection of the request is a local decision of the underlying protocol entity or a decision of the peer entity. The avoidance of such ambiguity must be accomodated by the specification model at the service specification level, and must be enforced by the specifier. Therefore, the service completeness, correctness and clarity are very critical and must be checked prior to the synthesis of the protocol. All these attributes contribute to the avoidance of service specification related design errors.

## 4.4. Service-Oriented Protocol Design

In general, a protocol design is more complex than a service specification because the latter is defined at a higher level of abstraction. Internal peer-to-peer communications are not visible at the upper SAPs; for example, for an error-recovery protocol function, a PDU, imbedded in an SP at a lower SAP, can be used to acknowledge an error-free receipt of previously sent user data without transmitting any SP to the peer service user at the peer

upper SAP. Furthermore, the projection of all interactions onto the upper SAPs must always yield a valid service behavior.

A *protocol design* consists of:

i. The identification of all protocol message types (PDUs) associated with the protocol entity to design.

ii. The mapping between the SPs at the upper SAP and the SPs at the lower SAP.

iii. The mapping of the PDUs to the SPs at both upper and lower SAPs.

iv. A formal specification of the behavior of protocol entities which considers the mapping in ii. and iii. This behavior is governed by the occurrences of events (SPs or internal events) at its interfaces.

Because of its complexity, a protocol design should be achieved in a modular and incremental fashion. First, a simple protocol design can be derived automatically from a service specification. The only function of such initial design is to convey information from one service user to a peer service user transparently (without any local protocol processing). This is often called the synchronization function or control part of a protocol. In another protocol design iteration, for example, the protocol designer could assume an unreliable lower service; therefore, there is a need to add an error-recovery function to the protocol design. This may involve the identification of new PDU types. A variety of protocol functions, such as throughput enhancement, segmentation and flow control functions, may also be incrementally added to a protocol design.

In a synthetic design approach, the fact that the protocol is derived from a protocol design-oriented service specification guarantees that the protocol is service-oriented or faithful to the service. This stresses our viewpoint on the importance of the specification of a complete

and correct service. Furthermore, synthesizing services with the desirable features discussed in Section 4.2, guarantees that the derived protocol will inherit those features.

## 4.5. Discussions and Conclusions

In this chapter, we emphasized the importance of the service concept [LOGR 82, VISS 85] and the relation between services and protocols in the context of layered communication systems. The separation between services and protocols is found to be essential for the development of correct and complete communication protocols. Moreover, we stressed the importance of the design of protocols starting from an unambiguous protocol-oriented service specification. Using a synthetic approach to design, we argue that desirable features of services, such as modularity, clarity and completeness, will be reflected onto similar desirable features in the synthesized protocol.

Another desirable attribute of services is "*testability*". Based on the principle of inheritance of features (from the service to the protocol), testable service specifications would result in testable protocol specifications. Therefore, it would be easier to deal with testability of protocols at the service level. Furthermore, we have identified two features closely related to testability: the modularity and the completeness of a specification. We feel that further research can be done for finding specification techniques and styles that promote desirable features such as modularity and testability of software in general, and communication software in particular.

# CHAPTER 5

# A SERVICE-ORIENTED PROTOCOL SYNTHESIS METHOD

In this chapter, we introduce a new protocol synthesis method with its proof of correctness. Given a complete and unambiguous service specification, the new method automatically synthesizes protocol entity specifications. Both the service and protocol specifications are modeled by finite state machines (FSM). This chapter is organized as follows. In Section 5.1, we introduce the architecture of the communication model we consider for the development of the method, the finite state machine model of both service and protocol specifications, and finally we define some operations on the FSM model. In Section 5.2, we present our synthesis method which uses a given FSM service specification to automatically derive the FSM specifications of the cooperating protocol entities that offer the given service. In Section 5.3, we provide the proofs for the semantic and syntactic correctness of the protocol specifications derived using the synthesis method described in Section 5.2. In Section 5.4, we apply the synthesis method to derive protocol specifications from service specifications examples. In Section 5.5, we compare our method to other synthesis methods that use the service specification as the starting point. In Section 5.6, we briefly present the reversibility feature of the synthesis method. Finally, in Section 5.7, we conclude the chapter by presenting the limitations of our synthesis method, and by outlining the extensions to both the model and the method that will be addressed in the next chapter.

## 5.1. The Model and Basic Definitions

In this section, we first introduce the communication model which defines the context for the development of our synthesis method. Then, we formally define the FSM specification

model for protocols and services, and finally we introduce some operations on the FSM model.

### 5.1.1. The Communication Model

At a high level of abstraction, we view a communication system as a black box which offers a set of specified services to some service users accessing the system through many distributed service access points (SAPs). The service specification consists of: (1) the identification of service primitives (SPs) that are used for the interaction between the service users and the system at the SAPs, and (2) the specification of the possible temporal ordering of SP occurrences at the distributed SAPs. This ordering is referred to as the *global service constraints*. To note here that the *local service constraints* at a particular SAP can be obtained from the global constraints by simply projecting the latter onto each of the SAPs, that is, by hiding from the global constraints the SP occurrences at other SAPs.

At a more refined level of abstraction, the communication services are provided to the service users by a number of cooperating protocol entities exchanging protocol messages that are not observable at the SAPs. In fact, these protocol messages are encapsulated in service primitives occurring at lower SAPs. Moreover, a protocol entity uses the lower service functions to relay protocol messages to another protocol entity.

For the development of the new synthesis method, the communication model we consider is based on the following architectural principles: the set of specified services offered to the service users, the cooperating protocol entities and the underlying services which are abstracted by a reliable first-in-first-out (FIFO) communication medium (Figure 4.1). In this chapter, we will refer to upper SAPs as simply SAPs, and we will refer to lower SAPs

explicitly when necessary. The precise definition of our communication model is given in the next section.

### 5.1.2. FSM Model for Specifying Services and Protocols

The finite state machine specification model is very natural and simple in its description of the sequence of behaviors of protocol entities. In addition, FSM descriptions are familiar to probably the largest audience of communications software engineers. For these reasons, we use the FSM model to develop our synthesis method.

In this section, we will formally define the FSM model for the specification of services and protocols in the context of the layered communication system introduced in Section 5.1.1. We also define some operations on the FSM model, and then we relate service specifications to protocol specifications.

#### 5.1.2.1. Service Specification Model

The service is described by an FSM which specifies the legal sequences of SP occurrences that can be observed at the distributed (upper) SAPs. However, this basic service specification model does not allow the simultaneous occurrence of SPs at the SAPs. Evidently, such limitation is imposed by the sequential nature of the basic FSM model. In the next chapter, we will deal with simultaneous occurrences of SPs, and we refer to them as concurrent SPs.

**Definition 5.1.** A *service specification* S is denoted by a tuple ($S_s$, $\Sigma_s$, $T_s$, $\sigma$), where:

$S_s$ is a non-empty finite set of service states

$\Sigma_s$ is a finite set of service primitives

Ts is a partial transition function between service states (a subset of the cartesian product $Ss \times \Sigma s \times Ss$).

$\sigma \in Ss$ is the initial service state

One or more outgoing transition can emanate from a given service state in S. Those transitions can be labeled by SPs occurring at different SAPs. However, because of the sequential nature of the FSM model, only one transition can be traversed, meaning that after reaching this service state, only one SP, labeling one of the outgoing transitions, can be observed. The choice of a particular SP is either made non-deterministically by the environment (e.g., service user) or deterministically by a protocol entity.

**Definition 5.2.** A *service primitive* $SP \in \Sigma s$ identifies the type of service event and the Service Access Point (SAP) at which it may occur.

For example, A1 means that the service primitive A may occur at SAP 1. Upper case letters are used to denote service primitives.

**Definition 5.3.** For every node representing a service state $s \in Ss$, $OUT(s)$ denotes the set of SAPs associated with the SPs of its outgoing edges.

**Definition 5.4.** A service primitive Si is of type '↑', written Si↑, if the SP is directed upward from the protocol entity servicing SAPi to the service user at SAPi. Similarly, Si is of type '↓', written Si↓, if the SP is directed downward from the service user at SAPi to the protocol entity.

REQUEST and RESPONSE SPs are typical examples of SPs of type '↓', and INDICATION and CONFIRMATION SPs are typical examples of SPs of type '↑'.

Note that the FSM service specification S can also be used to model the actions of the service users distributed across the SAPs. An SP labeling a transition in S corresponds to either an action originating from the service user at SAPi if the observed SP is of type '↓', or an action originating from a protocol entity and received by the service user, if the observed SP is of type '↑'.

**Definition 5.5.** A *trace* is a sequence of events observed at a well-defined set of observation points.

For example, a sequence of SPs obtained by observing the distributed SAPs is a service trace and the observation points are all the interaction points of the communication service.

**Definition 5.6.** The *projection* onto an access point X, written as $\Pi_X$, is a unary function which can be applied to either:

(i) a finite state machine (FSM) $(S, \Sigma, T, \sigma)$ yielding another FSM $(S, \Sigma', T', \sigma)$ in which $\Sigma'$ is a subset of $\Sigma \cup \{\varepsilon\}$ and $T' = T$ with relabeling to $\varepsilon$ of those events in $\Sigma$ not occurring at the SAP onto which the FSM is projected.

(ii) a set of traces yielding another set of traces in which each trace is missing those events which do not contribute to the SAPs onto which the set of traces is projected, but the order of remaining events is as in the original trace.

Note that $\varepsilon$-labeled transitions indicate changes which are not directly attributable to a message reception or transmission at a particular SAP. This is not the same concept as an internal event [ISO 8807].

**Definition 5.7.** A *projected* service (PSi) is the projection of the (FSM) service specification S onto SAPi (PSi = $\Pi_{SAPi}$ S) as described in Definition 5.6. PSi is represented by (Ss, Σ's, T's, σ), where Σ's, is a subset of Σs and T's is a subset of the cartesian product Ss × (Σ's U {ε}) × Ss.

**Note** that each of the PSi has the same number of states as in the service FSM S. Furthermore, for every transition in S between si and sj, there exist n transitions between si1 and sj1 in PS1, si2 and sj2 in PS2, ...., and sin and sjn in PSn (where n is the number of SAPs).

From Definition 5.7, it is clear that the FSM PSi represents the local view of the service as observed at SAPi. In other words, PSi shows the local contribution of the service user at SAPi and the protocol entity servicing SAPi to the service. However, the projected service considered separately would not reveal the nature of the service since the synchronization and interleaving of SPs at different SAPs is not evident.

**Definition 5.8.** $\mathcal{L}$(S) denotes the set of legal *global service traces* .

**Definition 5.9.** $\mathcal{L}$(PSi) denotes the set of legal *local service traces* . Thus, $\Pi_{SAPi}\,\mathcal{L}(S) = \mathcal{L}(PSi)$.

The two definitions shown above will be used later in this chapter to construct the proof of correctness of the synthesis method. These definitions help us to switch from the FSM domain to the trace domain.

### 5.1.2.2. Protocol Specification Model

The protocol specification consists of a tuple of specifications of a number of cooperating protocol entities. The interactions among the protocol entities through the underlying service must provide the services described in the service specification. Protocol entities are also modeled by finite state machines.

**Definition 5.10.** A *protocol entity specification* PE is denoted by a tuple $(S_p, \Sigma_p, T_p, \sigma)$, where:

$S_p$ is a non-empty finite set of protocol states

$\Sigma_p$ is a finite set of protocol events, $\Sigma_p = \Sigma's \cup IPE$, where $\Sigma's$ is a subset of $\Sigma_s$, and IPE is the set of internal protocol events such that for each SP in $\Sigma's$ there exists a unique internal protocol event in IPE

$T_p$ is a partial transition function between protocol states (a subset of the cartesian product $S_p \times \Sigma_p \times S_p$)

$\sigma \in S_p$ is the initial protocol state

**Definition 5.11.** A *protocol specification* (P) consists of several interacting PEs. In our communication model, we suppose that a one-to-one correspondence exists between a PE and a SAP.

**Definition 5.12.** $\mathcal{L}$(P) denotes the set of legal *global protocol traces* observable at the interaction points with the service users (at upper SAPs) and with the communication medium (at lower SAPs).

**Definition 5.13.** $\mathcal{L}$(PEi) denotes the set of legal *local protocol traces* of the protocol entity PEi.

The interaction between an S (modelling service users) and a PEi is based on the rendez-vous concept, meaning that the synchronization will only occur when both S and PEi agree on a given SP. However, the interactions among protocol entities are not assumed to be based on rendez-vous; rather they are synchronized by the underlying (FIFO) communication medium which reliably transfers an event transmitted by one entity to be received by another entity. An internal protocol event e can be either received (?e) or transmitted using the function !(e, X), where X is the set of protocol entities to which e is transmitted.

## 5.2. The Protocol Synthesis Method

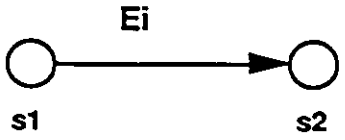
In this section, we first introduce the transition synthesis rules (TSRs), and the atomicity and optimization rules that are used to transform the projected service specifications (PSs) into protocol entity specifications (PEs). Then, we outline the synthesis algorithm.

### 5.2.1. Transition Synthesis Rules

In a PSi, two types of transitions exist: i) SP-labeled transition, and ii) ε-labeled transition which corresponds to an SP-labeled transition in a PSj ($j \neq i$). The existence of these two types of transitions is a direct result of the projection of S onto SAPi.

**Definition 5.14. Transition Synthesis Rules (TSRs).** These rules are applied correspondingly to each of the transitions (ε- or SP-labeled) in PSi. In the following, we require that a protocol event e be uniquely associated with service primitive E (specifically rules a.2, a.4 and b.2, b.4).

*a.Transition labeled by an SP E in PSi:*



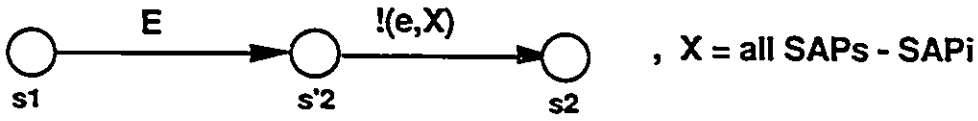

**Case 1:** s2 is not an initial state and OUT(s2) = {SAPi}.

**Rule a.1:** The transition remains unchanged.

The intuition behind this rule is that the flow of control needs not be transferred to another protocol entity (or service user), and therefore a synchronization message should not be transmitted at this point.

**Case 2:** s2 is the initial state (typically in a reset event) and E is of type '↓'.

**Rule a.2:** The transition is transformed into the following two transitions:




Since the SP is originating from the service user, and is taking the service back to its initial state, synchronization messages must be sent to all other protocol entities to synchronize at the initial state in each of the respective protocol entities. This would synchronize the protocol at the same initial global stable state.

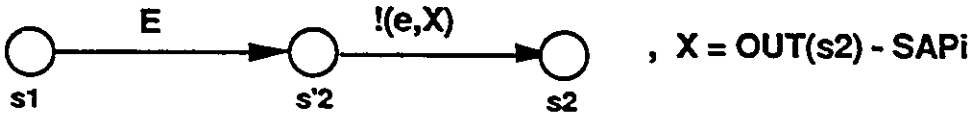
**Case 3:** s2 is the initial state and E is of type '↑'.

**Rule a.3:** The transition remains unchanged.

In this case, the SP is originating from the protocol entity and is taking the service back to its initial state. However this SP is most probably a result of a reset protocol message, and therefore there is no need to transmit any other protocol message.

**Case 4:** s2 is not an initial state, OUT(s2) ≠ {SAPi}, and E is of type '↓'.

**Rule a.4:** The transition is transformed into the following two transitions:




, X = OUT(s2) - SAPi

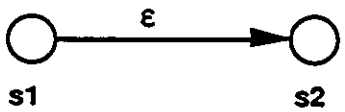
In this case, the SP is originating from the service user at SAPi. Following the occurrence of this SP, other SPs can be observed at other SAPs; and therefore a synchronization message should transfer the flow of control to other corresponding protocol entities.

**Case 5:** s2 is not the initial state, OUT(s2) ≠ {SAPi}, and E is of type '↑'.

**Rule a.5:** The transition remains unchanged.

This case is similar to Case 3.

*b. Transition labeled by ε in PEi:*

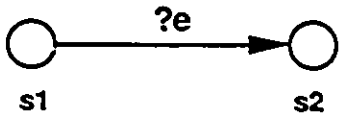



**Case 1:** s2 is not an initial state and OUT(s2) = {SAPi}.

**Rule b.1:** The transition remains unchanged.

**Case 2:** s2 is an initial state and E is of type '↓'

**Rule b.2:** Suppose E is the SP labeling the transition from s1 to s2 in S. In the figure below, ?e corresponds to this label.

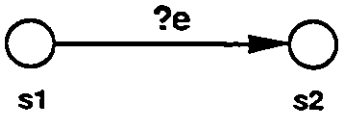



**Case 3:** s2 is an initial state and E is of type '↑'

**Rule b.3:** The transition remains unchanged.

**Case 4:** s2 is not an initial state, OUT(s2) ≠ {SAPi}, and E is of type '↓'

**Rule b.4:** Suppose E is the SP labeling the transition from s1 to s2 in S. In the figure below, ?e corresponds to this label.




**Case 5:** s2 is not an initial state, OUT(s2) ≠ {SAPi}, and E is of type '↑'

**Rule b.5:** The transition remains unchanged.

In Cases 1, 3 and 5, PEi must not expect any message at this point, since according to the service specification, no service action is expected at SAPi.

Note that the rules for synthesizing ε-labeled transitions are compatible with those rules for synthesizing SP-labeled transitions. For example, if for an SP-labeled transition from sj to sk in PSi the appropriate synthesis rule is applied, then a corresponding rule applies to synthesize an ε-labeled transition from sj to sk in PEj (i≠j).

### 5.2.2. Atomicity and Optimization Rules

In our model, we consider the reception of an event (SP at upper or lower SAP) by a protocol entity and the response to such event by the entity as an atomic protocol transaction. The protocol synthesis rules must be able to accomodate this atomicity requirement.

A protocol entity reacts to two types of stimuli as follows.

i) the reception of a PDU (encapsulated in a Service Data Unit - SDU) at a lower SAP causes either an SP of type '↑' to be observed at the upper SAP (Figure 5.1.a), or another PDU (encapsulated in an SDU) to be observed at the lower SAP (Figure 5.1.b).

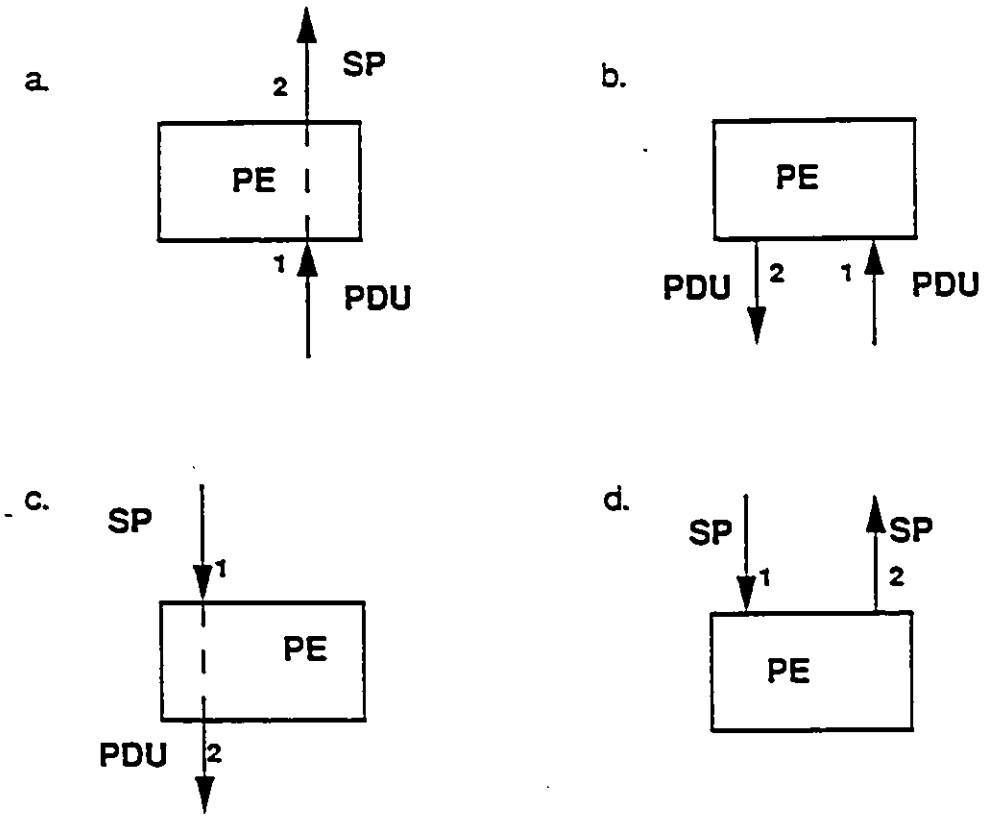
   A typical scenario of the first type of reactions is as follows. A Connect-Request (CR) PDU is received by the peer entity, and is followed by a CONNECT-INDICATION SP directed to the peer service user. However, a scenario of the second type of reactions is the rejection by the peer entity of the CR PDU.

ii) the reception of an SP of type '↓' at the upper SAP causes either a PDU (encapsulated in an SDU) to be transmitted to the peer entity through lower SAP (Figure 5.1.c), or another SP to appear at the same upper SAP (Figure 5.1.d).

