National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

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.

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.

Canada

Testability-Directed Specification of Communications Software

BY

Hua-Long Yu

THESIS

Submitted to the School of Graduate Studies and Research

in Partial Fulfillment of the Requirements for the

Degree of

Master of Science

IN

Computer Science

University of Ottawa

ISBN 0-315-80057-7

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

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: Professor Kessam Saleh of Concordia University, Professor Dave Parnas and Mr. Ya-Bo Wang of McMaster University for many helpful discussion during the progress of my work; as well as Professor Gerald Karam of Carleton University and Professor Luigi Logrippo of University of Ottawa, in the board of examiners of my thesis for their helpful comments on my thesis.

My deepest respect goes to my parent who gave me the moral support and encouragement to further my education in abroad - Canada.

Finally, I gratefully acknowledge financial support by the Telecommunications and Research Institute of Ontario (TRIO) and NSERC.

ABSTRACT

In terms of one common model for software engineering, there are four main phases in developing or evolving reliable, large-scale software. These are formal specification, design, implementation, and testing phases. Furthermore, it is widely recognized (but not necessarily widely adopted in practice) that the development of reliable communications software and systems should proceed from formalized specifications of the system requirements. However, very few pragmatic guide-lines exist for deriving effective, formal specifications of reliable software, and subsequently relating the specification method to testing activities. This area of relationship between completeness and effectiveness of specifications and testability of the corresponding systems is particularly critical for communications software which tends to be extremely large, complex, and highly distributed.

Experience has shown for such systems that the earlier relative timing information is made available in the development process, the earlier potential design errors and omissions can be identified. Timing aspects are one example of specification information which is important for correctly designing and adequately testing communications software. In addition, precise, complete specifications are necessary for automating the test execution and test result analysis steps. In turn, this test automation is critical to cost-effective testing of very large, distributed systems. In this thesis, we make a contribution to improving software testability by providing a useful model and guide-lines for constructing highly testable specifications of distributed real time (communications) systems.

In particular, initially, a model, namely global events model, is developed based on relative clock for interpreting concurrent aspect of a communications system. Then, based on the global event model, a new formal specification language and method, namely extended trace assertion language (ETAL), is presented for formally representing both the sequential and concurrent aspects of the system. A relative clock based specification in ETAL method also facilitates subsequent testing activity. Subsequently, based on the global events model, a new test result analysis approach including timing information is presented for communication service and protocol conformance testing. This approach is also ETAL specification-based. After the descriptions of the contributions mentioned above, the definition of a testability-directed specification and a study on the relations between the testability-directed specification and ETAL are then presented. This study can be regarded as one of contributions of this thesis for the area of software engineering in general.

Finally, the feasibility and usefulness of the relative clock based formal specification method is demonstrated by its application to OSI transport software (service and protocol). Also, the effectiveness of the relative clock based approach for test result analysis is illustrated in a set of realistic conformance testing examples. We also illustrate in these examples, the relationship between our formal specification method and our approach for test result analysis. This relationship is called "testability-directed specification". Thus, we offer a means of usefully linking the specification and testing phases, a relationship which is often overlooked in other software development approaches.

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

CHAPTER 1

INTRODUCTION

1.1 Context and Motivation for the Thesis

In terms of one common model for software engineering, there are four main phases in developing or evolving reliable, large-scale software. These are formal specification, design, implementation, and testing phases. Furthermore, it is widely recognized (but not necessarily widely adopted in practice) that the development of reliable communications software and systems should proceed from formalized specifications of the system requirements. However, very few pragmatic guide-lines exist for deriving effective, formal specifications of reliable software, and subsequently relating the specification method to testing activities. This area of relationship between completeness and effectiveness of specifications and testability of the corresponding systems is particularly critical for communications software which tends to be extremely large, complex, and highly distributed.

Experience has shown for such systems that the earlier relative timing information is made available in the development process, the earlier potential design errors and omissions can be identified. Timing aspects are one example of specification information which is important for correctly designing and adequately testing communications software. In addition, precise, complete specifications are necessary for automating the test execution and test result analysis steps. In turn, this test automation is critical to cost-effective testing of very large, distributed systems. In this thesis, we make a contribution to improving software testability by

providing a useful model and guide-lines for constructing highly testable specifications of real time communications (distributed) systems.

At a high level of abstraction, a communications system can be viewed as a service provider which offers some specified communications services to a number of service users who access the system through many geographically distributed service access points. The **communications service specification** describes the distributed functions provided by the **communications** system to its service users [Sale91] (see Section 2.1.1).

Conformance testing [ISO9646, Rayn87] is the assessment process to determine the extent to which communications software conforms to the requirements stated in international standards. One of the most important steps in a conformance testing procedure is the analysis of the test results, also called test result analysis. In this thesis, we also show how specifications incorporating relative time constraints are useful for guidelining the conformance testing process, as well as for enhancing the automation of test result analysis.

In recent years, a great deal of research has been done in the area of formal specification methods and conformance testing methods for communications software [PSTV, FORTE, PTS]. In the following, a brief overview on these issues is given.

On one hand, a wide range of specification formalisms and methods have been developed. Each of these specification methods attempts to offer desirable features for formally describing communications services and protocols, such as expressiveness, modularity and conciseness. However, none of them consider testability requirements (see Section 5.2). In addition, concurrent aspects such as collision scenarios are still not well expressed by those methods.

Moreover, although many contributions from both academia and industry have been made to conformance testing, there are not yet any practical solutions to test result analysis including relative timing (i.e., time-ordering) information for service-directed conformance testing (see Section 5.1.2). In addition, service-directed conformance testing has not yet been investigated very well so far. However, the commercial communications services such as telephone services have been increased dramatically in recent years, the studies for service-directed conformance testing is therefore needed urgently for enhancing the reliability of these services.

1.2 Main Ideas of the Solutions

In this section, we briefly introduce the main ideas of the solutions to the problems raised in the last section. Detailed demonstration on the solutions will be provided in the major part of this thesis.

An interesting comments relating system architecture to timed specifications is due to Lamport [Lamp78]: "If a system is to meet a specification correctly, then that specification (of the system) must be given in terms of events observable within the system. If the specification is in terms of physical time, then the system must contain real clocks.". However, in the same paper, it is pointed out that synchronization problems still exist even if the system contains real clocks since clocks used in different distributed locations are not perfectly accurate and do not keep precise physical time. Lamport then introduced a model based on *logical clocks,* rather than real clocks. However, in the paper, Lamport does not (or did not intend to) further develop a formal specification method based on the logical clock.

In this thesis, we extend Lamport's work on logical clocks. **In particular, we use logical (relative) time as a basis for both specifying and testing**

communications (distributed) systems. To avoid possible confusion with Lamport's formalism, we use the term "*relative clock time*" instead of "logical clock time". Relative clock time is defined and described in detail in Chapter 3.

We consider relative time as a virtual time scale to informally show how naturally and powerfully relative time can be used as a basis for specifying and testing a communication (distributed) system. We assume that the range of the scale (relative time) consists of a set of natural numbers. Therefore each local event can be mapped to one of the relative time values in the scale. We then can identify and analyze what happens in the system when more than one event is mapped to the same natural number, what relations exist between events mapped onto different values and what orders of the events are legal, in terms of the values on the scale. For example, in Figure 1.1, events *a* at SAP1 (Service Access Point) and *b* at SAP2 occur at the same relative time. Such capabilities directly enhance the expressive power of specifications and the effectiveness of testing methods.

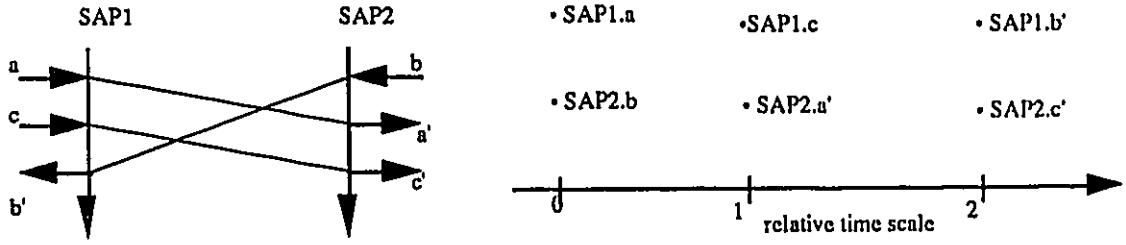

Figure 1.1: events and relative time scale

The trace assertion language (TAL) and method invented by Parnas [BaPa78] and formalized by McLean [McLe84a] is a useful formal specification language and method [PaWa89]. This language specifies a software module or a system in terms of the external observable behaviors of the module or the system. However, we feel that it is not well suited for specifying communications (distributed) systems. We also notice that there exist potential match between the trace assertion language (method) and the relative time based specification

method or test result analysis method. In this thesis, we present an extension of TAL, called ETAL (Extended Trace Assertion Language) to adapt it for specifying and testing distributed systems in terms of relative time. This specification therefore can be used not only to represent collision scenarios but also to validate the system (including time-ordering aspects) as long as the system (or testing environment) contains a global relative clock. We show that a relative clock-based-testing environment may be realized more easily than a real clock-based one; thus, relative clock-based-specifications in ETAL enhance testability.

1.3 Objectives and Overview of Main Contributions

The primary objective of this thesis is to define a expressive model for developing a formal specification method and a practical test result analysis approach for developing reliable communications services and protocols. By doing so, we define and discuss what is a testability-directed specification and recommend a language as a means to write such a specification.

The main contributions of this thesis are:

(1) A model, namely global events model, is developed based on relative clock for interpreting concurrent aspect of communications systems.

(2) Based on the global event model, a new formal specification language and method, namely extended trace assertion language (ETAL), is presented for formally specifying both the sequential and concurrent aspects of communications systems. An ETAL specification of OSI transport service [ISO8072] is also presented.

(3) Based on the global events model, a new, specification-based test result analysis approach with time-ordering information is presented for communications service and protocol conformance testing.

(4) The requirements of testability-directed specification and a discussion on the relationship between testability-directed specifications and ETAL are presented. This study can be also regarded as an contribution of this thesis for the area of software engineering in general.

Note that we will use the standard terminology for open system services and protocols [Zimm80]. In particular, we use OSI transport software as an example to illustrate these contributions. Also note that, our focus is on abstract foundations and basic techniques for specifying and testing communications systems in this thesis.

1.4 Outline of the Thesis

The rest of the thesis is organized as follows.

- In Chapter 2, we provide some background on the main concepts and a survey of previous related work, which are used for the development of the research conducted in this thesis.

- In Chapter 3, we will show why relative clocks are an adequate basis for global events (event-tuples) and, more generally, that the global events model based on relative clocks can adequately represent time-ordering information of sequential and concurrent features of a communications system.

- In Chapter 4, we describe the extension to Parnas' Trace Assertion Language (ETAL) and its application to OSI transport service example based on the global event model.