   A scenario of the first type of reactions is a CR PDU sent to the peer entity as a result of a CONNECT-REQUEST SP. However, a scenario of the second type of reactions is a CONNECT-REJECT SP after a CONNECT-REQUEST SP occurring at the same SAP. This is typical when the local protocol entity cannot accomodate such request (e.g., limited physical resources).

The choice taken by a protocol entity (in both cases i and ii) is either made non-deterministically by the entity or based on some predicate evaluation of some global or local conditions.

To enforce the atomicity of the action and reaction of the protocol entity to a particular stimuli, we introduce the following atomicity and optimization rules.




(1 and 2 correspond to the order of occurrence of events)

**Figure 5.1.** Reactions of protocol entities to different stimuli.

**Definition 5.15. The atomicity and optimization rules** can be applied to enforce the atomicity of the action and reaction of a protocol entity. The result of applying these rules would be the elimination of some transitions and states, and hence an optimization of the protocol entity. These rules are listed below.

*Rule 1.* A pattern of the type shown in Figure 5.2.a is transformed into a single atomic transition shown in Figure 5.2.b.

*Rule 2.* A pattern of the type shown in Figure 5.2.c is transformed into the atomic pattern shown in Figure 5.2.d. (In fact, Rule 1 is a special case of Rule 2, in which the ε-labeled transition is discarded).

*Rule 3.* A pattern of the type shown in Figure 5.2.e is transformed into the atomic pattern shown in Figure 5.2.f.

Table 5.1 summarizes the transition synthesis rules (after considering the optimization and atomicity rules) and the conditions for their application.

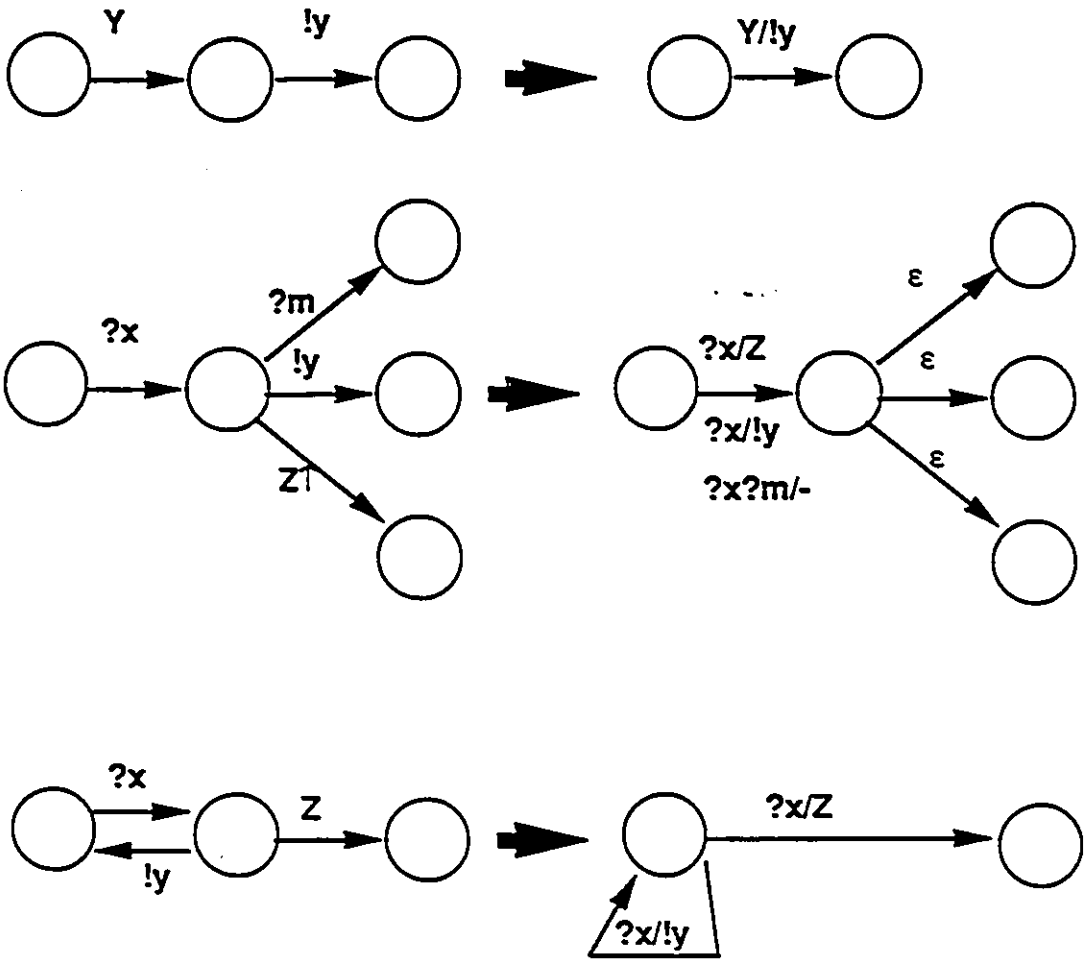



**Figure 5.2.** Patterns and their transformations for atomicity and optimization.

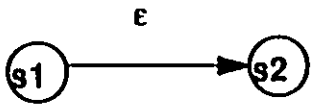
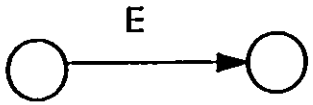
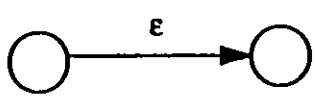
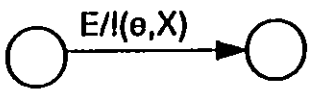
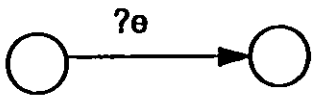
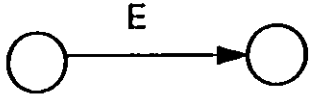
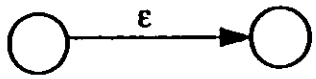
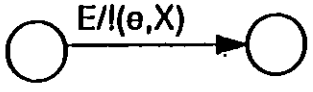
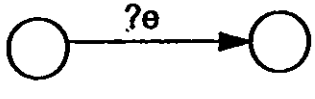
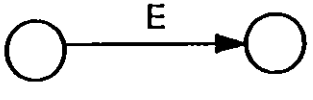
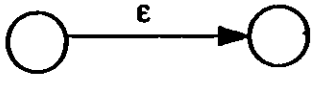
| CONDITIONS \ TRANSITION TYPE | Ei | ε |
|---|---|---|
| |  |  |
| s2 is not an initial state and OUT(s2) = SAPi | a.1:  | b.1:  |
| s2 is an initial state and E is of type ↓ | a.2:  X = all SAPs - SAPi | b.2:  |
| s2 is an initial state and E is of type ↑ | a.3:  | b.3:  |
| s2 is not an initial state OUT(s2) ≠ SAPi and E is of type ↓ | a.4:  X = OUT(s2) - SAPi | b.4:  |
| s2 is not an initial state OUT(s2) ≠ SAPi and E is of type ↑ | a.5:  | b.5:  |

Table 5.1. Summary of the transition synthesis rules and the conditions for their application.

### 5.2.3. The Synthesis Algorithm

Starting from an FSM specification of the service (S), the algorithm outlined below automatically derives the protocol entities that provide the set of services given in S.

**Synthesis Algorithm.** Derivation of protocol specifications from the service specification.

**Input:** FSM-based service specification (S)

**Output:** FSM-based protocol entity specifications (PEs).

**Steps:**

1. Project the service specification S onto each SAP to obtain the PSs.
2. Apply the *transition synthesis rules* to each transition in the PSs to obtain the PEs.
3. Apply the *atomicity and optimization rules* to the PEs.
4. Using the algorithms described in [BARR 79], remove ε-cycles and ε-transitions to obtain the PEs as reduced and equivalent finite state machines.

Note that the TSRs introduced new transitions and protocol states with the following features: i) no new transition is labeled by an SP, and ii) only a unique transition is outgoing from each of the new protocol states, and this transition is labeled by a protocol message which is unique to the SP which invoked the TSR.

A remark worth mentioning here is that our synthesis method starts from a given service specification. This service specification is assumed to be correct, meaningful and complete. The rules and techniques needed to achieve and prove such desirable features of the service definition are beyond the scope of this thesis, but are an important part of our ongoing research. This is an interesting but difficult open problem since it involves verification against non-formalized intentions.

### 5.2.4. Enhancement to the Service Specification

Now, after we have fomally introduced the service and protocol specification model, and the synthesis method, we show one weakness of the service specification model which requires some consideration.

According to the synthesis algorithm, the synthesis of a cycle of SPs (Xi↓ Yi↑) observed contiguously at SAPi (Rule a.1 applies to Xi↓ and Rule a.3 applies to Yi↑ ) would generate the same cycle in PEi. However, for such cycles, some protocol interactions may be invloved, since the projection onto SAPi of another protocol trace Xi !x ?y Yi also yields Xi↓ Yi↑. The following scenario is a typical example of the former protocol trace. A CONNECT-REQUEST SP (Xi↓) is sent to the peer entity via a CR PDU (!x). The peer protocol entity refuses the CR PDU by sending a CREF PDU (!y) to the requesting entity. Upon reception of the CREF PDU, the protocol entity acknowledges the service user by engaging in a CONNECT-REFUSE SP (Yi↑). However, the protocol entity supporting the requestor service user may refuse to handle the request and therefore, the sequence CONNECT-REQUEST CONNECT-REFUSE SPs (Xi↓ Yi↑) is observed at SAPi.

The FSM service model, as described previously, is unable to describe such behaviors and therefore, it requires an extension allowing the model to express such important architectural issues. This enhancement would definitely make the service specification more protocol-oriented.

## 5.3. Proofs of Correctness

In this section, we prove that the synthesized protocol specifications are semantically and syntactically correct. These proofs are needed to establish the correctness of the synthesis method, and to support our claim that no further validation of the synthesized protocol specifications is required. In fact, the main advantage of the synthetic approach to protocol design over the analytic approach is due to the fact that in the former approach, the validation of the protocol is a by-product of the synthesis method.

### 5.3.1. Semantic Correctness

In the following, we prove that the interactions among the derived protocol entities, using a reliable underlying FIFO communication medium, provide the service specified in S. In other words, we prove that the protocol satisfies the liveness properties.

The following definitions introduce two intermediate sets of traces that will be used to construct the proof for the semantic correctness of the derived protocol specifications.

**Definition 5.16.** $\mathcal{L}'$(PEi) is derived from $\mathcal{L}$(PEi) by removing all event transmissions and by replacing every event receptions with its corresponding SP.

For example if A !a ?e D $\in \mathcal{L}$(PEi), then A E D $\in \mathcal{L}'$(PEi).

**Definition 5.17.** $\mathcal{L}\varepsilon$ (PEi) denotes the set of *legal protocol traces* before the elimination of $\varepsilon$-transitions. The equivalence $\mathcal{L}\varepsilon$ (PEi) = $\mathcal{L}$(PEi) is based on the correctness of the $\varepsilon$-cycles and $\varepsilon$-transitions removal algorithms [BARR 79].

The following lemma formalizes the relationships between traces of the protocol and of the service specification.

**Lemma 5.1.** $\Pi_{SAPi}\,\mathcal{L}\varepsilon\,(PEi) = \Pi_{SAPi}\,\mathcal{L}(PEi) = \mathcal{L}(PSi) = \Pi_{SAPi}\,\mathcal{L}(S)$.

**Proof.** The equality $\mathcal{L}(PSi) = \Pi_{SAPi}\,\mathcal{L}(S)$ (Definition 5.9) is a direct result of the definition of the projection operation (Definition 5.6). Both a PSi and a PEi preserve the ordering of SPs at SAPi since the projection of each synthesized transition onto SAPi yields the original service transition in PSi. Furthermore, the TSRs guarantee that no new SP-labeled transitions are generated. In fact, the projection of synthesized transitions labeled by protocol messages yields ε-labeled transitions, the reduction of which returns the original SP-labeled transitions ■

**Lemma 5.2.** $\mathcal{L}'(PEi)$ describes the legal interleavings of SPs at SAPi with respect to some SPs occurring at other SAPs.

**Proof.** According to the TSRs, the reception of an event in PEi corresponds to the occurrence of an SP at SAPj ($j \neq i$). Therefore, if an SP Xi precedes ?y in PEi, then X precedes Y in $\mathcal{L}'(PEi)$ and in $\mathcal{L}(S)$. Consequently, the order of SP occurrences in $\mathcal{L}'(PEi)$ is consistent with their order of occurrence in $\mathcal{L}(S)$■

**Lemma 5.3.** $\Sigma\,\mathcal{L}'(PEi)$ (for all SAPi) specifies the legal interleavings of SPs at all upper SAPs. Also, the SP ordering relations in $\Sigma\,\mathcal{L}'(PEi)$ (for all SAPi) is consistent with the ordering of SPs in $\mathcal{L}(S)$.

**Proof.** We prove this consistency by showing that all traces in $\Sigma$ $\mathcal{L}$'(PEi) (for all SAPi) are consistent with $\mathcal{L}$(S), and also all ordering relations in the traces in $\mathcal{L}$(S) are preserved in $\Sigma$ $\mathcal{L}$'(PEi) (for all SAPi).

The first part of the proof is a direct corollary of Lemma 5.2. To prove the second part, we show that for any pair of contiguous SPs in S, the same pair of SPs exists contiguously at least once in $\mathcal{L}$'(PEi). The contiguous pair of SPs could be of two types: i) AiBi (both SPs appear at the same SAPi): the TSRs guarantee that the same pair of SPs appears contiguously in $\mathcal{L}$'(PEi), or ii) AiBj (either Ai corresponds to ?a in $\mathcal{L}$(PEj) or Bj corresponds to ?b in $\mathcal{L}$(PEi)): the same pair of SPs appears contiguously in $\mathcal{L}$'(PEj)■

**Lemma 5.4.** Only traces from $\mathcal{L}$(S) can be observed at the SAPs, and therefore the service semantics are preserved during the interactions among the synthesized protocol entities, meaning that only the specified services are offered by the communication system.

**Proof.** Lemma 5.3 shows that the order of occurrence of SPs at the SAPs is always correct with respect to the order of occurrence of SPs in S. Therefore, no traces other than the global service traces can be observed at the distributed SAPs ■

**Theorem 5.1.** The synthesized protocol is live.

**Proof.** The proofs for Lemmas 5.3. and 5.4. show that the protocol performs its intended functions, and therefore, satisfies the liveness properties ■

### 5.3.2. Syntactic Correctness

In the following, we show that the derived protocol entity specifications are syntactically correct, by proving that they are deadlock-free, livelock-free and complete (e.g., no unspecified reception error).

**Lemma 5.5.** The synthesized protocol is deadlock-free.

**Proof.** The TSRs are based on a *cause and effect* principle. The occurrence of an X SP at SAPi, where X is the label of a transition from state k to state l in S, will either involve sending a synchronizing protocol message x to all PEs servicing the SAPs in OUT(l) (Rules a.2 and a.4), or no protocol message will be transmitted (Rules a.1, a.3 and a.5). Therefore, the TSRs guarantee that the protocol will progress to a new global stable state when x is received at each of the PEs (Rules b.2 and b.4). From this new global stable protocol state, another SP can be observed at one of the other SAPs if synchronization messages were sent, or another SP will be observed at SAPi if no protocol message was transmitted. Eventually, after successive applications of those rules, the protocol entities will successfuly and synchronously terminate by reaching either the initial global state (Rule a.2) or a final global state in each of the PEs ■

**Lemma 5.6.** The synthesized protocol is livelock-free.

**Proof.** The derived PEs are free from non-productive looping because the TSRs guarantee that all loops in a PEi (for all i) will contain at least one SP. Therefore, any loop in a protocol entity will be contributing to the service (since the semantic correctness had been proved previously) ■

**Lemma 5.7.** The synthesized protocol entities are complete, meaning that no unspecified reception error will occur during the progress of the interactions among the derived protocol entities. (This Lemma requires that our standard assumption on service specification be satisfied, namely that no concurrent executions of SP are allowed).

**Proof.** The TSRs used to synthesize protocol message exchanges guarantee that every protocol message transmitted by an entity to one or more entities is always properly received at those other entities. This is clear from the fact that whenever the conditions for the application of a synthesis rule for a transition from state j to k are satisfied in PEi, and message transmission transitions are synthesized, other conditions are also satisfied for the application of other compatible synthesis rules for a transition from state j to k in the other PEs, which would generate all the necessary reception transitions. To complete this proof, we consider each of the rules that synthesize message transmission transitions:

i. Rule a.2: In this case, we apply Rule b.2 for the synthesis of ε-labeled transitions in each of PEj (for all $j \neq i$) (Figure 5.3.a).

ii. Rule a.4: In this case, we apply Rule b.4 for the synthesis of ε-labeled transitions in each of PEj (for all $j \neq i$ and $j \in$ OUT(s2)) (Figure 5.3.b).

Since in the service specification model no concurrent SPs are allowed at the distributed SAPs, asynchronous protocol entity behaviors are not possible. Therefore, the synthesized reception transitions are sufficient to complete the protocol specification, and therefore are enough to prevent the occurrence of unspecified reception errors ■

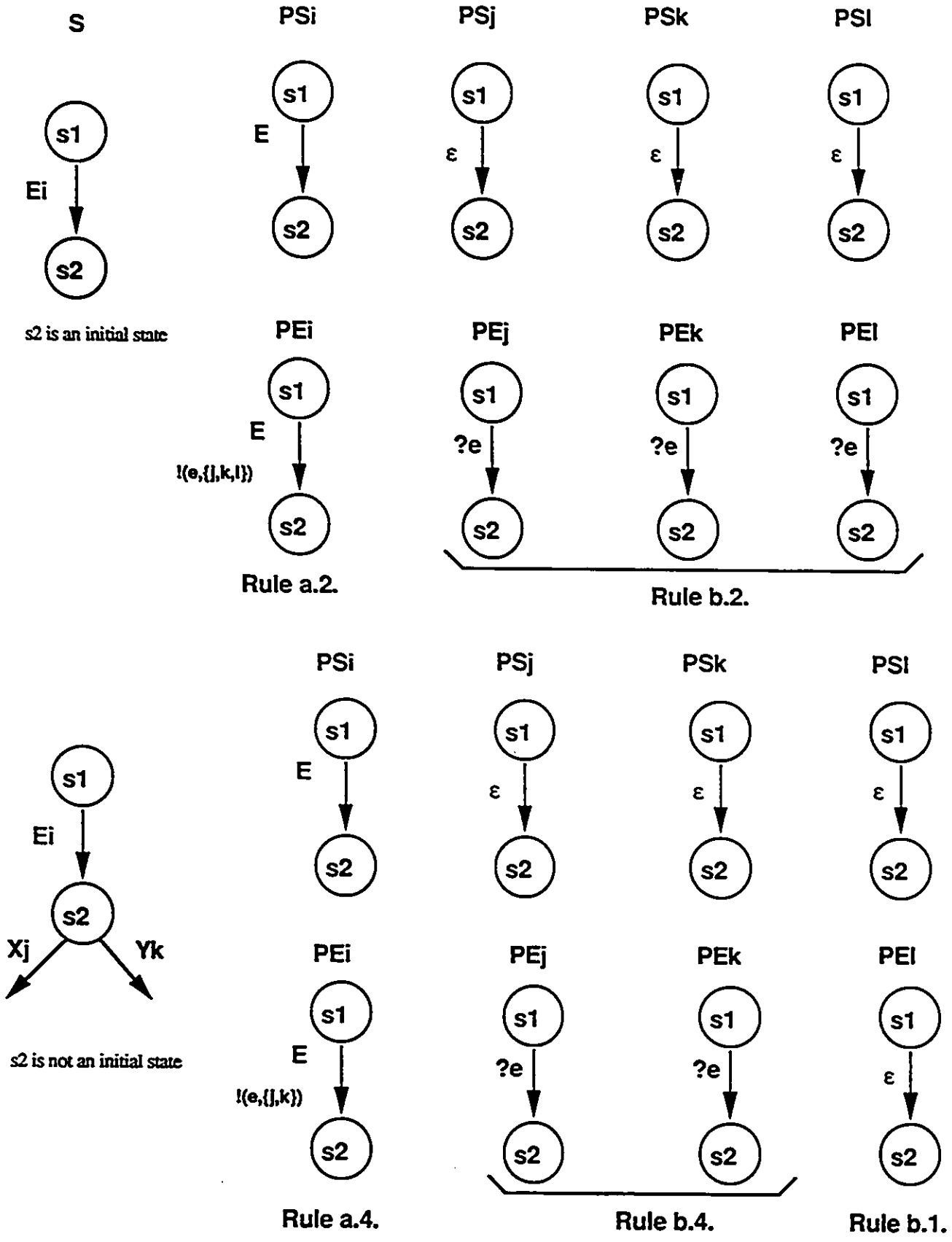



Figure 5.3. Rules for synthesizing message transmission and reception transitions.