- In Chapter 5, based on the global events model, a new approach for test result analysis (including time-ordering information) for communications service and protocol conformance testing and is illustrated with a set of realistic conformance testing examples using ETAL specifications.

- In Chapter 6, we presented some extensions and suggestions for further research.

- In Chapter 7, we conclude this thesis.

- In Appendix A, some key algorithms are presented. In Appendix B, we give a specification of the transport service in ETAL to facilitate the discussion in this thesis.

CHAPTER 2

BACKGROUND AND SURVEY ON RELATED WORK

2.1. Background

In this section, we provide only brief overview of the necessary background knowledge. The interested reader is invited to consult the references for more detail.

2.1.1. Communications Services and Protocols

The concept and specification of communications services play a key role in developing reliable computer communications protocols [ViLo86, Boch90a]. The relationship between protocols and services can be described as follows.

A communications protocol consists of a set of rules which govern the orderly exchange of messages among the system components in order to provide a specified set of services to service users located at different access points.

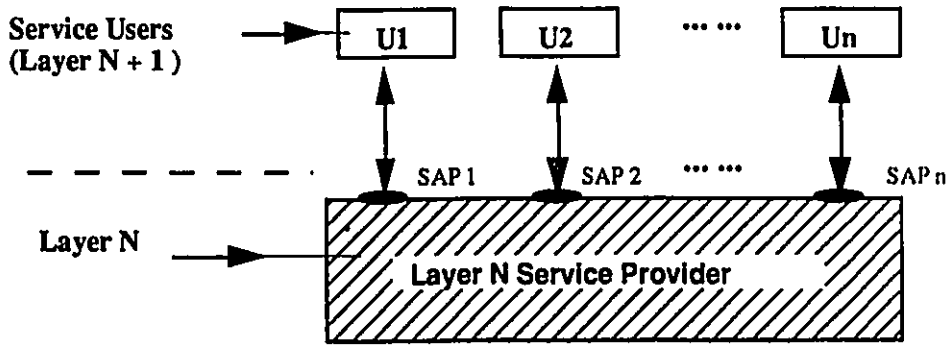

Figure. 2.1: An Abstract View of Distributed Communications Services

At a high level of abstraction, a communications system can be viewed as a service provider (a blackbox) which offers some specified communications services to a number of service users (U1, U2, .., Un) who access the system through many geographically distributed Service Access Points (SAP1, SAP2, ..., SAPn) (Figure 2.1).

At a lower level of abstraction, the communications system can be seen to consist of a number of cooperating protocol entities (PEs) which exchange protocol messages that are not observable to the users at the access points. The PEs exchange these messages over a reliable communications medium according to a FIFO discipline (first-in, first-out).

The global communications **service specification** describes the distributed functions provided by the communications (distributed) system (Layer N) to its service users (Layer N+1) (see Figure 2.1) [Sale91]. Local service specification is an abstract specification of the interface at a given service access point (SAP) of a communications (distributed) system; it includes the local rules about the temporal order of their execution for that SAP [Boch90a]. The communications **protocol specification** describes the behavior of the protocol entities, each servicing a particular access point. A **protocol entity specification** describes the behaviours of that entity with respect to its upper interface (SAPs) and with respect to its lower interface with the underlying service provider [Sale91].

The OSI transport service (**TS**) [ISO8072, Schw87] is the service provided to the Session layer at the boundary between the Session and Transport Layer of the Reference Model [Zimm80, ISO7498]. There are basically two ways to provide the service by the transport layer; one is non-connection-oriented, the other is connection-oriented on which we are focusing in this thesis. The connection-oriented transport service consists of three phases: connection establish phase, data transfer phase and connection release phase.

In general, the function provided by TS is transparent transfer of data between TS users. It relieves these TS users from any concern about the detailed way in which supporting communications media are utilized to achieve this transfer based on the TS protocol entity and the network service. Thus, the specification of transport service describes the function provided by transport layer to session layer.

2.1.2. Trace Assertion Language

Trace Assertion Language (**TAL**) is a technique invented by Parnas and Bartussek [BaPa78] for formal and abstract specification of software module. Work done in this paper can be also regarded as a work on the subject of module interface specification. The basic idea of trace specification language is specifying a module based on a set of assertions on the trace legality. Here, a trace consists of a sequence of observable events from the module, and the legality is legal ordering (usually defined in a requirement document or a standard of a module or a system) relation of the events in a trace.

McLean [McLe84a] made a substantial contribution to the formalization of the work in [BaPa78] by establishing a formal basis for traces, including a formal syntax and semantics, a derivation system, and completeness and soundness results. He also has subsequently explored both their application and the possibility of tools to assist in their use [McWeLa86].

In [Hoff85], Hoffman demonstrated the use of the trace specification method on substantial communications applications and provided an explicit statement of the "single element extension" method of writing the specification. Also, Hoffman and Snodgrass [HoSn88] described techniques for constructing formal, executable models from trace specifications.

Relating to the specification method and its applications, some other papers such as [PaMa90,Scho90] have been published, the interested reader is invited to read the reference.

In this thesis, we use the updated version of the trace assertion language and method [PaWa89] as the basis for the extended assertion language described in Chapter 3. A specification using this method consists of following sections:

- **Syntax section:**

  The section consists of an input table, an output table, an access-program table, and an event table.

- **Canonical trace section:**

  The section defines a predicate on traces, "canonical", with domain that includes the complete set of syntactically correct traces. The predicate of a set of traces such that (a) no two traces in the set are equivalent (≡), (b) every legal trace is equivalent to exactly one trace in the set.

- **Equivalence section:**

  The section presents one function for each class of events, such as the invocation of an access program.

- **Value section:**

  The section comprises the output variable section and the return value section.

In this method, trace assertions represent legal scenarios by means of canonical traces and equivalences. We briefly illustrate the format of a trace assertion for a transport service example, shown in Figure 4.1 and discussed in detail later. Acceptable scripts for transport connection establishment are represented by the following canonical trace expression, and equivalence (mapping function):

At any SAP, if one canonical trace is:

T = send (TCONreq),

then for the trace in which TCONconf appears immediately after trace T, then,

the trace assertion is:

"T.receive (TCONconf)" is said to be acceptable (i.e., legal).

Additional discussion of this example is given in Section 4.2 and Appendix B. More details on the principles of the trace assertion method and further examples can be found in [PaWa89].

2.1.3. Temporal Assertions

Temporal Assertions we use are based on event-based [Vogt82] temporal propositional logic. Temporal propositional logic is derived by adding standard temporal operators [Gotz90]. In our study, we use:

[] : true from now on and always; <> : true now or at some future time.

In our example shown in Figure 4.1, we could assert that a connection request at SAP1 will result in a connection indication appearing at SAP2 sometime in the future, as following :

[] (SAP1: sends "TCONreq" => <> SAP2: receives "TCONind ").

In Section 6.1, we will show how these temporal operators enhance the appropriateness of trace assertions.

2.1.4. Correctness Requirements

Two types of properties must be guarantied in a protocol design to provide a reliable services. They are: (1) the **safety properties** of the protocol which ensure that the protocol never enters an undesirable state, i.e. something bad never happens. These properties include

freeness from deadlocks, freeness from livelocks, and completeness (i.e. absence of unspecified reception errors), and (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, meaning that the protocol performs its intended functions with respect to the service specification. Both properties can be guarantied by verifying the absence of syntactic and semantic design errors. A review of these issues can be found in [PrSa91].

2.1.5. Conformance Testing

Conformance in the context of OSI is concerned only with the conformance of implementations and real systems to those OSI standards which specify applicable requirements. Those standards include protocol standards and transfer syntax standards. In the context of this thesis, the standards concerned here also include the standardized definitions of OSI 7-layers services, for example, the transport service definition [ISO8072].

Conformance testing is the assessment process to determine the extent to which an implementation of a protocol (a service) standard conforms to the requirements stated in the standard. For example, in protocol testing , the tester who applies the test is the entity which decides on the conformance of IUT (Implementation Under Test) to the protocol specification; if a given behaviour is not in conformance of IUT to the protocol specification, an error is detected. An international standard methodology has been developed [Rayn87, ISO96], and will be standardized in the fall of this year. In the following, we only review part of content of this standard, which directly relates to our studies in this thesis.

In this testing methodology, there are five main steps for a test procedure; the first step is the analysis of the PICS accompanying the IUT ; the second step is test selection; the third step is the execution of the parameterized executable tests; the fourth step is the analysis of the test results, also called as test result analysis. A verdict used in the test result analysis is a

statement of **pass** (i.e. pass a test), **fail** (ie. fail to pass a test) or **inconclusive** (i.e. no conclusion on whether pass or fail to pass a test) to be associated with every foreseen outcome in the abstract test suite specification.

The OSI abstract testing methodology is based upon the OSI reference model. Abstract test methods are described in terms of what outputs from the IUT are observed and what inputs to it can be controlled. There are four types of methods: local test method, distributed method, coordinated method and remote method. The last three methods together are also called as external test methods.

Figure 2.2 shows the external testing architecture for coordinated test method on which we will develop our method for test result analysis in Chapter 5. As shown in the architecture, the test system is divided into so-called Upper Tester (UT) and Lower Tester (LT) which access the upper and lower service interfaces of the IUT, respectively. In this test architecture, some form of coordination between the LT and UT is established through the exchange of message according to a so-called test management protocol through a (possibly separate) communications channel between the UT and LT. Test events in this method are specified in terms of (N-1) - ASPs (Access Service Primitives), (N) - PDUs and test management (TM) PDUs.

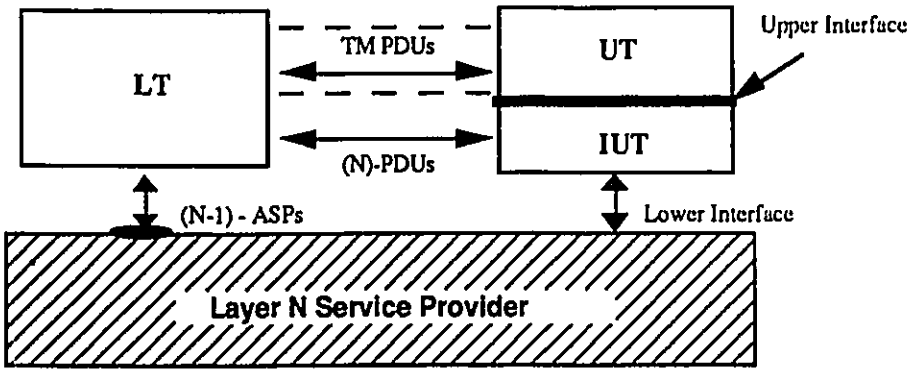

Figure 2.2: Coordinated Testing Architecture

TTCN (Tree and Tabular Combined Notation) presented in the Part 3 of this standard is an abstract standard comprised of a tabular notation for test specification notation (see also [PrMo91]). A test case written in TTCN specifies, as a sequence of atomic test events, the behaviours of a test system and a protocol entity in sufficient detail to judge the behaviour of an implementation of the protocol to formulate a verdict of pass or fail.