**Theorem 5.2.** The synthesized protocol is safe.

**Proof.** The safety properties of the protocol are guaranteed since the protocol has been proved to be deadlock-free (Lemma 5.5), livelock-free (Lemma 5.6) and complete (Lemma 5.7) ■

## 5.4. Examples

In this section, we demonstrate the application of our synthesis method to four service examples. These examples will be used in the next section to compare our method with other service-oriented methods we briefly described in Chapter 2 (Section 2.5.1).

***Example 5.1:***

This example is taken from [CHUL 88a] and is considered here to show the major differences between Chu's method and ours. The service is described as follows. A service user U1 may transmit data at SAP1 via the IN SP, and another service user U2 may retrieve data sent by U1 via the OUT SP at SAP2. Therefore, both IN and OUT SPs are of type '↓'. Furthermore, the executions of IN and OUT are governed by the following restriction which characterizes a two-places buffer: #(OUT) ≤ #(IN) ≤ #(OUT)+2, where #(IN) and #(OUT) denote the number of IN and OUT SPs executed by U1 and U2, respectively. The FSM service specification is given in Figure 5.4. Figure 5.5 shows the derived protocol entity specifications.

**Remarks:**

In this example, a concurrent execution of IN and OUT by U1 and U2, respectively, at service state s2 is not allowed. The FSM model shows only the sequential execution of SPs. However, to accomodate such concurrent behaviors, the FSM model and consequently, the synthesis method needs to be extended.

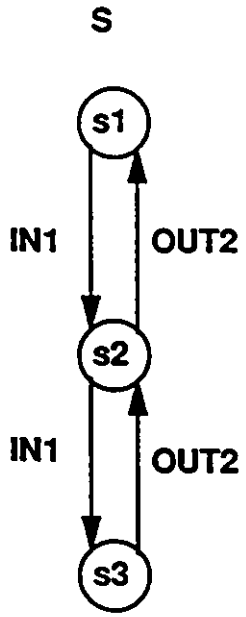



Figure 5.4. Service specification FSM for Example 5.1.

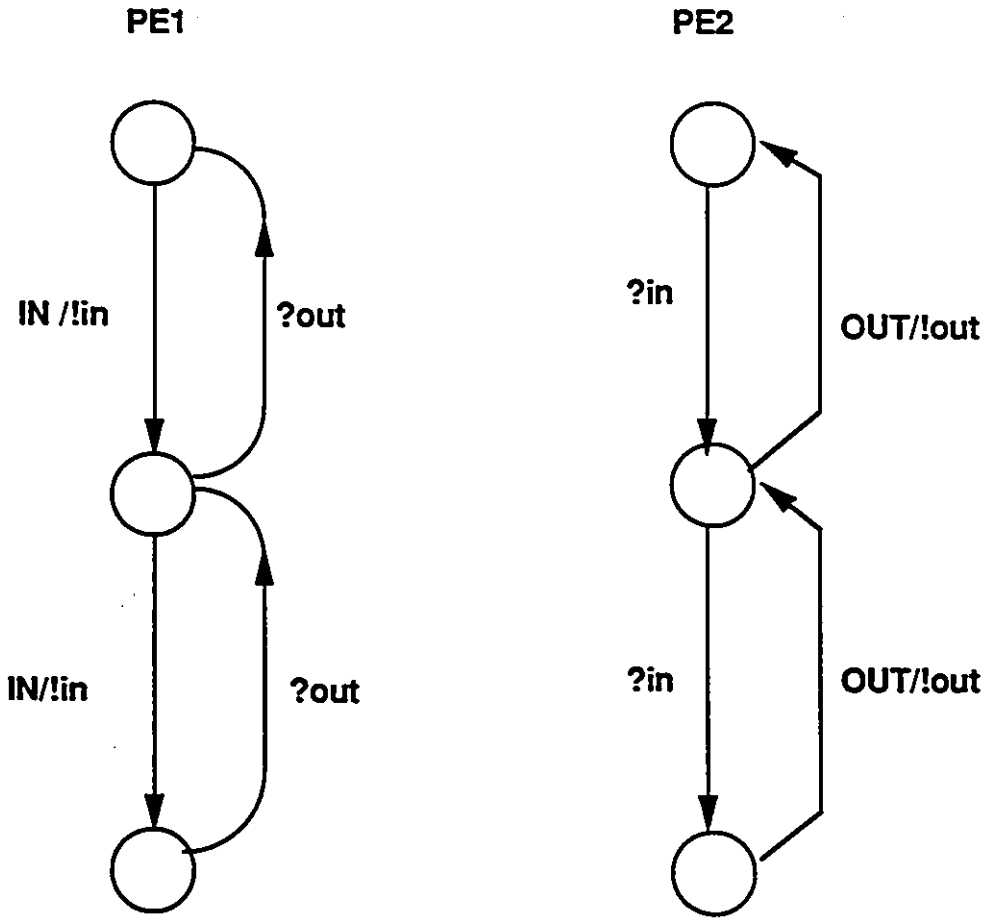



Figure 5.5. Synthesized protocol specifications for the service in Figure 5.4.

*Example 5.2:*

This service involves four service primitives: REQUEST (REQ), RESPONSE (RESP), INDICATION (IND) and CONFIRMATION (CNF) SPs.A service user U1 may issue a request via the REQ SP at SAP1 to another service user U2 at SAP2. U2 will become aware of U1's request when it receives the IND SP at SAP2. U2 will then respond either positively to the request by sending RSP+ SP or negatively by sending RSP- SP at SAP2. U1 will receive U2's response accordingly via the CNF+ or CNF- SP. U1 may also receive a CNF- SP without U2 being involved or aware of U1's REQ SP. Therefore, REQ and RSP SPs are of type '↓', and IND and CNF SPs are of type '↑'. Figure 5.6 shows the FSM service specification.

**Remarks:**

Based on the architectural model described in Section 5.1.1, we have observed that the service specification is missing more information concerning the reception of CNF- SP by U1 directly after REQ SP is executed. The service specification must clarify whether the negative confirmation is a result of a negative response from the peer entity or a rejection by the local entity. Obviously, the rejection by the peer entity involves some protocol messages to be transmitted through the medium; however, no protocol messages are involved when the rejection is a local decision by the protocol entity servicing U1. In general, any pair of contiguous SP occurrences at one SAP leads to such ambiguity and incompleteness at the protocol design level and therefore must be clarified at the service specification level.

In this example, we specify that CNF- SP follows a REQ SP at SAP1 as a result of a rejection (negative response) from the peer entity. Figure 5.7 shows the PEs after applying the TSRs (Step 2), and Figure 5.8 shows the protocol after the removal of ε-cycles and ε-transitions and the application of the atomicity rules.

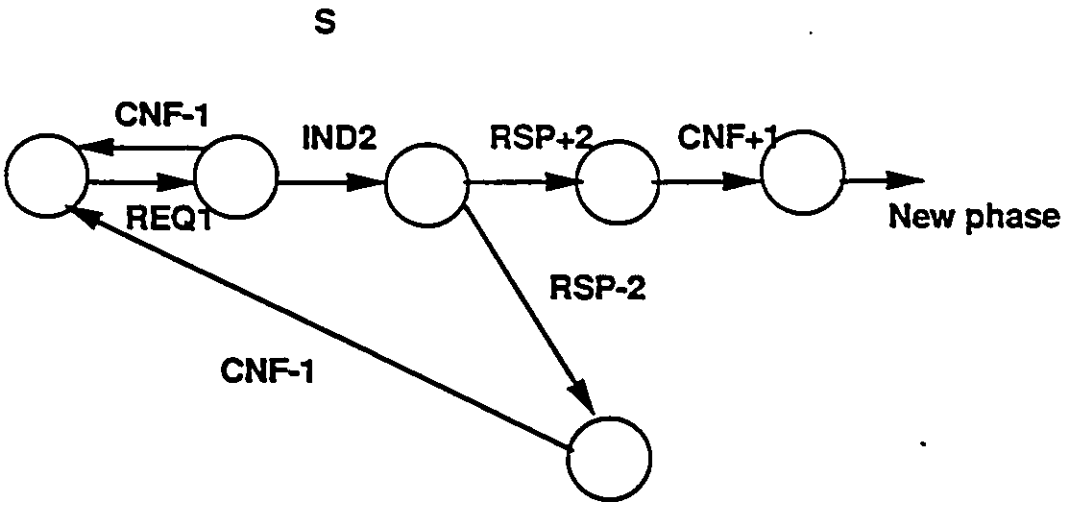



Figure 5.6. Service specification of Example 5.2.

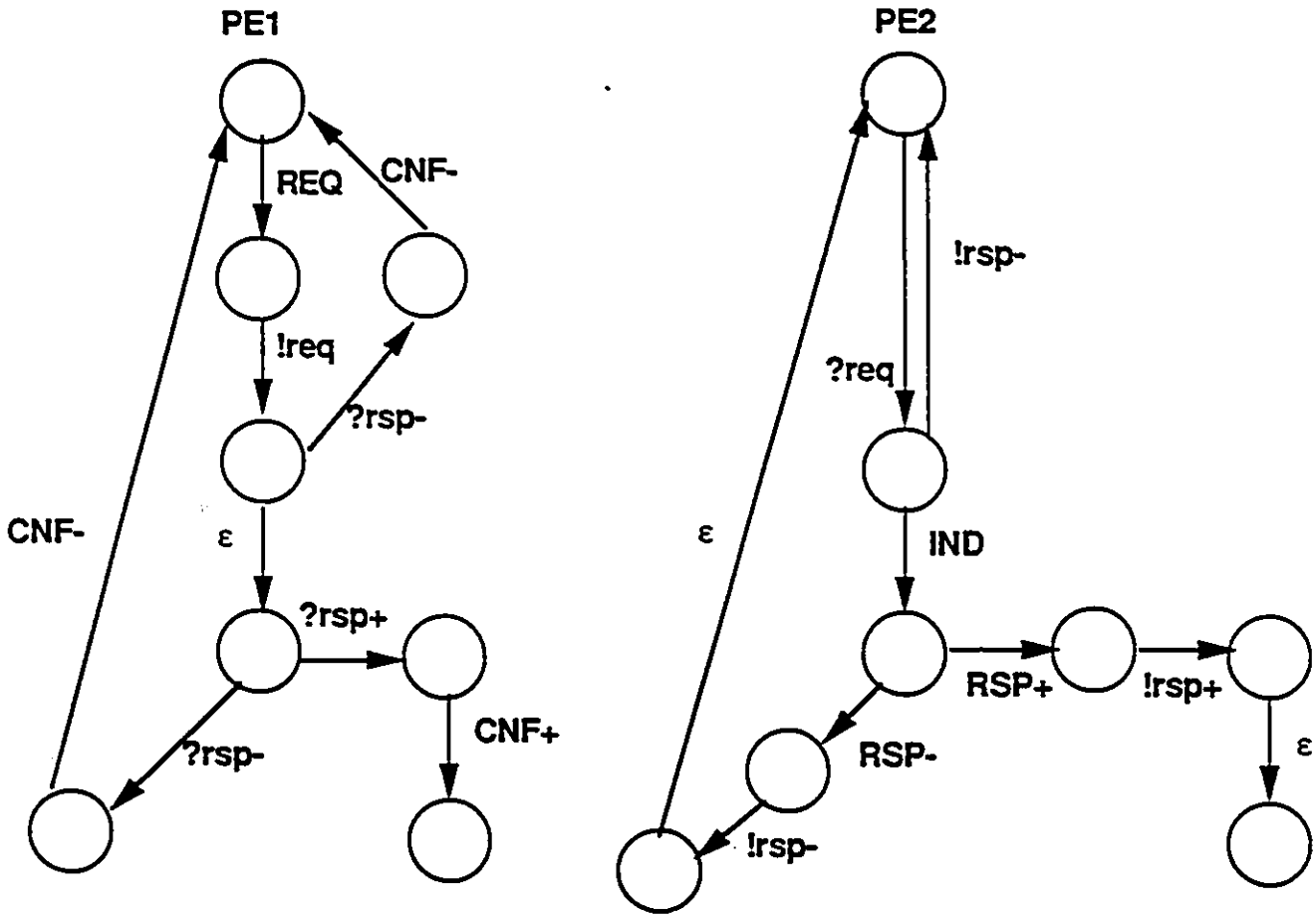



Figure 5.7. Protocol specifications before the removal of ε-cycles and ε-transitions.

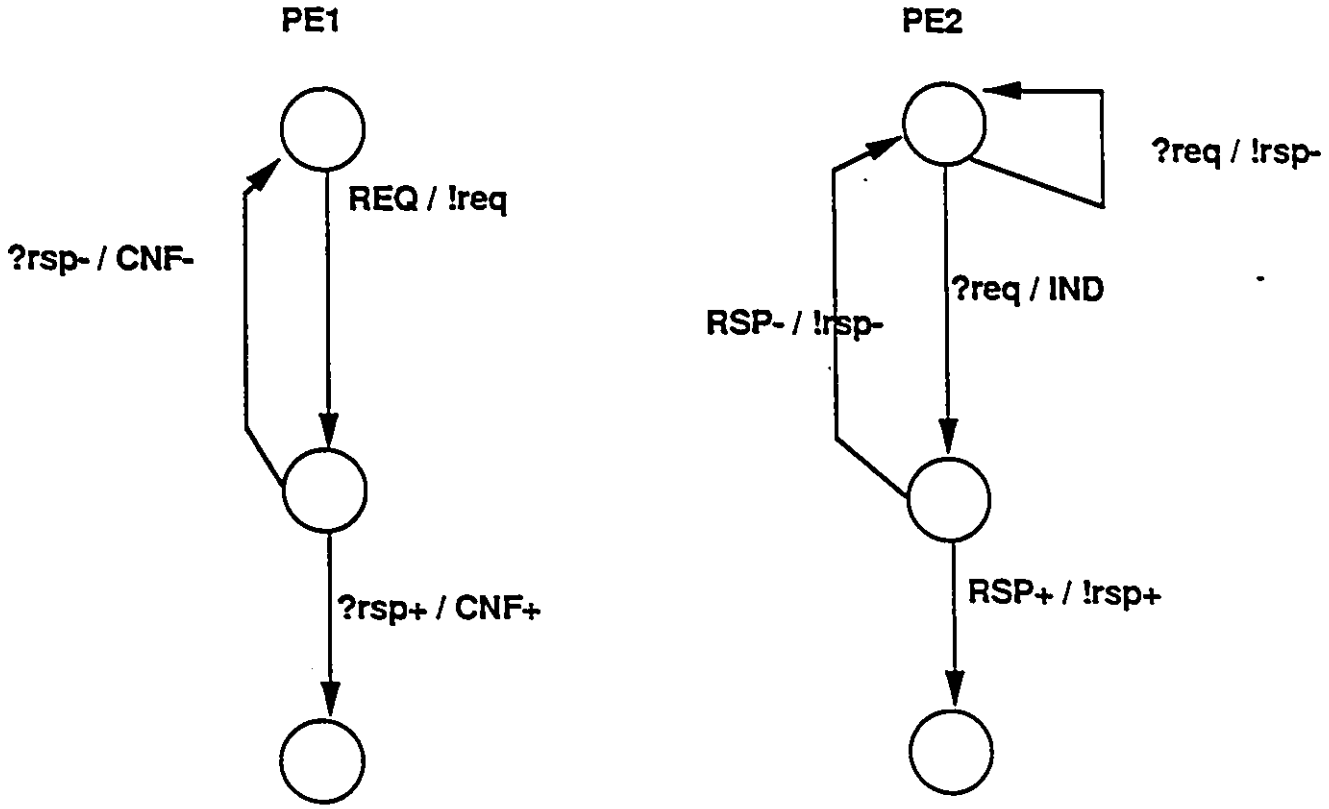



Figure 5.8. Protocol specifications for the service in Figure 5.6.

*Example 5.3:*

In this example, we consider a service specification S(Ss, $\Sigma s$, Ts, $\sigma$), in which: Ss = {s1,s2,s3,s4}, $\Sigma s$ = { A, B, C, D, E}, $\sigma$ = s1, OUT(1) = {sap1}, OUT(2) = {sap2,sap4}, OUT(3) = {sap3}, OUT(4) = {sap2}. Moreover, we assume that all SPs are of type '$\downarrow$'. The corresponding FSM, shown in Figure 5.9, describes the legal sequences of SPs observable at four SAPs. Our goal is to derive four protocol entities specifications (PE) from the given service specification. Figure 5.10 shows the four projected service specifications. Figure 5.11 shows the PEs after applying the TSRs (Step 2), and Figure 5.12 shows the protocol after the removal of $\varepsilon$-cycles and $\varepsilon$-transitions and the application of the atomicity rules.

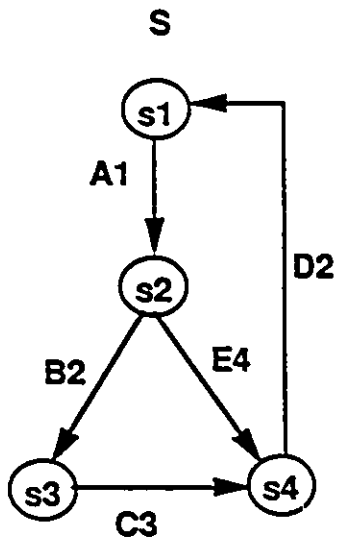



Figure 5.9. The service specification of Example 5.3.

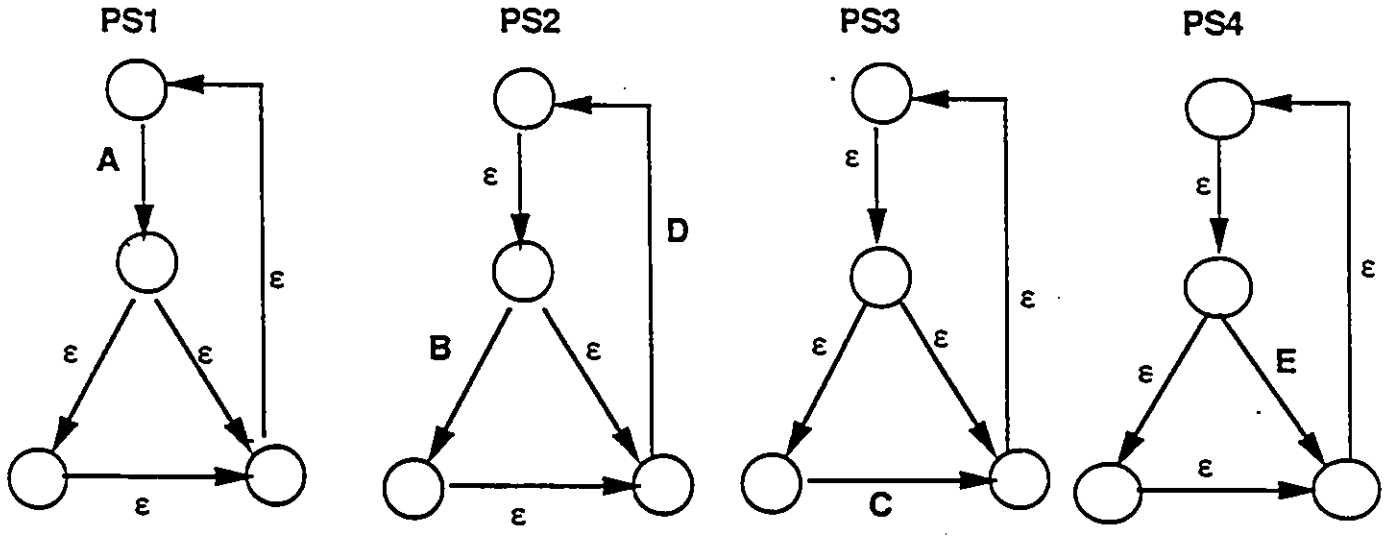



Figure 5.10. Projected service specifications onto four SAPs.