2.2. Survey of Related Work

2.2.1. Formal Specification Techniques for Real Time Distributed Systems

A formal specification method for specifying communications systems should have significant expressive power for describing the requirements (or properties) of the systems as well as the capability of highlighting the externally observable distributed characteristics of the systems [SpBr87, HoSn88]. In the following, we briefly review the related work on these topics and explore some unsolved problems.

To specify time-ordering information, particularly concurrent features of a communications (distributed) system, researchers working on specification methodology for the system have developed up to five timed models [Liu89]. Some corresponding specification languages (formalisms) were presented based on those models either before or after the presentations of the models. With respect to timing mechanisms, each presentation of a model, either

i) indicates that the time information expressed in the model is in terms of real time, i.e. real clock based time; or,

ii) just assumes that there exist clocks, whatever the type is, so as to provide a basis for reporting or identifying "time"; or,

iii) interprets concurrency in terms of ordering information, e.g., the interleaved semantics model.

These models, however, either ignore Lamport's observation [Lamp78] that the globally correct (synchronized) (real) clocks are not always available (i & ii), or fail to represent the concurrency adequately (iii). Therefore, a study on what is the nature of concurrency and what are the relationships between the reference clock and specification methods is needed.

Since Lamport's paper was published, some work on how to use Lamport's concurrent concept and logical (relative) clock has been done to solve problems such as distributed synchronization, fault-tolerant and stable property detection, etc. [Jeff85, Morg85, ChLa85]. However, those works mainly focus on solving the classical distributed computing problems without relating clock issues to formal specification languages and methods.

Some authors have mentioned that logical clocks can be used as timers in parallel programming languages such as Occam [Fidg91]. However, these studies do not (or did not intend to) directly relate logical clock to formal specification languages and methods for communications systems. This is perhaps because a partially ordered local event model may not be easily represented in some specification languages such as TAL (recall Section 2.1.2). Moreover, in [Fidg91], spontaneous events [Boch90b, BoDsZh89] are not taken into account (see more discussion in Section 3.1.1). However, specifications and recognition of such events are important for developing reliable communications software in practice.

In fact, as a result of Lamport's study, basing specifications on relative clocks is a natural means of overcoming problems arising from the use of real clocks in specifying the time-ordering aspects of a communications (distributed) system. However, no studies have been

published which directly relate relative clocks to the specification method. In the next chapters, we will present a totally ordered global events - based specification method for describing concurrent aspects of communications (distributed) systems in terms of relative time.

2.2.2. Current Approaches on Test Result Analysis for Conformance Testing

Among the activities in the testing process, the selection of test cases (for instance, [UrYa91]) receives much more attention than the analysis of test results. In fact, both activities are equally important for conformance testing. In this section, we survey current approaches for automated test result analysis using trace analysis. In Chapter 5, we will present a new approach on test result analysis using trace analysis as one of the contributions of this thesis.

There are basically two tasks to be accomplished in test results analysis using trace analysis:

i) correctly recording traces including time-ordering information, and

ii) analyzing the recorded traces with respect to the corresponding timed reference specifications to obtain test conclusions.

Focusing on either or both of these tasks, several papers such as [BoDsZh89, BoBe89, UrPr86, Boch90a] have introduced techniques for automated (untimed) test result analysis. However, the problem of how to merge local timestamped traces for global test results analysis is still unsolved.

The translation from local to global time-ordering information is crucial for achieving an effective global error detection capability. For example, the error detection effectiveness of

the approach in [BoDsZh89] is limited because of the lack of time-ordering information. In fact, there are two limitations associated with timestamped trace recording and analysis:

i) totally ordering distributed local events is not easy based on either relative or real clocks, and
ii) the events used in a local trace are not expressed in event-tuples. Therefore, the specification cannot be directly used as reference specification to test either relative or real concurrent aspects of a communications (distributed) system.

In addition, how to derive reference specifications and the trace analysis module (knowledge), how to **automatically** reconcile test results, particularly the test results of testing concurrent features of services, against the reference specification by the trace analysis module are also still vague. Therefore the present approach is not sufficient for complete and formal specification-based testing and test result analysis, although the total ordering of (local) events can be accomplished based on real clock.

A number of other papers on the application of relative clocks to parallel and distributed program debugging have also appeared. For example, Fidge [Fidg91] claims that all errors in parallel programs could be debugged by logical clock-based techniques in principle. However, some problems are still unsolved for testing a communications system in practice. Examples of important, practical problems are the following:

i) How might we base testing activities on formal specifications in practice? This is desirable for satisfying more stringent testing criteria [Boch90a].

ii) How can we recognize the occurrences of spontaneous events [Boch90b] during testing?

iii) How can we enhance the efficiency of testing method?

We will present some solutions based on the GE model to these problems in Chapter 5.

As mentioned in Chapter 1 and Section 2.1, service-directed conformance testing (associated with service the definition standard) becomes more and more important in developing reliable communications systems. Therefore, an effective test result analysis process is needed for service-directed conformance testing. However, there are not yet practical approaches for the kind of test result analysis methods found in the literature. In Chapter 5, one of applications of our new approach to test result analysis is presented as a possible, practical approach.

CHAPTER 3

A NEW SPECIFICATION TECHNIQUE BASED ON RELATIVE CLOCK

Specifications of concurrent scenarios, such as collisions are helpful, perhaps even crucial for developing a reliable and complete understanding and subsequent trustworthy implementation of communications systems. However, we feel that concurrency is not well understood (recall Section 2.1.1). Therefore, formal studies leading to methods for specifying concurrent scenarios, such as collisions, are needed and will help to improve our understanding of the relationship between specifications and test methods.

In Chapter 1, we have introduced the motivation of the relative clock based specification technique informally. In this chapter, we first introduce a new concept "relative concurrency" based on an extension of the logical clock of Lamport. Following an analysis of the nature of collisions, we present a model called (totally ordered) global events model to interpret and formally define relative concurrency. Later, in Chapter 5, we illustrate by examples how (erroneous) spontaneous events can be recognized by the model. To simplify our discussion of the relative clock-based specification of a communications system, we only discuss the service specification [Boch90a, Gotz90], namely the OSI transport layer [ISO8072]. However, the results derived here can be easily extended to "end-end models" [Sale91].

For representing the service of a system in a more general, testable manner, the following definitions and notations are given in terms of the (service) user point of view, i.e. in terms of "what does the user gets from the service provider", or more precisely, what messages are

sent or received by a user, through an associated Service Access Point (SAP) [Gotz90]. Hence, the structure of each primitive should include an explicit or implicit description of the direction of any message sent or received.

3.1. Relative Concurrency

As compared to previous theories, such as Lamport's concurrency [Lamp78], we claim that the invariant "absolute concurrency" of distributed events does not exist practically or more correctly, cannot be cost-effectively, infallibly detected (measured) in existing implementations of real communications systems, where the concurrency is defined in terms of **physical time**. Two implications are: i) events at different locations cannot be said to have occurred at exactly the same physical time, and ii) we may not be able to tell from external observation which event happened first. Therefore, to specify **and** thoroughly test a communications system, "relative concurrency" rather than real clock-based concurrency is a more practical model of both sequential and concurrent features of the system. For this reason, we do not need to satisfy the strong clock condition [Lamp78] in our model. Also, for conformance testing, particularly test result analysis, the intended order of test events are preselected and the test architecture discussed in Chapter 5 allows us to exclude anomalous behaviors from consideration.

3.1.1. The Concept of Relative Clock

First, we define relative clock, and then define relative concurrency of distributed events.

Definition 3.1: A local event is an interaction (sending or receiving a message) with a user through a SAP associated with the user, in an instant of physical time. The local event is observable by the user associated with it.

Notation 3.1: **SAP_ID.m** denotes a local event, where i) SAP_ID is the ID of the SAP where the event occurs (here, it is assumed that each SAP has an unique number as its ID); ii) "." denotes the action performed by the users though the SAP. Thus, in the model , "!" is used instead of "." to mean that a user **sends** a message; "?" is used instead of "." to mean that a user **receives** a message; and iii) *m* is the type of message sent or received by the user through the SAP. The SAP_ID can be also considered as the ID of the user when the user is accessing the services through the SAP.

Definition 3.2: A **conversation** is a sequence of events distributed over two or more SAPs during a given period of time.

Notation 3.2: L_i denotes the set of all local events observed at SAPi which occur in a given conversation.

In order to be able to recognize relatively concurrent events, we defined below a means of implementing global relative clocks. Also, this allow us to recognize spontaneous events.

Definition 3.3: A **local relative clock** *Ci* is a function which assigns a natural number Ci(SAPi.*a*) $\in$ Ti to any event SAPi.*a* $\in$ Li , where Ti is the set of natural numbers. Ci(SAPi.a) is said to be the **relative time** associated with event SAPi.*a*.

As we mentioned in Chapter 2, the identification of collisions caused by spontaneous events (transitions) is very important for detecting erroneous spontaneous events [Boch90b, BoDsZh89]. IN Chapter 5, an example is given to show that spontaneous events, if not detected, could be interpreted incorrectly in traces (test results). Formally, we define a spontaneous event as follows:

Definition 3.4: A spontaneous event is an event, say SAPi?M, which occurs but has not been caused by any external stimulus. Provided that the system has a local relative clock, *M* received at SAPi will not be given a timestamp.

Conceptually, there is no overlap between the "relative clock" and "temporal logic" which is introduced in Section 2.1.3. However, the "temporal" or "temporal interval" information over which (logic) assertions are made in temporal (interval) logic (specifications) can be identified based on relative clock (of course can be also based on real clock ideally).

To represent the relatively concurrent relations between any two events, a relation of local events is defined by satisfying the following relative clock conditions (Lamport's logical clock conditions). Also, the relation between a spontaneous event and other events should satisfy these conditions (see also **implementation rules** below).

Relative clock conditions:

C1: If SAPi.a and SAPi.b are events occurring at SAPi, and SAPi.a comes before SAPi.b, then Ci(SAPi.a) < Ci(SAPi.b);

C2: If SAPi.a is the sending of a message at SAPi and SAPj.b is the receipt of that message at SAPj, then Ci(SAPi.a) < Cj(SAPj.b).

3.1.2. Implementation Rules for the Relative Clock

To implement the (global) relative clock in terms of the relative clock conditions, a set of **Implementation Rules (IRs)** (i.e. a distributed algorithm) is needed for synchronizing the (local) relative clocks.

In addition to the two IRs given in [Lamp78], we provide three more rules to take into account our model (see Chapter 5) of communications software. These rules are:

- **IR1:** Each SAP is assigned a counter as the relative clock of all user-observable events occurring at that SAP, the counters at all local SAPs of a communications system are initialized to the natural number "1" once a new *conversation* starts,

- **IR2:** Each user *i* increments C_i by "1" once a new (local) event occurs at the SAPi .