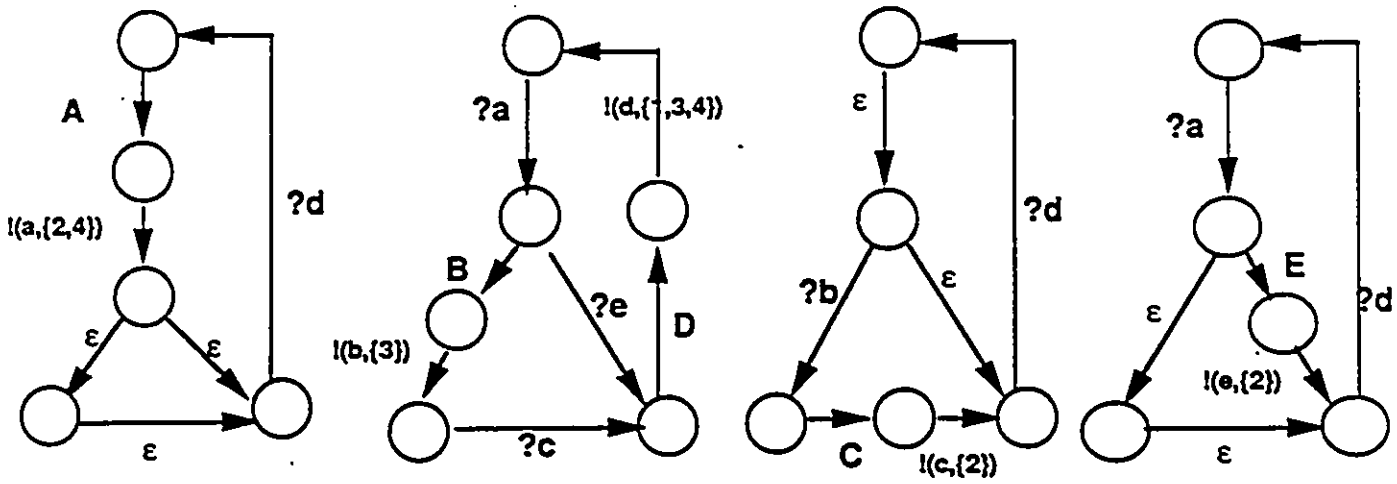



Figure 5.11. Protocol specification before the removal of ε–cycles and ε–transitions.

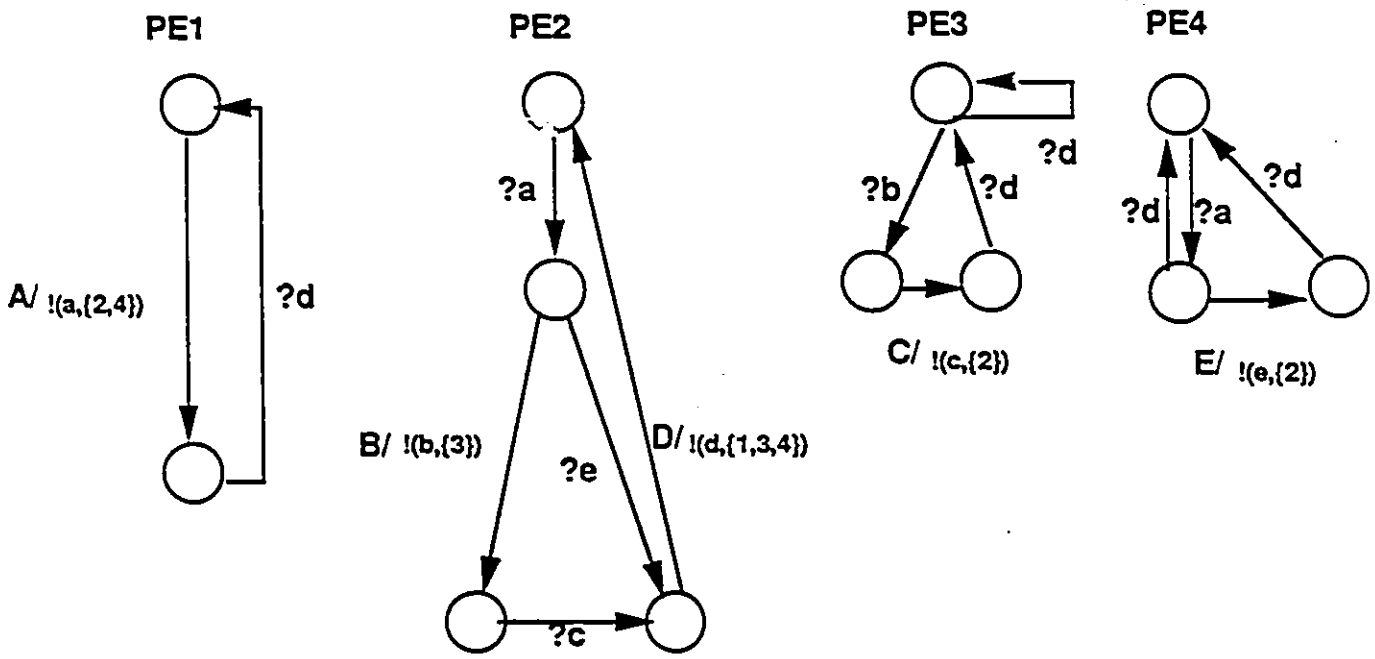



Figure 5.12. Protocol specification derived from the service in Figure 5.9.

*Example 5.4: Application of the Synthesis Method to an OSI service*

In this example, we demonstrate the application of the synthesis method to the association control service of the application layer [ISO 8649]. The service definition is given as an English description of the possible ordering of SPs at the SAPs. This English description is transformed into a FSM description. Figure 5.13 shows the FSM service specification of the association control, and Figure 5.14 shows the EPEs after the application of the synthesis algorithm.

The resulting EPEs are found to be equivalent to the Association Control Protocol Machine (ACPM) [ISO 8650] described in a state transition table in Figure 5.15. Moreover, although we have considered a simple OSI service, the example shows clearly that given a clear and complete service specification we can synthesize the protocol, and thus we avoid the time-consuming analytical development of protocols.

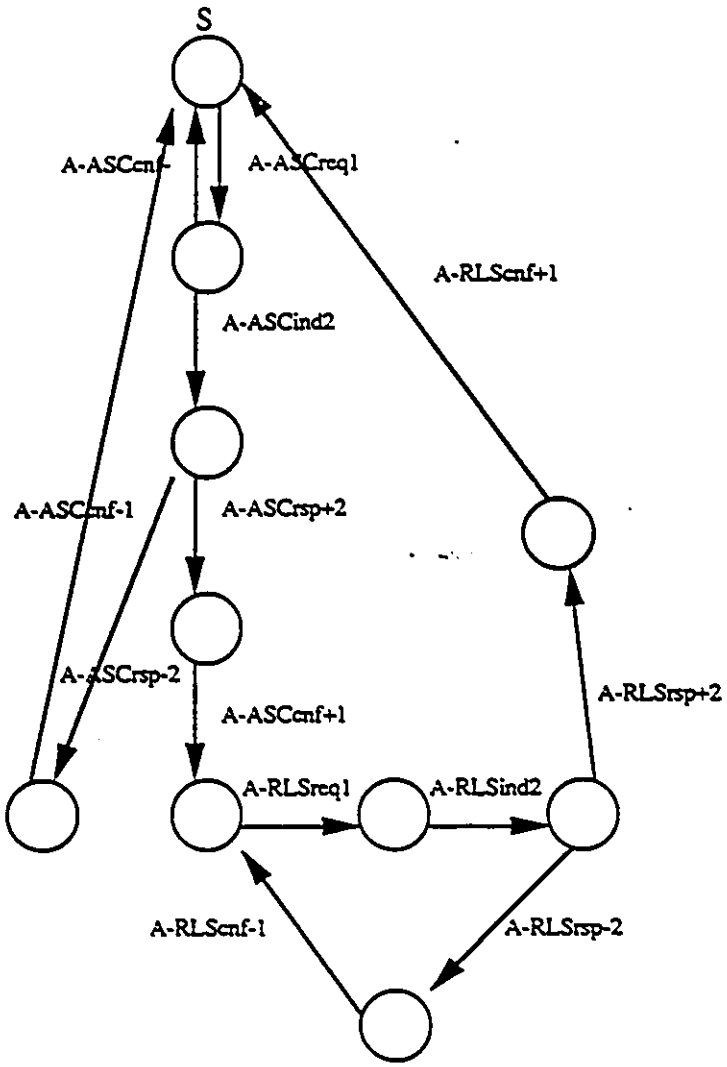



**Figure 5.13.** The Association Control service specification.

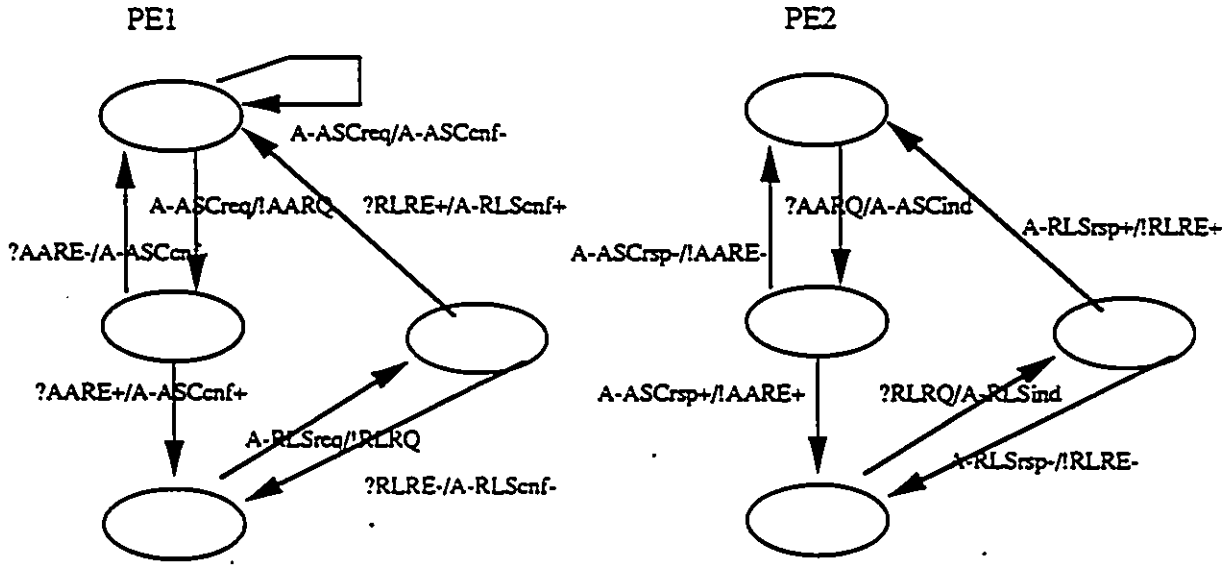



**Figure 5.14.** Association control protocol specifications.

| | STA0 Idle - unassoc. | STA1 Awaiting AARE | STA2 Awaiting A-ASCrsp | STA3 Awaiting RLRE | STA4 Awaiting A-RLSrsp | STA5 Assoc-iated |
|---|---|---|---|---|---|---|
| A-ASCreq | p1: AARQ STA1 | | | | | |
| A-ASCrsp+ | | | AARE+ STA5 | | | |
| A-ASCrsp- | | | AARE- STA0 | | | |
| AARQ | p1: A-ASCind STA2: ¬p1: AARE- STA0 | | | | | |
| AARE+ | | A-ASCcnf+ STA5 | | | | |
| AARE- | | A-ASCcnf- STA0 | | | | |
| P-CONcnf- | | A-ASCcnf- STA0 | | | | |
| A-RLSreq | | | | | | p2: RLRQ STA3 |
| A-RLSrsp+ | | | | | RLRE+ STA0 | |
| A-RLSrsp- | | | | | p3. RLRE- STA5 | |
| RLRQ | | | | | | A-RLSind STA4 |
| RLRE+ | | | | A-RLScnf+ STA0 | | |
| RLRE- | | | | A-RLScnf- STA5 | | |
| A-ABRreq | | ABRT STA0 | ABRT STA0 | ABRT STA0 | ABRT STA0 | ABRT STA0 |
| ABRT | | A-ABRind STA0 | A-ABRind STA0 | A-ABRind STA0 | A-ABRind STA0 | A-ABRind STA0 |
| P-PABind | | A-PABind STA0 | A-PABind STA0 | A-PABind STA0 | A-PABind STA0 | A-PABind STA0 |

Figure 5.15. Transition table for the Association Control Protocol Machine (ACPM).

## 5.5. Comparison with Other Service-Oriented Synthesis Methods

A complete survey and comparison of ten synthesis methods for protocol design was presented in Chapter 2. Among these synthesis methods, only two [CHUL 88a] [BOCH 86] follow the same architectural concepts as ours, and derive the protocol specifications starting from the service specification. In this section, we compare our synthesis method with these two methods.

### 5.5.1. Comparison with Bochmann's Method

Bochmann [BOCH 86], Gotzhein [GOTZ 89] and Khendek [KHEN 89] synthesize the protocol entity specifications starting from a syntactic tree representation of a LOTOS-like service specification. The derived protocol specifications are also described by LOTOS-like behavior expressions. The method is too restrictive since it does not allow a distributed choice; therefore, for the simple LOTOS-like specification equivalent to the FSM specification of Example 5.1: P = IN1 ; Q and Q = OUT2 ; P [] IN1 ; OUT2 ; Q (where P and Q are two LOTOS processes), their synthesis method is unable to derive the protocol specification. Similarly, because of the same restriction, the method cannot be applied to Examples 5.2 ans 5.3. Moreover, no formal proofs of semantic and syntactic correctness of the synthesized protocol are provided in the publications.

### 5.5.2. Comparison with Chu's Method

Chu [CHUL 88a] derives the protocol entity specifications starting from an FSM description of the service. Chu's method is based on the same architectural concepts and the same service and protocol specification modeling formalism as ours.

However, after a careful examination of this method, we have observed the following three potential problems:

*1. Faithfulness and fairness of the protocol specifications:*

To illustrate this problem, we consider Chu's solution to Example 5.1 in Section 5.4. The solution in Figure 5.16 shows that the protocol entity PE1, after first executing IN SP, decides to either allow PE2 to execute OUT SP by sending -b or to execute a consecutive IN SP. This clearly means that PE1 overrides the service specification and decides on the future behavior of the system, therefore the protocol is not faithful to the service.

A faithful and fair interpretation of the service specification is that it is up to the environment (service users) and not to the protocol entities to decide whether to execute two consecutive IN SPs or IN SP followed by OUT SP since both are allowed in the service specification. Our solution follows this interpretation, and therefore it is more faithful to the service.

*2. Potential tempo-blocking (or livelock) :*

Referring to Chu's solution to Example 5.1, the protocol may engage in a tempo-blocking by infinitely or non-productively executing the cycle -b +c in PE1 and +b -c in PE2. This type of cycles in the design requires a lot of testing after the protocol is implemented, and hence its avoidance is desirable at the design stage. Moreover, this potential problem will always exist in all protocol specifications derived from services which include at least one service state whose outgoing edges are labeled by SPs occurring at different SAPs. This is due to the handling of such service nodes by the synthesis algorithm.

*3. Incomplete service specification:*

Chu's solution to Example 5.2 in Section 5.4 is shown in Figure 5.17. For every cycle of SP occurrences at the same SAP, the synthesis algorithm will derive a similar cycle in the corresponding protocol entity specification, and hence it discards other possible behaviors inherited from the architecture of the communication model. It is obvious that the corresponding protocol trace is legal; however another protocol trace which results from the rejection by the peer protocol entity is also legal since when projected on the SAP, it yields the same legal service trace. Without elaborating on such cycles of SPs (or in general, any pair of contiguous SPs in S occurring at the same SAP), we think that the protocol specifications will be incomplete. We believe that extensions on the service specification model should be considered to make it more powerful, expressive and unambiguous.

In summary, our method is more faithful to the service specification and therefore is really a "service-oriented" synthesis method. Furthermore, our method applies to the synthesis of multi-party protocols; however Chu's method needs to be reformulated (a non-trivial task) for such protocols. Finally, Chu's work lacks the formal proofs for the semantic and syntactic correctness of the synthesized protocol entities.

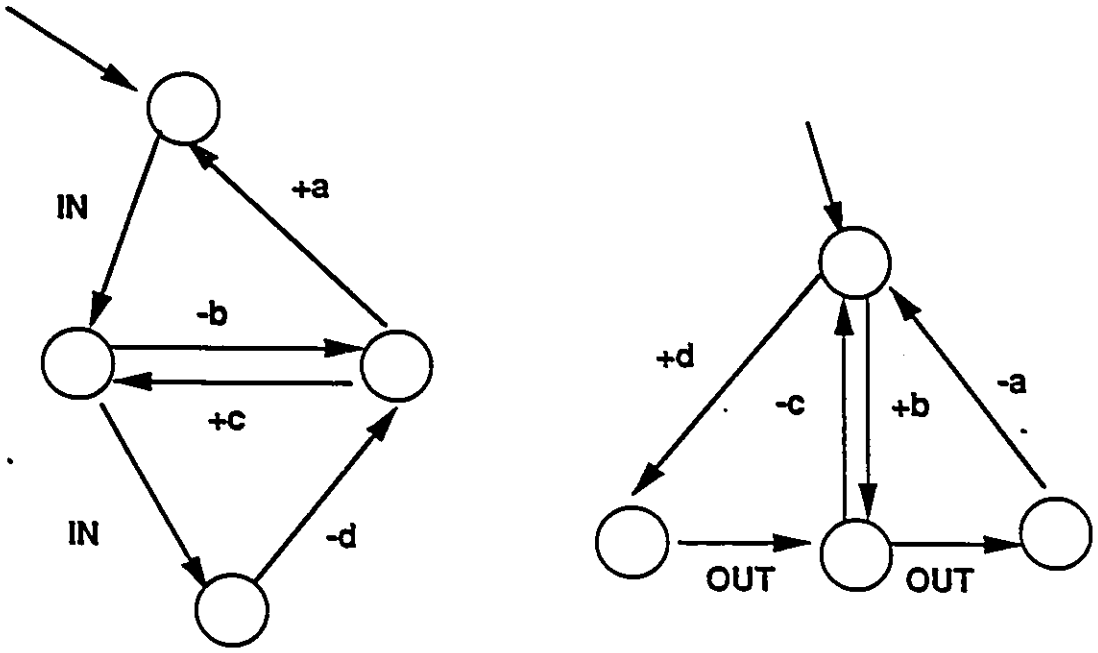



Figure 5.16. Protocol specification for Example 5.1 derived by Chu's method.

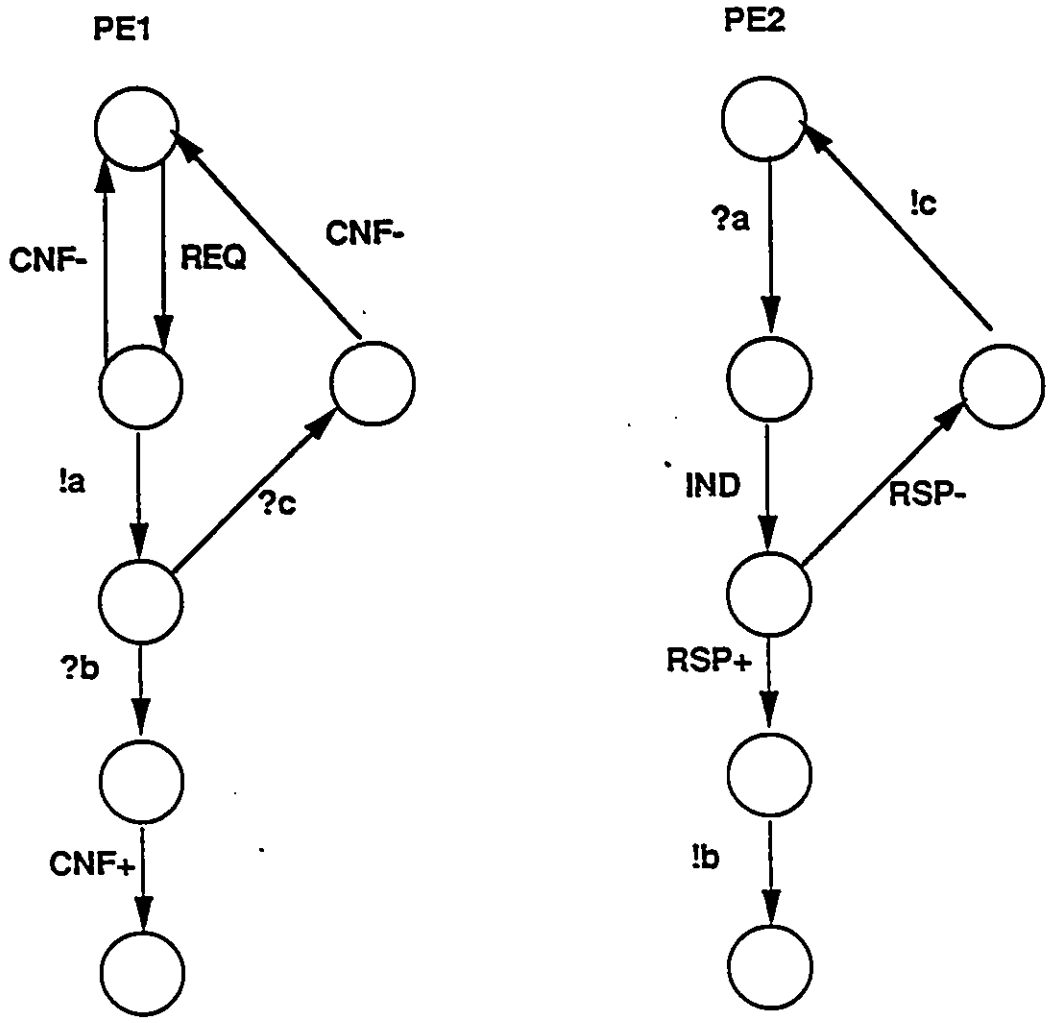



Figure 5.17. Protocol specifications for Example 5.2 derived by Chu's method.

## 5.6. Reversibility of the Synthesis Method

The synthesis method described in this chapter is reversible, meaning that given the protocol specifications derived by the method, and the communication and interaction model, we can uniquely derive the service provided by the interacting protocol entities. This is a desirable feature of synthesis methods since it allows the reconstruction of the service specification.

In the following, we describe a method to prove that the composition (based on the communication model of interaction defined for the method) of the cooperating protocol entities (PEs) yields S.

*Construction of the Global Stable State Graph*

First, we construct the Global Stable State Graph (GSSG) for the cooperating protocol entities. The GSSG is similar to the interaction graph introduced in [LINS 89]. The GSSG is a finite state machine which consists of stable global protocol states (Definition 2.3) and transitions between those states meaning that intermediate states, in which the channels are not empty, are not included in the graph (by definition of stable state), and a state is added to the graph only after the occurrence of an interaction (message transmission and its corresponding reception). The transitions are labeled by service primitives that cause the global interactions.

The application of an FSM minimization procedure [KOHA 79] to GSSG yields the service specification S.

*Terminology*. Let gssg0 = ($\sigma$1, ....., $\sigma$n), where $\sigma$i is the initial protocol state of PEi, be the initial global protocol state. Obviously, gssg0 is also a stable state.

Let $s_{xi}$ be a protocol state in PEi. -(m,$\Delta$) denotes the transmission of message m to the protocol entities servicing the SAPs in $\Delta$.

The algorithm below constructs the GSSG.

step 1. Initial state. Add gssg0 to GSSG.

step 2. For each state gssg = ($s_{x1}$, ....., $s_{xn}$) in GSSG,

2.a. **If** a transmission transition -(m,$\Delta$) between $s_{ki}$ and $s_{li}$ exists in PEi, and receiving transitions +g between states $s_{kj}$ and $s_{lj}$, exist in PEj (for all j in $\Delta$)

**Then** Add a state gssg' = ($s'_{x1}$, ....., $s'_{xn}$) in GSSG provided gssg' is not equal to any previously added state, where $s'_{xe} = s_{xe}$ if e is not equal to i or j (for all j in $\Delta$), otherwise if e = i, $s'_{xi} = s_{li}$, and if e = j (for all j) then $s'_{xj} = s_{lj}$ (for all j).

2.b. Add a transition from gssg to gssg' in GSSG provided the transition is new. The new transition must be labeled by the service primitive that caused the interactions to occur, together with the SAP at which the SP occurs.

step 3. If no transition or state has been added in Step 2, the GSSG is complete, otherwise repeat Step 2.

The GSSG for the protocol synthesized in Example 5.3 (Figure 5.12) is shown in Figure 5.18. The minimization of the resulting GSSG yields S.

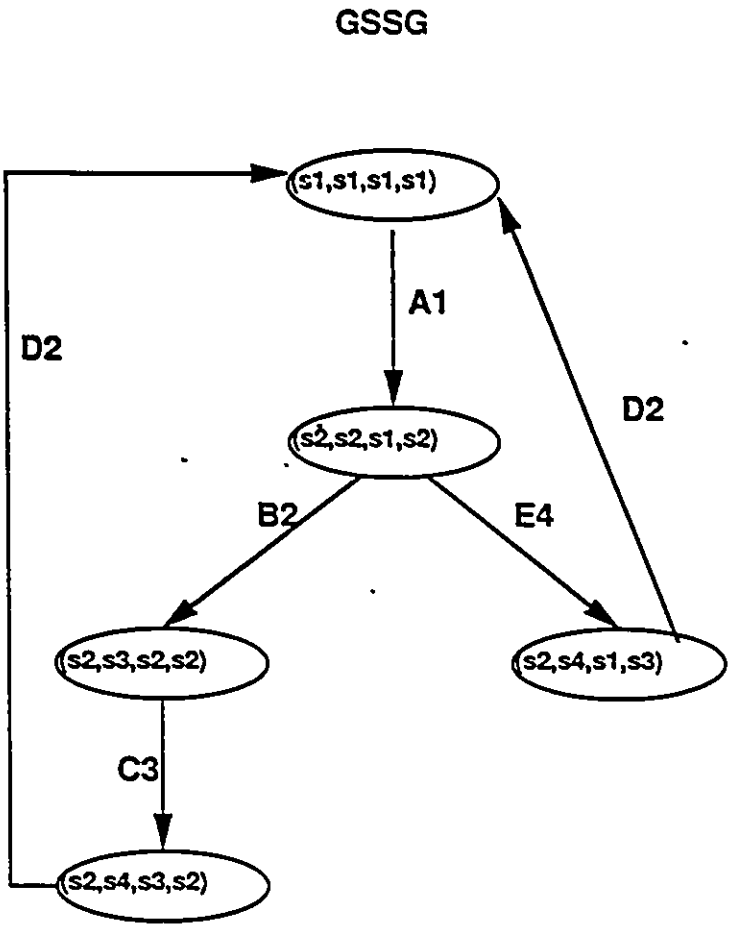



Figure 5.18. GSSG constructed from the protocol in Figure 5.12.

Other features of the method and the model are worth mentioning at this point. First, for a complex service specification FSM which is decomposable into a number of component FSMs, we can apply the synthesis method to each of the components to get partial protocol specifications. These protocol specifications can then be connected together to yield the protocol which provides the whole service. Theoretical work has already been done on the correctness of such connected protocol components [CHOI 86b, CHOW 85].

In addition, some functions such as error-recovery can be added incrementally to the protocol design without affecting the semantic or syntactic correctness of the resulting protocol, as described in Chapter 6.

## 5.7. Discussions and Conclusions

In this chapter, we have introduced a new method for the synthesis of protocol entity specifications starting from the service specification. Both protocol and service specifications are modeled by deterministic finite state machines. The interactions between the protocol entities and the service users are tightly or strongly synchronized. However the interactions among the protocol entities are based on the sending and the eventual reception of protocol messages via the underlying FIFO communication medium.

The derived protocol specifications are proven to be semantically correct, meaning that the interacting protocol entities provide the specified service, and syntactically correct, meaning that no design errors, such as deadlocks or unspecified receptions, can occur during the progress of the interactions among the derived protocol entities.

We have also compared our method to two other service-based synthesis methods introduced in [BOCH 86] and [CHUL 88a]. We have shown that ours is less restrictive, faithfully reflects the service specification, and applies to the synthesis of multi-party protocols as well. We have also applied the synthesis method to an OSI service, namely, the association control service for the common application service of the application layer.

Our experience with the method shows that the FSM model requires some extensions to make the service specification complete, unambiguous and more expressive. We have observed that using the FSM model, we are unable to express concurrent service behaviors at the distributed SAPs. Also, the service specification model requires some enhancements to

allow the expression of some architecturally important information needed for the derivation of complete and unambiguous protocol specifications.

In the next chapter, we propose some extensions to the specification model to enhance its power to express concurrent behaviors of service users, and extensions to the synthesis method to add more protocol-related functions to the protocol, such as error-recovery when an unreliable underlying communication medium is assumed.

# CHAPTER 6

# EXTENSIONS AND CONCLUSIONS

In this chapter, we present two possible extensions to the specification model and the synthesis method we have introduced in Chapter 5, and then we conclude the thesis. In Section 6.1, we propose the two extensions. The first involves an extension to the service specification model to allow the specification and handling of concurrent behaviors at the distributed Service Access Points (SAPs). This extension requires some additions to the synthesis rules and the algorithm developed in the previous chapter. The second involves an extension to the synthesis method to include error-recovery mechanisms in the derived protocol specifications. This extension expands the protocol specifications resulting from the synthesis algorithm outlined in Chapter 5, and is based on the assumption that the underlying communication medium is unreliable. Finally, in Section 6.2, we conclude the thesis and we present some suggestions for further research.

## 6.1. Extensions

In this section, we propose two possible extensions to the synthesis method we presented in the previous chapter. The first extension involves introducing the concept of **controlled concurrency** as a model for service specification. The second extension involves the inclusion of error-recovery mechanisms in protocols synthesized from basic service specification described in the previous chapter.. For the sake of simplicity of the

presentation of these two extensions, and without loss of generality, we will present the two extensions separately, i.e. the extensions are independent.

### 6.1.1. Handling Concurrent Behaviors

One of the drawbacks of the finite state machine (FSM) specification model for communication software (protocols) lies in the inadequacy of the model to express complex control flow aspects of protocols such as the specification and handling of concurrent behaviors, synchronization and timing issues. FSMs are more suitable to model sequential communication behaviors expressed as a set of linear sequences of transitions. Therefore, the FSM service and protocol specification model defined in the previous chapter is not suitable to express the asynchronous and concurrent nature of user activities distributed at the different SAPs of the communication system.

In this section, we first extend the basic FSM model to allow the specification of concurrent user activities at the distributed SAPs. Then, we study the implications of this extension on the synthesis algorithm.

For the sake of simplicity of the presentation of this extension, and without loss of generality, we will consider the case of only two concurrent SPs.

#### 6.1.1.1. Extension to the Specification Model

***Extended service and protocol specification model***

In the extended service specification, concurrent service user behaviors are possible when a service state has two or more outgoing transitions labeled by service primitives occurring at different SAPs. A concurrent activity at state $n$ is only possible when $\#(OUT(n)) > 1$ (the

number of distinct SAPs in the *OUT* set of state *n* is greater than 1). We will refer to such service state as a *mixed* state.

In Figure 6.1.a, state *n* is not a mixed state since #(*OUT(n)*) = 1. However, in Figure 6.1.b, *n* is a mixed state since #(*OUT(n)*) = 3.

Since according to the service specification, the occurrence of two simultaneous service primitives is allowed, a collision of service primitives is then possible. The collision of primitives is defined below:

**Definition 6.1.** A collision of service primitives occurs when two or more service users simultaneously issue service requests to each other.

**Definition 6.2.** A Reset service primitive R is a special primitive of type '↑' used by the protocol entities to acknowledge service collisions.

Controlled Concurrency:

We control the degree of concurrency in our service specification model by characterizing the type of each mixed state and by specifying the approach to handling concurrent SPs at these mixed states once a collision of service primitives has been detected in the protocol entities.

*i*. *Mixed State of type '||':* Only one SP may occur at a time. In this case, only synchronous behaviors are allowed, and therefore a collision will never occur.

*ii.* ***Mixed State of type '$\|_{r0}$'***: Concurrent SPs are allowed. However, when an SP collision is detected by the protocol entities, the service specification machine is reset to its initial state.

*iii.* ***Mixed State of type '$\|_{rL}$'***: Concurrent SPs are allowed. However, when an SP collision is detected in the protocol entities, the service specification machine is reset to the last mixed service state where the collision has occurred.

*iv.* ***Mixed State of type '$\|_{i}$''***: Concurrent SPs are allowed. However, when an SP collision is detected in the protocol entities, it will be resolved by giving the service user **i** a higher priority (and by acknowledging the low priority users).

Figure 6.2 shows the four types of mixed states, and the use of Reset SPs by protocol entities to handle service collisions.

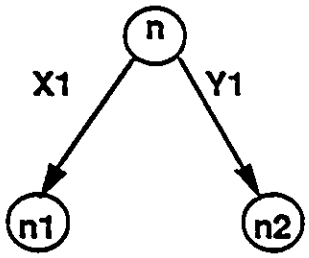



Figure 6.1.a. Non-mixed state n.

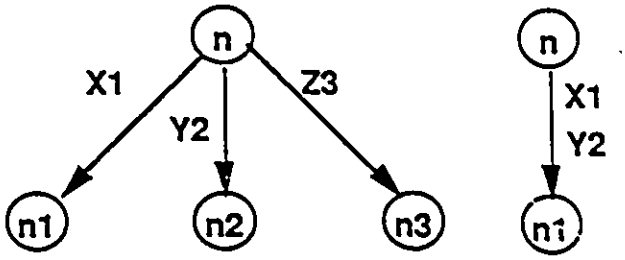
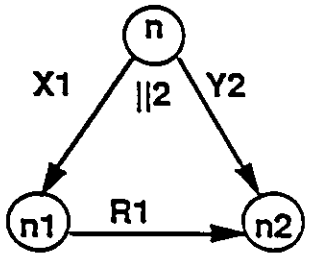
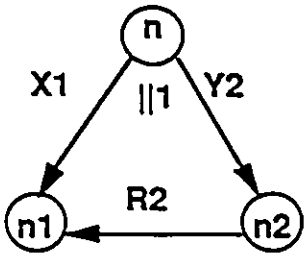
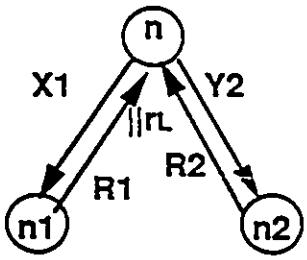



Figure 6.1.b. Mixed state n.



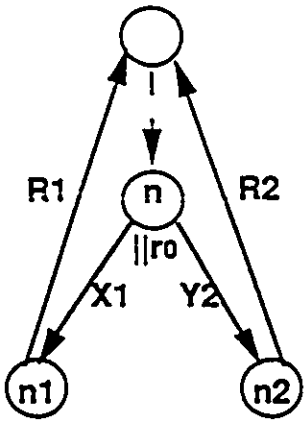




Figure 6.2. Four types of mixed states and handling of service collisions.

The set of outgoing SP-labeled transitions at a mixed state **n** is partitioned: all transitions labeled by SPs occurring at the same SAP are included in the same partition. A concurrent behavior at a mixed state of type other than '||;' implies the concurrent execution of at least two SP-labeled transitions, each belongs to a different partition. Therefore, we do not consider the case when SPs occur at the same SAP as a concurrent activity, since the service user can only engage in one activity at a time.

Furthermore, a service specification machine is of type **SM;**, **SMr0**, **SMrL** or **SMi** if all mixed states are of type '||;', '||r0', '||rL' or '||i', respectively.

The extended service specification ES models the extended concurrent behaviors of service users, and is formally defined as follows:

**Definition 6.3.** An extended service specification ES is denoted by a tuple ($S_S$, $\Sigma_S$, $ET_S$, $\sigma$), where:

$S_S$ is a non-empty finite set of service states

$\Sigma_S$ is a finite set of service primitives

$ET_S$ is a partial transition function between service states (a subset of the cartesian product $S_S \times (\Sigma_S \cup R) \times S_S$)

$\sigma \in S_S$ is the initial service state

Similarly, the extended protocol specification EP consists of the protocol entity specifications EPEs. A protocol entity is formally defined as follows:

**Definition 6.4.** A extended protocol entity specification EPE is denoted by the tuple ($S_P$, $\Sigma_P$, $ET_P$, $\sigma$), where:

$S_P$ is a non-empty finite set of protocol states

$\Sigma_P$ is a finite set of protocol events, $\Sigma_P = \Sigma'_S \cup IPE$, where $\Sigma'_S$ is a subset of $\Sigma_S$, and IPE is the set of internal protocol events (including reset events)

$ET_P$ is a partial transition function between service states (a subset of the cartesian product $S_P \times (\Sigma_P \cup R) \times S_P$)

$\sigma \in S_P$ is the initial protocol state

**Definition 6.5.** A fair repetition of SP occurrences is denoted by $(\mathcal{X})^F$, where $\mathcal{X}$ is a list of SPs. For example, $(X1 \mid Y2)^F$ expresses the concurrent execution of the sequences X and Y at SAP1 and SAP2, respectively. This concurrent execution occurs at least once and at most a fair and finite number of times.

**Definition 6.6.** $\mathcal{L}(ES)$ denotes the set of legal *global service traces* of the extended service specification. This set contains the traces of the basic service model ($\mathcal{L}(S)$) in addition to other traces that can be derived from $\mathcal{L}(S)$. These new traces are derived as follows.

Let $\mathcal{Y} = Y'\ Y''\ \ldots$, $\mathcal{Z} = Z'i\ Z''\ \ldots$, and $\mathcal{W} = W'j\ W''\ \ldots$, be three sequences of SP occurrences. Let $\mathcal{YZ}$ and $\mathcal{YW}$ be two traces in $\mathcal{L}(S)$. The existence of these two traces in $\mathcal{L}(S)$ implies that there exists a mixed state *n* whose outgoing transitions are labeled by Z'i and W'j. Depending on the type of mixed state *n*, $\mathcal{L}(ES)$ also contains one of the following legal service traces ($\mathcal{V}$):

a. **n** of type ||i: $\mathcal{V} = Y'Y''\ (Z'i \mid W'j)\ Rj\ Z''\ \ldots$

b. **n** of type ||j: $\mathcal{V} = Y'Y''\ (Z'i \mid W'j)\ Ri\ W''\ \ldots$

c. **n** of type ||rL: $\mathcal{V} = Y'Y''\ ((Z'i\ Ri) \mid (W'j\ Rj))^F\ Z'i\ Z''\ \ldots$

   or $\mathcal{V} = Y'Y''\ ((Z'i\ Ri) \mid (W'j\ Rj))^F\ W'jW''\ \ldots$

d. **n** of type ||r0: $\nu$ = Y'Y" ((Z'i Ri) | (W'j Rj))

e. **n** of type ||;: $\nu = \varepsilon$.

Note that when all mixed states are of type ||; (the service machine is of type SM;), no concurrent SPs are allowed, and therefore $\mathcal{L}(S) = \mathcal{L}(ES)$.

#### 6.1.1.2. Extension to the Synthesis Method

The extension to the method is based on the synthesis of new states and transitions which are required for the handling of SP collisions at mixed service states of type '$||_{r0}$', '$||_{rL}$', or '$||_{i}$'. In fact, these new protocol states and transitions are added to the EPEs derived by the basic synthesis method in an incremental fashion. This incremental construction builds on the top of syntactically and semantically correct protocol specifications (proven in Chapter 5), which consequently simplifies the proof of correctness of the extended protocol specifications.

In the following, we first consider the simpler case of handling concurrent SPs in a two-users (or SAPs) communication service, then we generalize to multi-user services. In this generalization, we show the potential *divergence* problems caused by SP collisions and for which we propose some solutions.

#### A) Handling Concurrent SPs in Two-Users Communication Services

In a two-users communication service, a mixed state *n* in a service specification FSM involves only two SAPs. Suppose that the outgoing transitions of a mixed state *n* are labeled by X1 and Y2. After applying the synthesis method, we obtain two protocol entities which cooperate to provide the specified service. In general, as shown in Figure

6.3, we could have two partitions of outgoing transitions: one labeled by SPs at SAP1 consisting of {X1, X'1, X"1, ..} and the other labeled by SPs at SAP2 consisting of {Y2, Y'2, Y"2, ... }.

**Definition 6.7.** *T(n)* denotes the set of labels of the transitions outgoing at state *n*.

**Definition 6.8.** The application of the extended synthesis algorithm derives the extended protocol (EP) which consists of the extended protocol entities (EPEs).

**Definition 6.9.** $\mathcal{L}$(EPEi) denotes the set of legal traces in EPEi. If the service machine is of type SM;, $\mathcal{L}$(EPEi) = $\mathcal{L}$(PEi) for all protocol entities.

**Definition 6.10.** A protocol state is called a mixed state, if after hiding the SP labels from its outgoing transitions, one of the transitions is labeled by a message reception, the other by a message transmission.

**Definition 6.11.** For every mixed service state *n* in ES, there exist two mixed protocol states, one in each of the EPEs. These states are easily and uniquely identified.

After applying the TSRs and the synthesis algorithm defined in Chapter 5, we obtain the patterns shown in Figure 6.4.