- **IR3:** i) If event *SAPi a* is the sending of a message *a* through SAPi, then the message *a* contains a local relative **timestamp T(a)**:= C_i(SAPi.a) (for synchronization), ii) Upon receiving a message *n* with T(n) from a remote user, the user associated with SAPj sets $C_j := 1 + \text{Max}\{C_j, T(n)\}$,

- **IR4:** For any message *s* which comes spontaneously from the system under specification (SUS) will not have a timestamp. When it is observed at SAPi, that SAP will set Ti(s) to 0.

- **IR5:** Immediately after a modification of a relative clock according to one of the above IRs, **the *relative time* of the event is defined to be the value of the relative clock located at the SAP where it occurs** (see the example in Section 5.1.3). Also, we assume that there is a mechanism (see Appendix A) for each SAP to record each event observed at the SAP and its relative time.

It should be noticed that the counters (relative clocks) are either implicit for specification or explicit for testing (discussed in Chapter 5); that is, the IRs may also be used in order to specify the system with virtual time. In other words, the clocks are not necessarily a real part

of the system. Note also that both the relative clock-based partial order of (local) events and the timestamp for each local event are unique based on the above set of implementation rules.

3.1.3. Relative Concurrency

In terms of the concept of relative clock, we define the relative concurrency as follows:

Definition 3.5: Two local events i.*a*, j.*b* are ***relatively concurrent*** if and only if $C_i(i.a) = C_j(j.b)$. The corresponding relation is denoted ***relative concurrency***.

An intuitive interpretation of the relative concurrency of events *i.a* and *j.b* can be described as follows:

i) **globally**, the events *i.a* and *j.b* occur at two communications locations, SAPi and SAPj, respectively, independently of each other or without having a sender-receiver relationship, and

ii) **locally**, the relative time period between the event *i.a* and the first event at SAPi is equal to the relative time period between event *j.b* and the first event at SAPj within a conversation between the SAPs.

Noted that we slightly constrain Lamport's concept of "concurrency' in Definition 3.5; namely, we require that relatively concurrent events have the same relative time. The reasons are:

i) Lamport's definition probably did not intend to require global concurrency to satisfy local restrictions. Specifically, consecutive events at a SAP may be regarded without relative timestamps as "equally-concurrent" from his definition if a remote event is taken as a

referee. However, this is somewhat counterintuitive since Lamport gives different timestamps to consecutive events at one SAP;

ii) Partially ordered local events based on his definition may not be easily represented in most specification languages;

iii) Our definition implicitly identifies and reflects changes in "global state" [ChLa85] of the system even though we do not use the concept "global state" as a basis in this thesis. Therefore, our definition is adequate to specify and further validate Lamport's concurrent scenarios systematically as in Sections 3.2, 3.3 and Chapter 5.

iv) Definition 3.5 is based on Lamport's interpretation of the nature of distributed concurrent events [Lamp78].

We show by a simple example (see Figure 3.1) that Definition 3.5 facilitates specification and analysis (e.g., test result analysis) of concurrent scenarios such as collision scenarios (also, see more discussion in Section 3.3) :

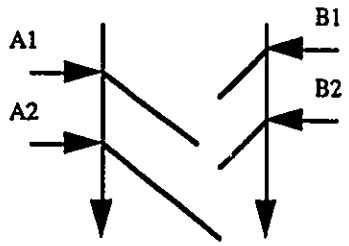

Figure 3.1: A Relatively Concurrent Scenario

Based on Lamport's concept of concurrency, A1, B1 can be defined as logically concurrent and A1 and B2 can also be defined as logically concurrent. However, in order to analyze system behaviours, the "concurrency" concept should identify and reflect (global) state

changes of the system. Relatively concurrent events A1, B1 identify the global state in which A1 and B1 just occurred, independently of each other. A1, B2 could also have identified a global state of the system. However, for specification of legal scenarios of a system and further testing the system based on the specification (see Chapter 5), we need only one instance of a global state; that is, for the scenario in Figure 3.1, either a global state defined by <A1, B1> or a global state defined by <A1, B2>, but never by both of them. This is because it is counterintuitive and not necessary to define the "global state" in which A1, B1 are logically concurrent and at the same time A1, B2 are logically concurrent. This violates our understanding that the occurrences of B2 brings about a change in global state. Therefore, only the events with the same relative time value are defined to be relatively concurrent (Definition 3.5) in order to yield a meaningful and natural definition of global states as a basis for specification and analysis of the system. The work in the rest of this Chapter and Chapter 4 will show that specifications based on this concept are sufficient to specify allowed concurrent scenarios in a language called ETAL, and in Chapter 5, the specification based on this concept is shown to be directly useful for test result analysis including time-ordering information.

With respect to OSI framework, we consider an OSI transport service example to illustrate the concept of "relative concurrency". Two events A!TCONreq and B!TCONreq are said to take part in a relatively concurrent scenario if and only if they happen at same relative time. Such a scenario can be expressed with an event-tuple : <A!TCONreq, B!TCONreq>

Note that the events A!TCONreq and B!TCONreq do not necessarily happen at exactly the same physical time (if it can be measured), but they must happen independently of each other. However, this does not exclude the extreme case of real (physical) concurrency.

3.2. Global Events Model

3.2.1. The Model

In this section, a model for the interpretation of relatively concurrent scenarios is described. As discussed in Section 2.2.1, none of existing models are expressive enough to interpret concurrent feature of a purely distributed system. For example, models based on the interleaving semantics have been used to interpret concurrent behaviours (e.g., in a LOTOS expression of [Boch90b]) of a communications (distributed) system. However, these models fail to completely interpret either of physical time based and relative time based distributed concurrency. Also, it may be either too costly to provide or not practical* enough to enforce a means of imposing a total order on distributed local events based on real clock or relative clock, respectively. Accordingly, we define a non-interleaving model, called a global events model, based on "global relative clock" to interpret the relative concurrency.