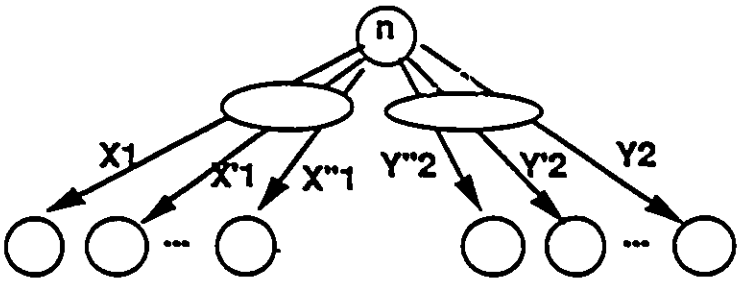



**Figure 6.3.** Partitioning the transitions for a mixed service state.

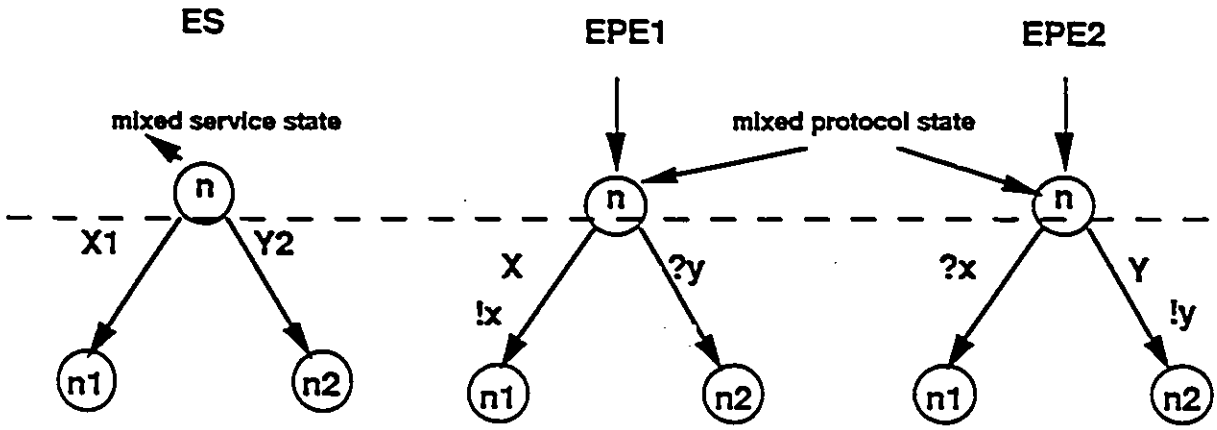



**Figure 6.4.** Mixed service and protocol states.

Since the SP-labeled transitions in the protocol entities are tightly coupled with the service users at the corresponding SAPs, we can see that for every mixed state in the service specification, a collision of events may occur: a send event in one entity collides with another event sent from the peer entity.

To prevent the occurrence of an unspecified reception design error, each protocol entity must be ready to receive the colliding protocol message sent from the other entity at the state reached after sending the colliding message, and at every other state along the path of event transmissions and/or SP occurrences. Figure 6.5 shows the collision of events and the receptions transitions added to avoid unspecified receptions design errors.

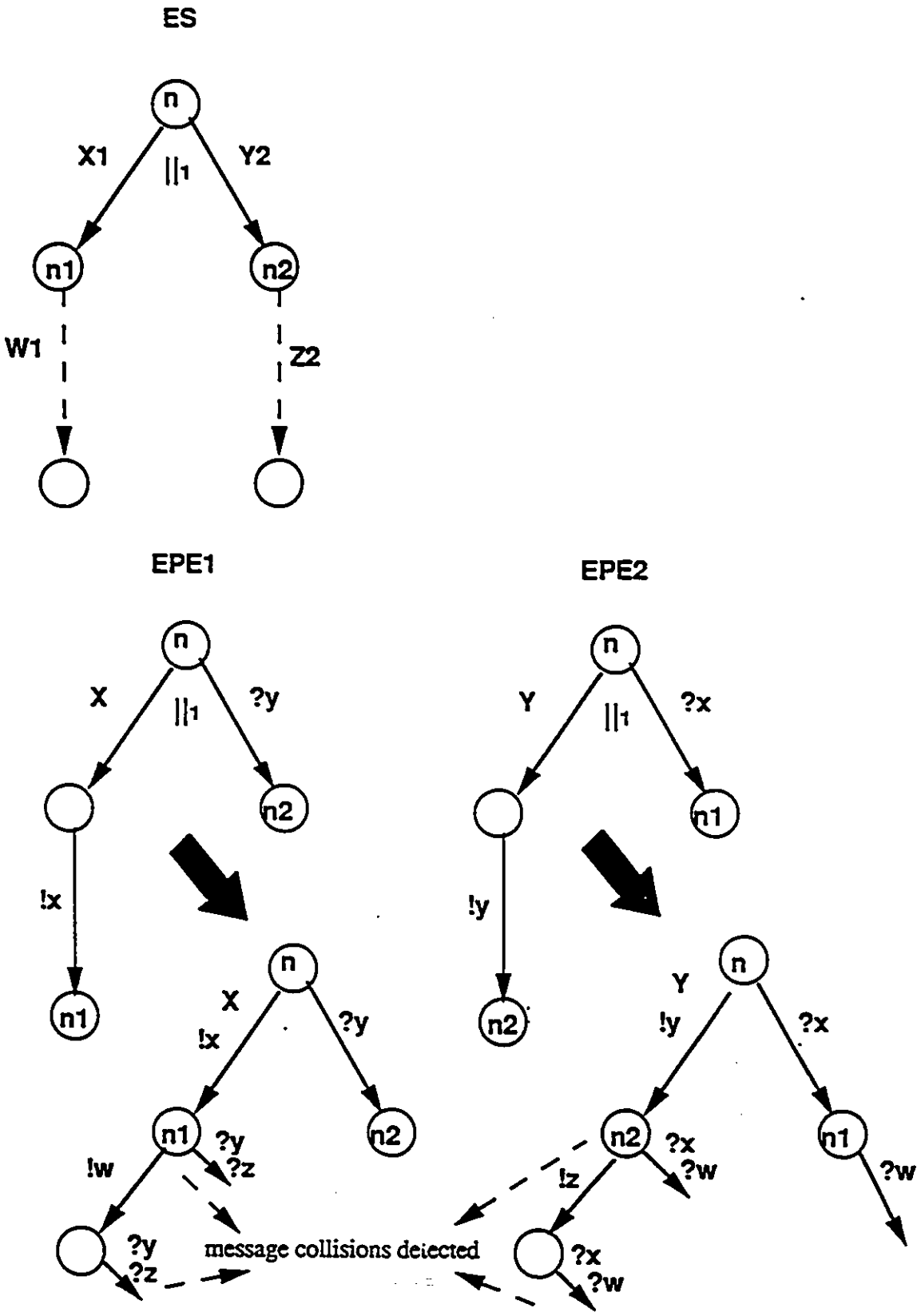



Figure 6.5. Collision detection and avoidance of unspecified receptions.

## B) Handling Concurrent SPs in Multi-User Communication Services

The synthesis rules applied to multi-user services are much complex since several protocol entities may be involved. To develop these rules, we first state the *protocol divergence problem,* and then we examine the conditions under which the protocol will potentially diverge. Finally, we propose mechanisms to avoid such divergence.

***Problems of Asynchrony and Potential Divergence***

When many communicating protocol entities are involved, if a collision occurs between events originating from two entities, under some conditions, it becomes more complex to control the asynchronous execution of other entities not involved in the collision. Therefore, a mechanism is required to rollback the asynchronous execution of other entities when necessary.

In the example shown in Figure 6.6, the concurrent execution of X at SAP1 and Y at SAP2, where $T(n) = \{X,Y\}$ and $n$ is of type $||_2$, will potentially cause the divergence of the protocol. If EPE1 sends x to EPE3 before it becomes aware of the occurrence of a collision (by receiving the colliding message y from EPE2), EPE3 will proceed asynchronously. EPE1 will synchronize with EPE2 since the mixed protocol state of the latter is assigned a higher priority. The divergence of EPE3 will allow the occurrence of some service behaviors that are not included in $\mathcal{L}$(ES) (according to Definition 6.6).

To avoid such divergence, we need to introduce a mechanism by which the divergence of EPE3 is avoided when a possible collision causing such divergence is recognized.

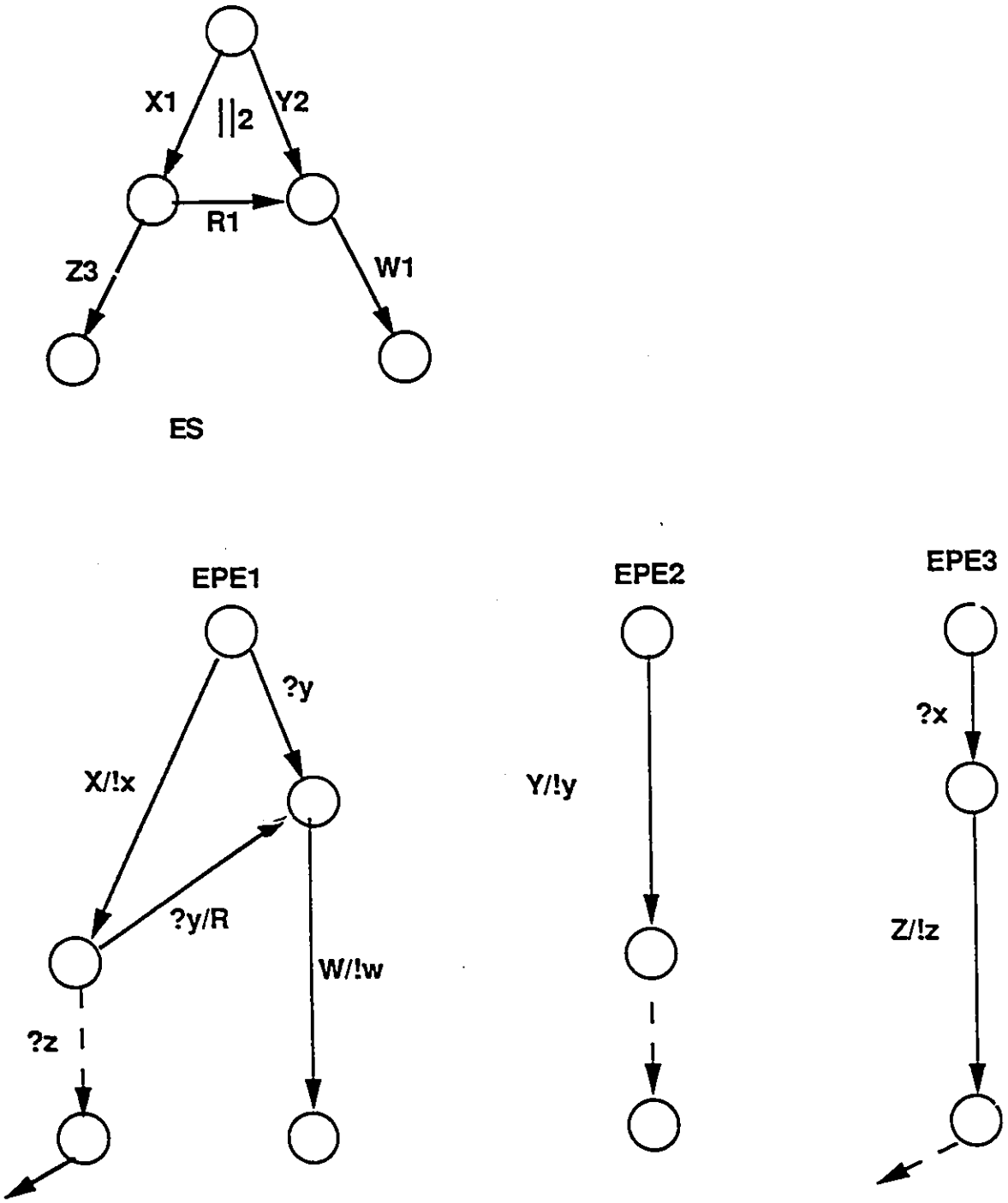



**Figure 6.6.** Potential divergence of EPE3 caused by a collision at n.

*Conditions for Divergence*

In a multi-user service, at a mixed state *n*, let ni and nj be the sink states reached after executing all consecutive SPs occuring at SAPi and SAPj, respectively. We can identify three conditions that can potentially cause a divergence.

a. State *n* is of type ||i and *OUT*(nj) - *OUT*(*n*) ≠ Φ. EPEk, for every k in *OUT*(nj)-*OUT(n)*, will potentially diverge.
b. State *n* is of type ||j and *OUT*(ni) - *OUT(n)* ≠ Φ. EPEk, for every k in *OUT*(ni)-*OUT(n)*, will potentially diverge.
c. State *n* is of type ||rL or ||r0, and *OUT*(nj) - *OUT(n)* ≠ Φ or *OUT*(ni) - *OUT(n)* ≠ Φ. EPEk, for every k in (*OUT*(ni) U *OUT*(nj)) - *OUT(n)*, will potentially diverge.

The *Reset-or-Timeout* mechanism we introduce below can be applied whenever the divergence conditions are satisfied at a mixed service state.

**Reset-or-Timeout Mechanism:**

i) Condition a or b is satisfied:

For a mixed protocol state *n* of type ||j (j ≠ i) in EPEi, the potentially colliding event *e* is augmented and called e* before being sent to EPEk. The reception of e* will alert EPEk to the potential collision and divergence. Hence, upon reception of e*, EPEk will move to an intermediate state after which, it either rollbacks if the collision occurred (EPEi receives a higher-priority colliding message and immediately sends the rollback message to EPEk) or it proceeds after timing out. To properly implement this mechanism, the timeout must be computed based on the time required to detect a collision by EPEi and the time needed for the rollback message *r* sent by EPEi to reach EPEk. Also, the protocol entity acknowledges the service user it supports by issuing a Reset SP.

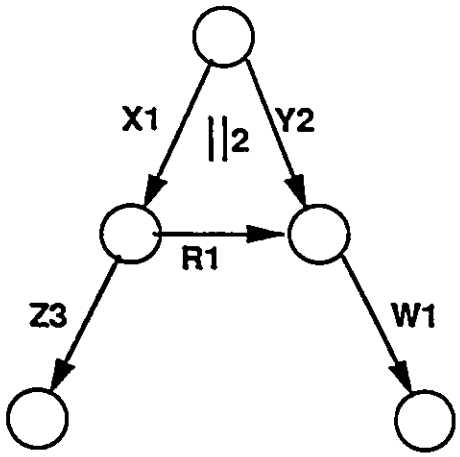
ii) Condition c is satisfied:

For every mixed service state n of type IIrL, add intermediate states in each of the protocol entities involved (the entities sending the colliding messages and the potentially divergent entities). Replace the sink state of the transitions sending the colliding messages and the transitions receiving messages in the divergent entities by the intermediate state created in each of these protocol entities. On timeout, the entities resume their executions at the replaced sink states. However, once a collision is detected, rollback messages are exchanged to resume the execution at the last mixed protocol state of each protocol entity.

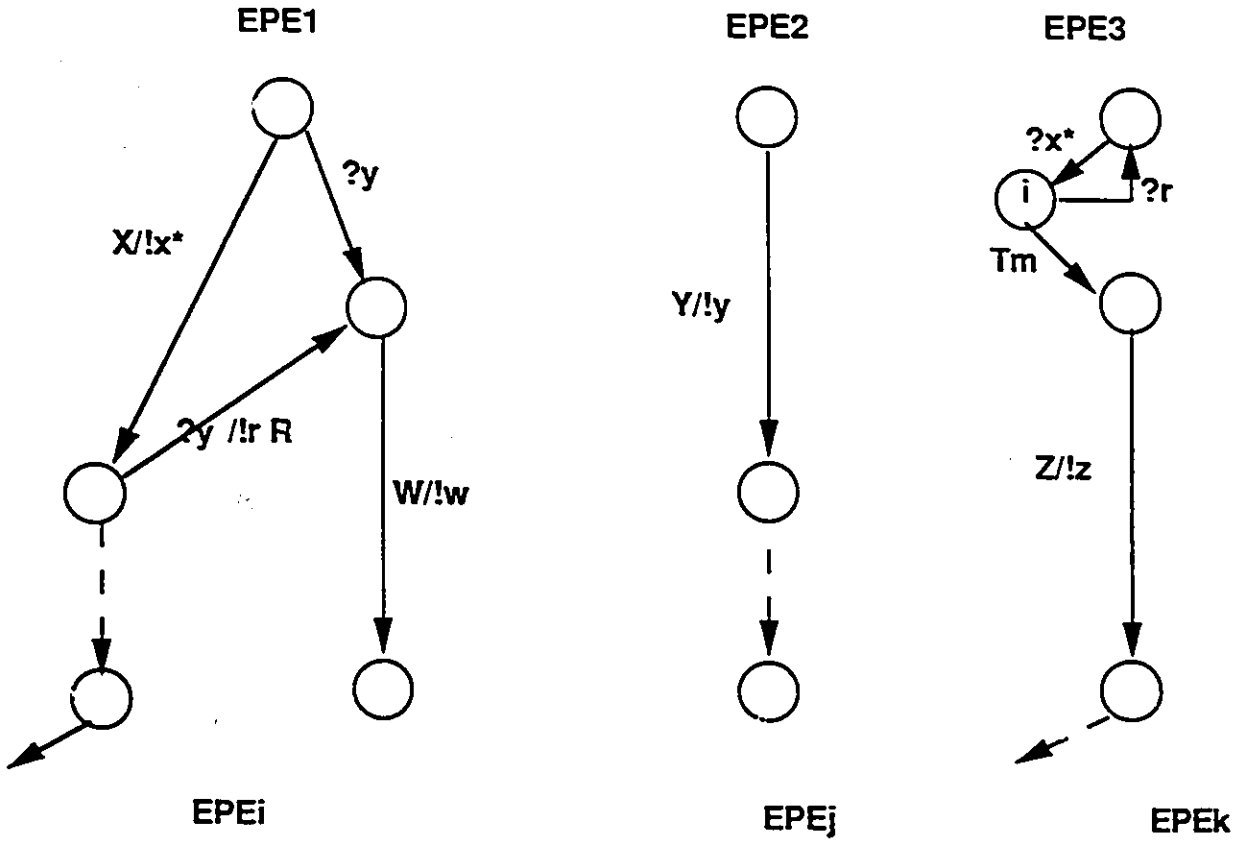
Evidently, this mechanism degrades the speed of the protocol especially when the divergence conditions are satisfied for every mixed state in ES-SPEC.

Figures 6.7 and 6.8 illustrate the divergence avoidance mechanisms explained above for the example in Figure 6.6 when the mixed state is of type II2 and IIrL, respectively.

Extensions to the Transition Synthesis Rules (TSRs) and the synthesis algorithm of the basic service model must be defined to handle the collisions in the protocol specifications according to the different types of mixed service states for a multi-user communication service. These extensions and their proof of correctness are under further study.

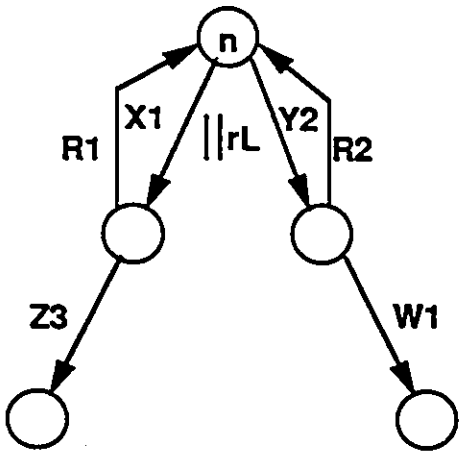



ES




EPEi EPEj EPEk

**Figure 6.7.** Mechanism to avoid the divergence of EPE3.



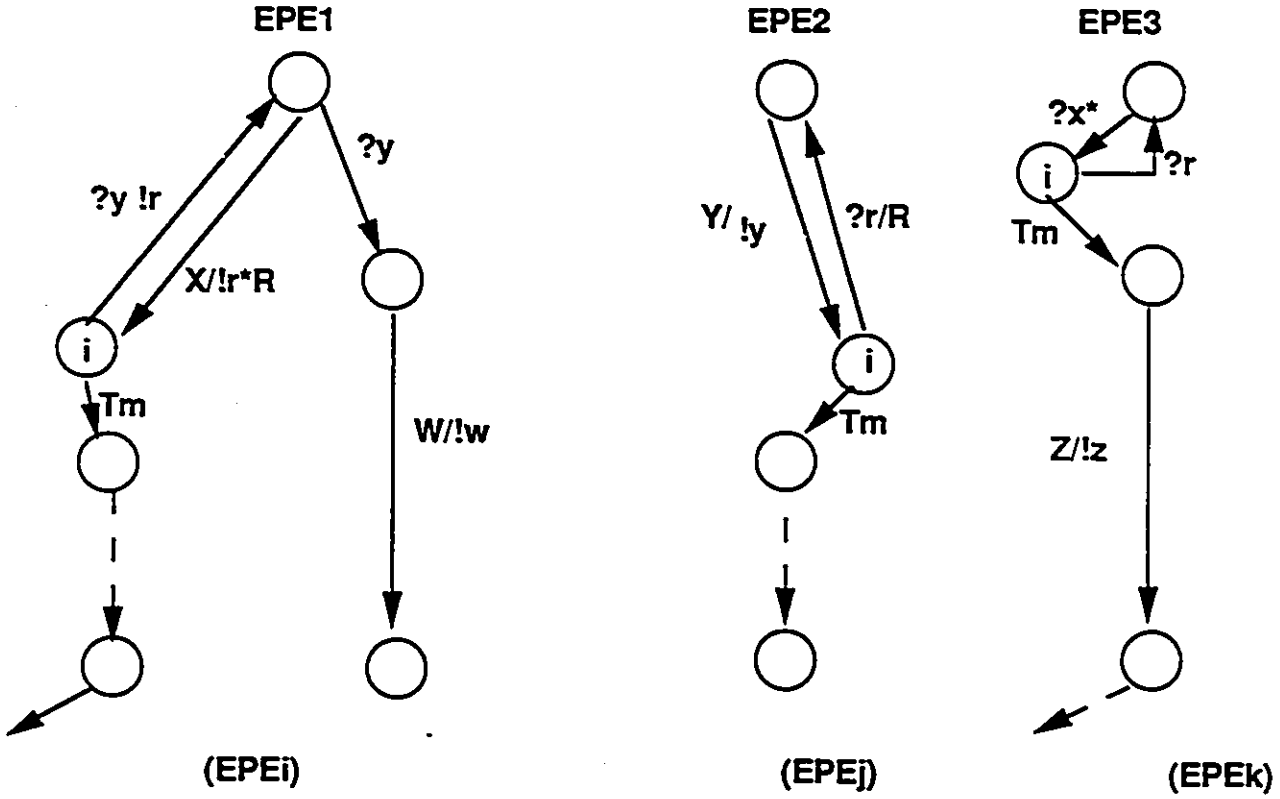




Figure 6.8. Mechanism to avoid the divergence of EPE3 (n of type ||rL).

### 6.1.2. Synthesis of Error-Recoverable Protocol Specifications

Our synthesis method, described in Chapter 5, deals with the derivation of control part of the protocol entities communicating through the communication medium (which possibly abstracts the lower services used by the protocol) in order to provide the set of specified services to the distributed service users. Moreover, the communication medium is considered to be a reliable FIFO channel. By reliability, we mean a channel that never loses, garbles, corrupts or duplicates messages.

Among the ten synthesis methods surveyed and assessed in Chapter 2, only two methods tackle the synthesis of error-recoverable communication protocols:

(i) Ramamoorthy [RAMA 86] describes an extension to the automatic protocol synthesizer (APS) described in [RAMA 85]. Error recovery mechanisms are automatically derived from the error-free version of the protocol that assumes a *reliable* communication medium, and

(ii) Chu [CHUL 88a] uses similar error recovery mechanisms to derive an error recoverable protocol from the error-free version of the same protocol.

However, the approaches used by the two methods to derive the error-free version of the protocol are different: Ramamoorthy constructs the protocol starting from a given complete protocol entity specification and automatically derives the peer entity specification. Chu automatically derives finite state machine protocol specifications from a given finite state machine service specification. Furthermore, both approaches are limited to the synthesis of two-party protocols.

The two methods use the same approach to construct the error-recoverable version of the protocol: First, the respective synthesis method derives the error-free version of the protocol specification in which a lossless and reliable channel is assumed. Then, the specifications are elaborated (extended) using some fixed communication patterns which are used for error-recovery when an unreliable medium is assumed.

In this section, we first introduce the basic error-recovery concepts, then we extend our synthesis method by introducing error-recovery mechanisms in the derived protocol to obtain error-recoverable protocol specifications. We also outline the extended synthesis algorithm for deriving error-recoverable specifications, and we show the semantic and syntactic correctness of the resulting protocol. Finally, we give an example showing the application of the modified synthesis algorithm for the synthesis of error-recoverable protocols.

#### 6.1.2.1. Basic Concepts for Error-Recovery

For the sake of simplicity of our presentation, we will only consider three types of communication errors: the reception of *corrupted messages*, the reception of *duplicate messages* and messages that are sent but *lost* in the communication medium.

Error recovery procedures must be ideally transparent to the service user, meaning that only the protocol entities will be concerned about the recovery, and therefore must provision for transmission errors and provide the mechanisms to recover from them. Furthermore, the service user will not be aware of the occurrence of an error or be involved in its recovery. Therefore, this functional extension to the synthesis method requires no modification to the service specification model. We also assume that the communication

medium is *fair*, meaning that if a transmission error occurs, eventually within a predefined finite time, the protocol will recover and resume its normal behavior.

Basically, error-recovery mechanisms consist of fixed patterns of additional states and transitions (in a finite state machine model) required to allow the communicating entities to return to a normal behavior execution after an error transmission has occurred and has been detected. Therefore, an error recovery pattern should be associated with each occurrence of message transmission in each of the derived protocol entities. Basically, this pattern should provision for retransmission of a previously transmitted message when an error is detected.

A message can be either an *acknowledgment* or a *protocol* message, and both types of messages are equally likely to be lost or corrupted due to the non-discriminating nature of the communication medium. Furthermore, an acknowledgment message can be either a positive or a negative acknowlegment. Corrupted and lost messages can be treated similarly by a timeout mechanism that will resend the (lost or corrupted) message.

A simple error-recovery mechanism must ensure that eventually every message transmitted by one entity (E1) is properly received by the destination entity (E2). We assume that a timing mechanism exists at the sending entity which starts a timer when a protocol message is sent and stops the same timer when an acknowledgment for the previously sent message is received. Furthermore, when the timer expires after time $\tau$, the protocol entity which started the timer will progress by executing the corresponding timeout transition (labeled by Tm).

In the following, we consider all possible scenarios of message exchanges, losses, corruptions and duplications when a two-party protocol is involved:

i. Transmission of a protocol message m by E1:

a. A protocol message is sent by E1 and properly received by E2.
b. A protocol message is sent by E1 but lost in the channel.
c. A protocol message is sent by E1 but received corrupted by E2.
d. A protocol message is sent by E1 and properly received by E2 after some delay ($\delta$).

ii. Transmission of an acknowledgment message ack by E2:

e. An acknowledgment to a received protocol message is sent by E2 and received properly by E1.
f. An acknowledgment to a received protocol message is sent by E2 and lost in the channel.
g. An acknowledgment to a received protocol message is sent by E2 but received corrupted (but recognized) by E1.
h. An acknowledgment is sent by E2 and properly received by E1 after some delay ($\delta$).

Scenarios a and e involve normal message exchanges, however d and h involve a message duplication error, scenarios b and f involve lost messages, scenarios c and g involve corrupted messages. Furthermore, in scenarios d anf h, we assume that $\delta > \tau$.

A simple error recovery mechanism called the Positive Acknowlegment and Retransmission (PAR) can be used to recover from the abnormal scenarios listed above (Figure 6.9).

***Assumptions:***

*Ignoring corrupted messages:* When a corrupted message is received, the protocol entity will just ignore it and will not progress to a new state. Therefore, the sending entity will timeout and resend the same message again.

*Timeout and message loss:* A protocol message will be assumed lost in the channel if, after a time $\tau$ equal to the time delay for a message to arrive to its destination and its positive acknowledgment to be sent back, no acknowledgment is received.

*Message delays:* Since $\delta > \tau$ is assumed, the protocol entity must be ready to accept a delayed protocol message or an acknowledgment; otherwise, an unspecified reception error will occur.

*Fairness:* Eventually, after a finite number of retransmissions of a (lost or corrupted) message, the message will be received properly at its intended destination.

*Throughput and performance enhancement:* The approach to handling corrupted messages as simply lost messages introduces a degradation of the throughput and processing speed of the protocol. By discriminating between lost and corrupted messages, the error recovery mechanism can be based on a positive or negative achnowledgment and retransmission (PNAR) mechanism in which, if a corrupted message is received, a negative acknowledgment (NACK) is sent back to the originating entity, which then immediately retransmitts the message instead of waiting for a timeout as in the case of the PAR mechanism. Figure 6.10 shows the PNAR mechanism applied to a pattern of one message transmission and reception.

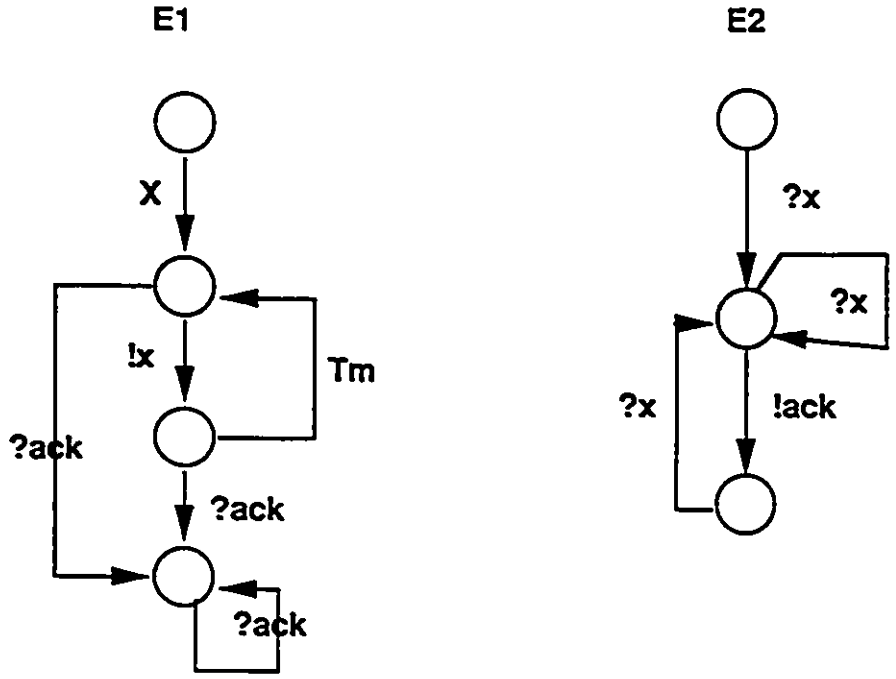



**Figure 6.9.** Positive acknowlegment and retransmission (PAR) mechanism for error recovery.

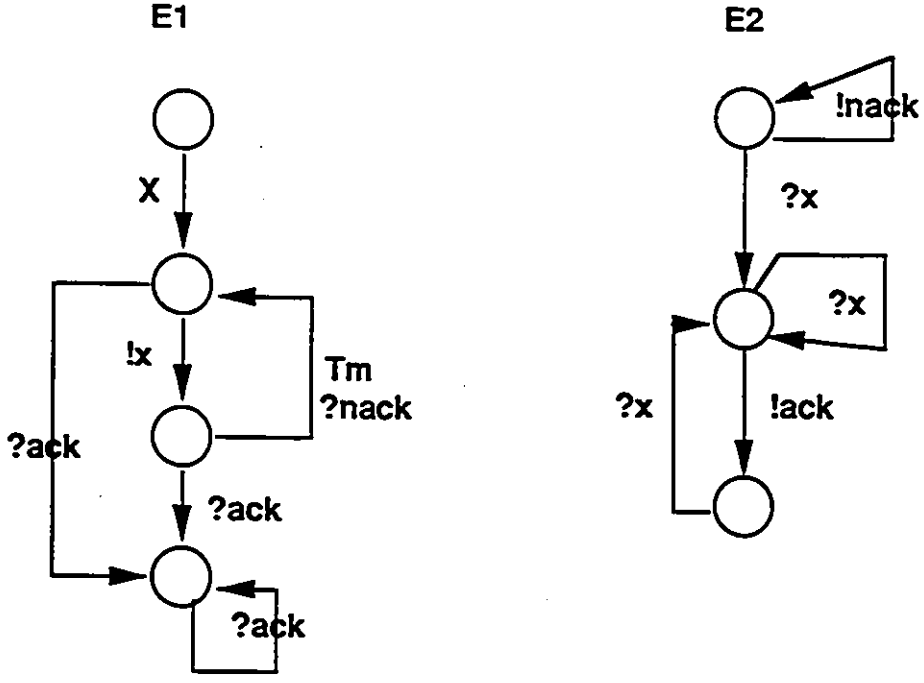



**Figure 6.10.** Positive or negative acknowlegment and retransmission (PNAR) mechanism.

The recovery from duplicate messages is dealt with by attaching a sequence number for every transmitted message and by attaching the same sequence number to acknowledge receipt of the same message by the destination entity. A discussion of the problem and its solutions is provided in [CHUL 88a] and [TANE 88].

### 6.1.2.2. Extension to the Synthesis Method

In our synthesis method, we will apply the simplified PAR error-recovery mechanism for every pattern of message transmission and reception. However, any other enhancement to the mechanism, such as the PNAR, could be applied as well. This extension to the method does not require any changes to the service or protocol FSM specification model as defined in Chapter 5. However, we will refer to the protocol synthesized by the extended method as the error-recoverable protocol ERP, and to the constituent protocol entities as the ERPEs.

#### A) Error-Recovery for Two-Users Services

As a result of the application of the transition synthesis rules (TSRs) introduced in Chapter 5, the possible communication pattern which can exist in the protocol corresponds to the case of a pair of consecutive SPs in the service specification each occurring at a different SAPs. In this case, we first apply the PAR mechanism for every message transmission and its corresponding reception. Then, we add two other message reception transitions to prevent an unspecified reception error that can be caused by a loss of an acknowledgment message.

Figure 6.11 shows the synthesized error-recovery patterns. In ERPE2, two consecutive send transitions (!ack !b) are present. The reception of b by ERPE1 before receiving the

acknowledgment message implies that the latter message was lost, and therefore ERPE1 must be ready to accept b at two other states prior to the transmission of b's acknowlegment. In general, whenever a sequence !e' ?ack ?e" exists in ERPEi, we should synthesize two other sequences as shown in Figure 6.11.

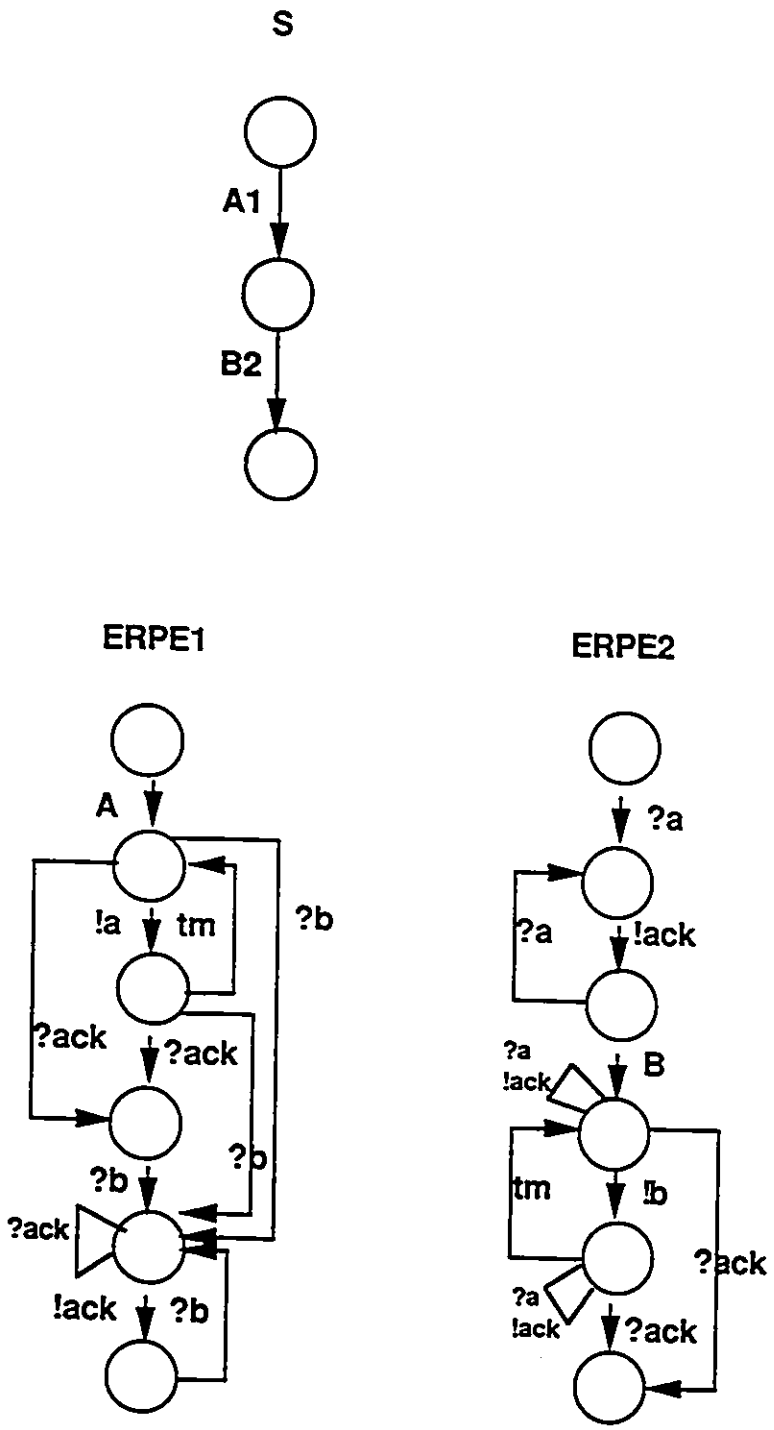



**Figure 6.11.** Error-recovery patterns (two SPs at different SAPs).

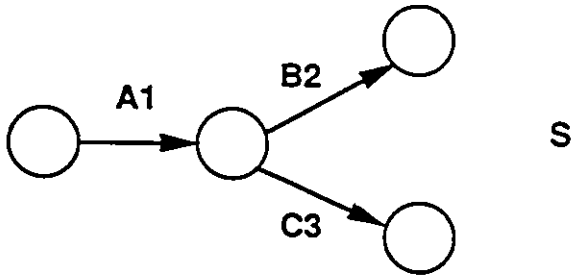
B) Error-Recovery for Multi-user Services

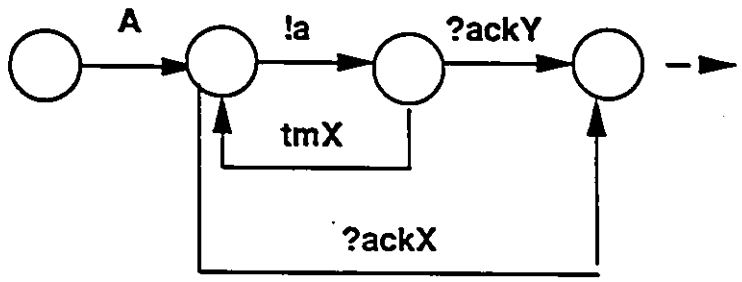
The application of the error-recovery patterns for multi-user services requires a careful examination. At a mixed protocol state, an acknowledgment message may be expected from two or more protocol entities. The question that arises is what to do when none or some of the acknowledgment messages is received and what is the meaning of a timeout on an acknowledgment in this case.

To avoid a deadlock, we have to make sure that a transmitted message is received at all SAPs in *OUT(n)*, where n denotes the mixed state. Therefore, the timeout transition must indicate the set of originating entities (X) for which the acknowledgment message was not received. The message will be retransmitted to all SAPs in X. The solution tends to favor those protocol entities supporting the SAPs in *OUT(n)* which reach their respective mixed protocol states first. Figure 6.12 shows an example of error recovery patterns for multi-party protocols.

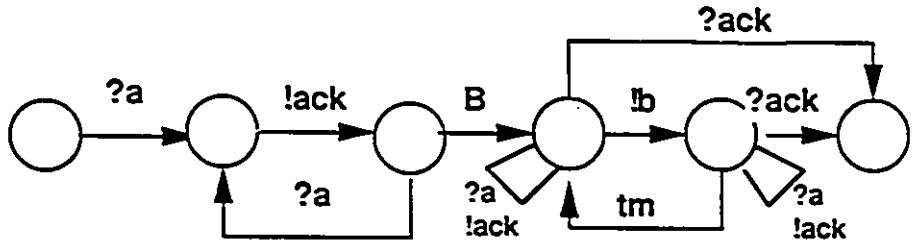
Note that the atomicity rules (Step 3 of the synthesis algorithm described in Chapter 5) need not be applied for the synthesis of error-recoverable protocols, since an intermediate state is required for the separation between a protocol event and a service primitive. The loss of a transmitted protocol event will be detected at this intermediate state.

Extensions to the Transition Synthesis Rules (TSRs) and the synthesis algorithm of the basic service model must be defined to add error-recovery patterns in the protocol specifications. These extensions and their proof of correctness are under further study.

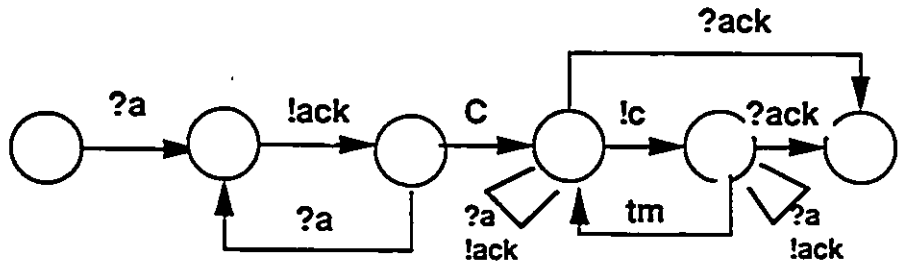



ERPE1




ERPE2




ERPE3

**Figure 6.12.** Error-recovery patterns for multi-party protocols.

## 6.2. Conclusions and Suggestions for Further Research

The main motivation for this thesis was the study of techniques for the automatic generation or synthesis of software structures. Control flow structures for communication software (protocols) presented a very attractive field to study the feasibility of such synthesis techniques.

The contributions of this thesis are summarized below:

(1) Survey and comparison of existing protocol synthesis methods presented in Chapter 2.

(2) An efficient localized and synthetic validation technique has been proposed in Chapter 3. In addition to validating a protocol, this synthetic technique can be extended to synthesize or complete a protocol specification in order to satisfy the service specification.

(3) The importance of the service specification as the starting point for the automatic synthesis of reliable communication protocols has been emphasized in Chapter 4.

(4) A new automatic, service-oriented synthesis method for the design of communication protocols starting from the service specification has been introduced in Chapter 5. Given the services a protocol is supposed to provide, this method automatically synthesizes the specification of the communicating protocol entities. Both service and protocol specifications are modeled by FSMs. The method also guarantees the syntactic and semantic correctness of the synthesized protocol.

Proposed extensions to the synthesis method and to the specification model for the handling of both concurrent behaviors of service users and unreliability of the medium were also presented.

The extensions proposed in Section 6.1 must be formalized. Additional synthesis rules must be defined and the synthesis algorithm must be modified. The proofs of syntactic and semantic correctness of the synthesized protocol for both extensions must also be provided.

Although the service-oriented method we introduced is more general and simpler than other methods, more work has to be done to address some of the outstanding issues that need to be resolved in order to make synthesis techniques more applicable and useful for the design of real-life protocols. A partial list of these issues is given below:

- ***Handling Concurrency.*** A formal characterization of well-formed concurrent service specification must be introduced. Rules for handling such formalized concurrent behaviors, and their proof of correctness must be provided in the extended synthesis method.

- ***Use of Formal Description Techniques.*** In the context of the OSI reference model and the activities of the International Organization for Standardization (ISO) related to the definition of standard formal description techniques (FDTs), it appears desirable that a synthesis method uses one of the FDTs to specify both the service and the synthesized protocol. In order to promote applicability to real protocols and services, it will be necessary for standards groups to develop internationally agreed-upon FDT representations of real services.

- ***Flexibility of the Synthesis Process.*** The synthesis method should allow more flexibility in the synthesis process. Thus, various communication patterns for message exchanges should be considered. This implies that some way of interacting with the protocol designer must be provided in order to allow the selection of particular patterns and to make the synthesis process more application-sensitive.

- ***Integration into a Protocol Design Environment.*** The synthesis method should be integrated into a protocol (design) engineering environment, in which, not only the service at the upper SAPs is considered, but also the services offered by the underlying service at the lower SAPs should be taken into consideration during the design process. This will make the method more suitable for the design of protocols in a layered architecture such as the OSI reference model.

- ***Synthesis of Testable Communication Software.*** A desirable feature of a synthesis method is to synthesize highly testable protocol specifications. The testability of the protocol can be considered as a specific protocol function. More research has to be done to extend existing synthesis methods to accomodate such desirable feature. In particular, the relationships need to be identified and investigated between the correctness of synthesis and the completeness of protocol tests.

- ***Definition of the service.*** Since ideally a synthesis method starts from a complete and correct service specification, more research has to be directed toward techniques and tools for the definition of services which are complete, correct, and highly modular. This is an interesting and difficult open problem since it involved verification of non-formalized intentions.

We believe that the service-oriented synthesis technique developed in this thesis can be used as a basis for further study of the issues identified above.

## REFERENCES


[BARR 79] Barrett W.A. and Couch J.D., Compiler Construction: Theory and Practice, Science Research Associates, 1979.

[BART 77] Bartussek A.W. and Parnas D.L., "Using traces to write abstract specifications for software modules", *UNC Report TR 77-012*, Univ. North Carolina, Chapel Hill, N.C., 1977.

[BELI 88] Belina F. and Hoegrefe D., "The CCITT: specification and description language SDL", *Computer Networks and ISDN systems*, Vol. 16, 1988, pp. 311-341.

[BERT 82] Berthelot G. and Terrat R., "Petri nets theory for the correctness of protocols", *IEEE Trans. on Commun.*, COM-30, No. 12, Dec. 1982, pp. 2497-2505.

[BILL 82] Billington J., "Specification of the transport service using the Numerical Petri nets", in *Proc. 2nd IFIP International Workshop on Protocol Specification, Testing, and Verification*, North-Holland, May 1982, pp. 77-100.

[BLUM 86] Blumer T.P. and Sidhu D.P., "Mechanical verification and automatic implementation of communication protocols", *IEEE Trans. on Soft. Eng.*, SE-12, Aug. 1986, pp. 827-843.

[BOCH 77] Bochmann G.V. and Gescei J., "A unified method for specification and verification of protocols", *Proc. IFIP 77*, 1977, pp. 229-234.

[BOCH 78] Bochmann G.V., "Finite state description of communication protocols", *Computer Networks*, Vol. 2, No. 4/5, Sept.-Oct 1978. pp. 361-372.

[BOCH 80] Bochmann G.V. and Sunshine C., "Use of formal methods in communication protocol design", *IEEE Trans. on Commun.*, COM-28, No. 4, Apr. 1980, pp. 624-631.

[BOCH 86] Bochmann G.V. and Gotzhein R., "Deriving protocol specifications from service specifications", in *Communications, Architectures & Protocols, ACM SIGCOMM'86*, 1986, pp. 144-156.

[BOLO 87] Bolognesi T. and Brinksma E., "Introduction to the ISO specification language LOTOS", *Computer Networks and ISDN Systems*, Vol. 14, No. 1, 1987, pp. 25-59.

[BRAN 78] Brand D and Joyner W.H., "Verification of protocols using symbolic execution", *Computer Networks*, Vol. 2, No. 4/5, Sept.-Oct. 1978, pp. 351-360.

[BRAN 80] Brand D. and Zafiropulo P., "Synthesis of protocols for unlimited number of processes", *Proc. Trends and Application: 1980 Computer Network Protocols*, NBS, Gaithersberg, MD, 1980.

[BRAN 83] Brand D. and Zafiropulo P., "On communicating finite state machines", *J.ACM*, Vol. 30, No.2, Apr. 1983, pp. 323-342.

[BROO 84] Brooks S.D., Hoare C.A.R. and Roscoe A.W., "A theory of communicating sequential processes", *J. ACM*, Vol. 31, July 1984, pp. 560-599.

[BUDK 87] Budkowski S. and Dembinski P., "An introduction to ESTELLE: a specification language for distributed systems", *Computer Networks and ISDN systems*, Vol. 14, No. 1, pp. 3-24, 1987.

[BUHR 83] Buhr R.J.A., System Design with ADA, Prentice Hall, 1983.

[CCITT 88] CCITT recommendations Z.100: Specification and Description Language SDL, APIX-35, 1988.

[CHOI 86a] Choi T.Y., "A sequence method for protocol construction", in *Proc. 6th IFIP International Workshop on Protocol Specification, Testing and Verification*, Montreal, Canada, June 1986, pp. 9/1-9/18.

[CHOI 86b] Choi T.Y. and Miller R., "Protocol analysis and synthesis by structured partitions", *Computer Networks and ISDN systems*, Vol. 11, pp. 367-381, 1986.

[CHOW 85] Chow C.H. and Gouda M.G., "A discipline for constructing multi-phase communication protocols", *ACM Trans. on Computer Systems*, Vol. 3, No. 4, Nov. 1985, pp. 315-343.

[CHUL 88a] Chu P.M. and Liu M.T., "Synthesizing protocol specifications from service specification in FSM model", in *Proc. Computer Networking Symp.*, pp. 173-182, April 1988.

[CHUL 88b] Chu P.M. and Liu M.T., "Protocol synthesis in a state-transition model", in *Proc. COMPSAC'88.*, pp. 505-512, 1988.

[DANT 77] Danthine A., "Petri-nets for protocol modeling and verification", in *Proc. Computer Networks Teleprocessing Symposium*, vol. II, Budapest, Hungary, Oct. 1977.

[DANT 80] Danthine A., "Protocol representation with finite state models", *IEEE Trans. on Commun.*, COM-28, No. 4, Apr. 1980, pp. 632-643.

[DIAZ 82] Diaz M., "Modeling and analysis of communication and cooperation protocols using Petri net based models", in *Proc. 2nd IFIP International Workshop on Protocol Specification, Testing, and Verification*, May 1982.

[EILE 74] Eilenberg S., Automata, Languages and Machines, Academic Press, New York, 1974, pp. 19-21.

[FDT 88-89] First and Second International Conference on Formal Description Techniques for distributed systems, 1988 and 1989.

[GARA 89] Garavel H. and Najm E., "TILT: from LOTOS to labelled transition systems", in *The formal description technique LOTOS*, pp. 327-336, North-Holland, 1989.

[GOTZ 90] Gotzhein R. and Bochmann G.V., "Deriving protocol specifications from service specifications including parameters", to appear in *ACM TOPLAS.*

[GOUD 84a] Gouda M.G. and Yu Y.T., "Protocol validation by maximal progress state exploration', *IEEE Trans. on Commun.*, COM-32, 1984, pp. 94-97.

[GOUD 84b] Gouda M.G. and Yu Y.T., "Synthesis of communicating finite-state machines with guaranteed progress", *IEEE Trans. on Commun.*, COM-32, July 1984, pp. 779-788.

[GOUD 84c] Gouda M.G., "Closed covers: to verify progress for communicating finite state machines", *IEEE Trans. on Soft. Eng.*, SE-10, No. 6, Nov. 1984, pp. 846-855.

[HAAS 85] Haas O., "Formal protocol specification based on attribute grammars", in *Proc. 5th IFIP Intern. Workshop on Protocol Specification, Testing and Verification*, 1985, pp. 39-48.

[HAIL 80] Hailpern B.J., "Verifying concurrent processes using temporal logic", *PhD thesis*, Computer science laboratory, Dept. of electrical engineering, Stanford university, Aug. 1980.

[[HANS 89] Hansson et al., "Specification for verification", in *Proc. 2nd Intern. Conf. on Formal Description Techniques*, Eds., S. Vuong, pp. 347-364, Dec. 1989.

[HARA 78] Harangozo J., "Protocol definition with formal grammars", *Proc. Computer Network Protocols Symposium*, University of Liege, Feb. 1978, paper F6.

[HENN 85] Hennessy M., "Acceptance trees", *Journal of the ACM*, Vol. 32, No.4, Oct.1985, pp. 896-928.

[HOAR 85] Hoare C.A.R, Communicating Sequential Processes, Prentice-Hall, 1985.

[HOFF 85] Hoffman D., 'The trace specification of communications protocols', *IEEE Trans. on Computers*, C-34, No. 12, Dec. 1985, pp. 1102-1113.

[HOLT 88] Holtzmann G.J., "An improved protocol reachability technique", *Software Practice and Experience*, Vol. 18, No. 2, 1988, pp. 137-161.

[HOPC 69] Hopcroft J.E. and Ullman J.D., Formal Languages and Their Relation to Automata, Addison-Wesley. Massachusetts, 1969.

[IEEE 83] Proceedings of the IEEE, Special Issue on OSI, Vol. 71, No. 12, Dec. 1983.

[IFIP 81-90] Protocol Specification, Testing, and Verification, Proceedings of the IFIP WG 6.1 First International Workshop on Protocol Specification, Testing, and Verification. North-Holland, 1981-1990.

[ISO 3252] ISO- Information Processing Systems - 'Proposed draft technical report on guidelines for the application of Estelle, LOTOS and SDL', ISO/IEC/JTC/SC21, Dec. 1988.

[ISO 7498] ISO- Information Processing Systems - 'Basic Reference Model for Open Systems Interconnection', IS 7498, 1983.

[ISO 8649] ISO - Service definition for common-application-service-elements-Part 2: Association control, DIS 8649/2, 1986.

[ISO 8650] ISO - Protocol specification for common-application-service-elements-Part 2: Association protocol, DIS 8650/2, 1986.

[ISO 8807] ISO- Information Processing Systems - Open Systems Interconnection- 'LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour', IS 8807, 1989.

[ISO 9074] ISO- Information Processing Systems - Open Systems Interconnection- 'ESTELLE- A Formal Description Technique Based on tan Extended State Transition Model', IS 9074, 1988.

[ISO 9646] ISO- Information Processing Systems - Open Systems Interconnection- 'Tree and Tabular Combined Notation (TTCN)", Feb. 1989.

[KAKU 86] Kakuda Y. et al., "A new algorithm for fast protocol validation", *Proc. COMPSAC'86, IEEE,* 1986, pp. 228-236.

[KAKU 87] Kakuda Y. and Wakahara Y., "Component-based synthesis of protocols for unlimited number of processes", in *Proc. COMPSAC'87,* pp. 721-730, Oct. 1987.

[KHEN 89] Khendek F., Bochmann G.V. and Kant C., "New results on deriving protocol specifications from service specifications", *ACM SIGCOMM'89 Symp.*, 1989, pp. 136-145.

[KNUT 68] Knuth D.E., "Semantics of context-free languages", Mathematical System Theory, Vol. 2, 1968, pp. 127-145.

[KOHA 79] Kohavi Z., Switching and Finite Automata Theory, McGraw-Hill, 1978.

[LAMS 84] Lam S.S. and Shankar U., "Protocol verification via projections", *IEEE Trans. on Soft. Eng.*, Vol. SE-10, Jul. 1984, pp. 325-342.

[LANG 90] Langerak R., "Decomposition of functionality: a correctness-preserving LOTOS transformation", in Proc. of the tenth IFIP Intern. Symp. on Protocol Specification, Testing and Verification, pp. 203-218, June 1990.

[LINC 88] Lin F.J., Chu P.M. and Liu M.T., "Protocol verification using reachability analysis: The state space explosion problem and relief strategies", *Proc. Data Commun. Symposium*, 1988, pp. 126-135.

[LINS 89] Lin H. and Stovall H., "Self-synchronizing communications protocols', *IEEE Trans. on Computers*, Vol. 38, No. 5, May 1989, pp. 609-625.

[LOGR 82] Logrippo L. and Probert R.L., "Protocol specification-level validation", in *Proc. 2rd Intern. Workshop on Protocol Specification, Testing, and Verification*, 1982, pp. 303-304.

[LOGR 83] Logrippo L., "Constructive and executable specifications of protocol services by using abstract data types of finite state tranducers", in *Proc. 3rd Intern. Workshop on Protocol Specification, Testing, and Verification*, 1983, pp. 111-124.

[MAXE 87] Maxemchuck N.F. and Sabnani K., "Probabilistic verification of communication protocols", in *Proc. 7th IFIP International Workshop on Protocol Specification, Testing, and Verification*, Zurich, May 1987, pp. 307-320.

[MCLE 84] McLean J.A., "A formal method for the abstract specification of software", *J.ACM*, Vol. 31, No. 3, July 1984, pp. 600-627.

[MERL 76] Merlin P.M., "A methodology for the design and implementation of communications protocols", *IEEE Trans. on Commun.*, COM-24, No. 6, June 1976.

[MERL 83] Merlin P.M. and Bochmann G.V., "On the construction of submodule specifications and communication protocols", *ACM TOPLAS*, Vol. 5, No. 1, Jan. 1983, pp. 1-25.

[MILL 90] Miller R., "Protocol verification: the first ten years, the next ten years; some personal observations", in *Proc. 10nt Intern. Workshop on Protocol Specification, Testing, and Verification*, 1990 (invited paper).

[MILN 80] Milner R., A Calculus of Communicating System, LNCS vol. 92, 1980.

[MILN 89] Milner R., Communication and Concurrency, Prentice-Hall, 1989.

[RAMA 85] Ramamorthy C.V., Dong S.T. and Usuda Y., "An implementation of an automated protocol synthesizer (APS) and its application to the X.21 protocol", *IEEE Trans. on Software Engineering*, Vol. SE-11, No.9, Sept.1985, pp. 886-908.

[PROB 89] Probert R., Ural H. and Hornbeek M., "A comprehensive software environment for developing standardized conformance test suites", *Computer Networks and ISDN Systems*, Vol. 18, 1989/90, pp. 19-29.

[RAMA 86] Ramamoorthy C.V., et al., "Synthesis of error-recoverable protocols", *ACM SIGCOMM'86 Symp.*, 1986, pp. 227-235.

[RUDI 88] Rudin H., "Protocol engineering: A critical assessment", in *Proc. 8th IFIP International Workshop on Protocol Specification, Testing, and Verification*, 1988.

[SAJK 84] Sajkowski M., "Protocol verification techniques: status quo and perspectives", in the *Fourth IFIP Intern. Symp. on Protocol Specification, Testing and Verification*, June 1984.

[SALE 88] Saleh K., "An incremental protocol design and validation approach", *Ph.D. Candidacy paper*, University of Ottawa, Ottawa, Ontario, Canada, Dec. 1988.

[SALE 90a] Saleh K. and Probert R.L, "A localized synthetic approach to protocol validation", Proceedings of the *IFIP TC6 International Conference on Computer Networking COMNET'90*, Budapest, May 1990, pp. 308-318.

[SALE 90b] Saleh K. and Probert R.L, "Synthesis of error-recoverable protocol specifications from service specifications", Proc. of the *Second Intern. Conf. on Computing and Information (ICCI'90)*, Niagara Falls, May 1990, pp. 428-433.

[SARA 87] Saracco R. and Tilanus P.A.J., "CCITT SDL: Overview of the language and its applications", *Computer Networks and ISDN Systems*, Vol. 13, No. 2, 1987, pp. 65-74.

[SCHW 82] Schwartz R.L. and Melliar-Smith P.M., "From state machines to temporal logic: specification methods for protocol standards", *IEEE Trans. on Commun.*, Vol. COM-30, No. 12, Dec. 1982, pp. 2486-2496.

[SIDH 82] Sidhu D.P., "Protocol design rules", in *Proc. 2nd Intern. Workshop on Protocol Specification, Testing, and Verification*, 1982, pp. 283-300.

[SIDH 83] Sidhu D.P., "Protocol verification via executable logic specifications", in *Proc. 3rd Intern. Workshop on Protocol Specification, Testing, and Verification*, 1983, pp. 237-250.

[SUNS 79] Sunshine C., "Formal techniques for protocol specification and verification", *IEEE Computer*, Vol. 12, Sept. 1979, pp. 20-27.

[TANE 88] Tanenbaum A., Computer Networks, Prentice Hall, second edition, 1988.

[TENG 78] Teng A.Y. and Liu M.T., "A formal approach to the design and implementation of network communication protocol", *Proc. of COMPSAC*, 1978.

[URAL 86] Ural H., "Specification of distributed systems in PROLOG", *Journal of Systems and Software*, Vol. 11, 1990, pp. 143-154.

[VISS 85] Vissers C. and Logrippo L., "The importance of the service concept in the design of data communication protocols", in *Proc. 5th Intern. Workshop on Protocol Specification, Testing, and Verification*, 1985, pp. 3-17.

[VISS 89] Vissers C.A. et al., The Architecture of Interaction Systems, *Lecture Notes*, Twente University of Technology, The Netherlands, May 1989.

[VUON 82] Vuong S.T. and Cowan D.D., "A decomposition method for the validation of structured protocols", in *Proc. INFOCOM*, 1982, pp. 209-220.

[WEST 78] West C.H., 'An automated technique of communication protocol validation', *IEEE Trans. on Commun.*, Vol. COM-26, No. 8, Aug. 1978, pp. 1291-1295.

[WEST 86] West C.H., "Protocol validation by random state exploration", in *Proc. 6th IFIP International Workshop on Protocol Specification, Testing, and Verification*, Montreal, June 1986, pp. 233-242.

[WEST 89] West C.H., "Protocol validation in complex systems", ACM SIGCOMM'89 Symp., Sept. 1989, pp. 303-312.

[YUAN 88] M.C. Yuang M.C., "Survey of protocol verification techniques based on finite state machine models", in *Proc. of Computer Networking Symp.*, 1988, pp. 164-172.

[YUAN 89] Yuang M.C. and Kershenbaum A., "Parallel protocol verification using the localized approach: the two-phase algorithm", in *Ninth IFIP Intern. Symp. on Protocol Specification, Testing, and Verification*, June 1989.

[ZAFI 78] Zafiropulo P., "Protocol validation by duologue matrix analysis", *IEEE Trans. on Commun.*, Vol. COM-26, 1978, pp. 1187-1194.

[ZAFI 80] Zafiropulo P. et al., "Towards analyzing and synthesizing protocols", *IEEE Trans. on Commun.*, Vol. COM-28, No.4, Apr. 1980, pp. 651-661.

[ZHAN 88] Zhang X.Y. et al., "An interactive protocol synthesis algorithm using a global state transition graph", *IEEE Trans. Soft. Eng.*, Vol. SE-14, No. 3, Mar. 1988, pp. 394-404.

[ZHAN 89] Zhang Y.X., "A study on design method of communication protocols and its support systems", *Ph.D. Dissertation*, Tohoku University, Sendai, Japan, 1989.

[ZIMM 80] Zimmermann H., "OSI reference model- The ISO model of architecture for Open Systems Interconnections", *IEEE Trans. on Commun.*, Vol. COM-28, No. 4, Apr. 1980, pp. 651-661.