Definition 3.6: A **global events model** is defined as a 4-tuple GE=<L,T,C,G> where

- $L = \cup\{L_i \mid i = 1..n\}$, is the set of all local events occurred in one conversation.
- $T = \cup\{T_i \mid i = 1..n\}$, is the set of natural numbers.
- Global Relative Clock C is a function with domain L and range T such that C assigns a number, $C(l) = Ci(l) \in T$, to any event $l \in Li$ as the ***global relative time*** of l based on C.
- G is the set of all global events in one conversation, where any global event $E \in G$ is a nonempty subset of L, i.e. $L \supseteq E$; for any local event $e_i, e_j \in E$, $C(e_i) = C(e_j)$; the global relative time of E, $C(E) = C(e)$ for any $e \in E$.

---

* Using "Pi < Pj" as a means to enforce a total order [Lamp78] might not be desirable if a pure distributed system is concerned.

The scenario in Figure 1.1 can be represented in our model as GE = <L, T, C, G>, where :

L_{sap1}= {SAP1!a, SAP1!c, SAP1?b'}; L_{sap2}= {SAP2!b, SAP2?a', SAP2?c'};

L ={SAP1!a, SAP1!c, SAP1?b', SAP2!b, SAP2?a', SAP2?c'};

T_{sap1}= {1, 2, 3}; T_{sap2}= {1, 2, 3}; T = {1, 2, 3};

G = {{SAP1!a, SAP2!b}, {SAP1!c, SAP2?a'}, {SAP1?b', SAP2?c'}} = { E_1, E_2, E_3 };

$C(E_1)$ =C(SAP1!a) =C(SAP2!b) = 1; ... .

Note that the definition of global events is different from the one in [HaWe88]. The purpose of our definition is that the formal specification language ETAL can be defined based on relative clock for specifying concurrent features of communications systems. Also, our definition facilitates specification-based testing activities. More discussion of testability benefits can be found in Section 5.3.

Lemma 3.1: All local events in a global event are relatively concurrent.

Proof: The proof is obvious from Definitions 3.4 and 3.5.

Intuitively, there are "relative snapshots" which "record" the global patterns of all local events which occur at different given global relative time in a given conversation. The "photo" made by a snapshot at a given relative time is a global event. In this thesis, a global event is expressed in an "event-tuple".

3.2.2. The Specification Method "Event-Tuple" based on GE Model

In this section, based on GE model, we develop a specification method "event-tuple" to represent the concurrent features of communications (distributed) systems. In the next

Chapter, we will presented a language, namely extended trace assertion language to specify communications (distributed) system with this method.

Notation 3.3: Let "." denotes concatenation of consecutive event-tuples in a global events trace.

Notation 3.4: An **event-tuple** denotes a global event. Based on the global events model, it expresses a global event as follows:

i) a global event consists of relatively concurrent events

< SAP1.a, SAP2.b, ..., SAPn.m > :

This indicates that the relative time of occurrences of these local events in "< >" is exactly the same based on global relative clock. In other words, the events are relatively concurrent.

ii) sequential global events w.r.t. sequential local events

case1: < SAP1.a, SAP2. _ , ... , SAPn. _ > . < SAP1.a', SAP2. _ , ... , SAPn. _ >

where, "_" means no event observed; it indicates that event SAP1.*a* happened before event SAP1.*a'*, that is, the time of occurrence of SAP1.*a* is earlier than the time of occurrence of SAP1.a', based on global relative clock.

case2: < SAP1.a, SAP2. _ , ... , SAPn. _ > . < SAP1._, SAP2.b , ... , SAPn. _ >

where, "_" means no event observed; it indicates that event SAP1.*a* happens before event SAP2.*b*, that is, the time of occurrence of SAP1.*a* is earlier than the time of occurrence of SAP2.*b*, based on global relative clock.

Note: case 1 **specifies** the case of sequential local event occurrences and case 2 **specifies** the case of sequential global event occurrences.

Theorem 3.2: The order of the global events in one conversation is total based on GE model.

Proof: The proof can be obtained by proving the satisfaction of the condition: *each global event must be assigned an unique natural number as its relative time.*

It is obvious that each global event must be assigned a natural number (Definition 3.6). To prove that the number is unique, we assume that there exist two global events which have the same global relative time in a system of two distributed entities (two SAPs), and we show a contradiction. There are two cases:

case1: <SAPx._,SAPy.b> and <SAPx.a,SAPy._>

case2: <SAPx.a,SAPy._> and <SAPx.a',SAPy._>

For case 1: Based on the above assumption,

$C(<SAPx._,SAPy.b>) = C(<SAPx.a,SAPy._>)$ and using Definition 3.6, we have

$C_y(SAPy.b) = C(<SAPx._,SAPy.b>) = C(<SAPx.a,SAPy._>) = C_x(SAPx.a)$

Based on Definition 3.6, SAPx.a and SAPy.b should be in one global event; That is <SAPx.a,SAPy.b>; this contradicts the Definition 3.6.

For case 2: Since SAPx.a and SAPx.a' happen at the same location SAPx,

$C_x(SAPx.a) \neq C_x(SAPx.a')$ by the implementation rule IR1,

thus from Definition 3.6,

$C(<SAPx.a,SAPy._>) = C_x(SAPx.a) \neq C_x(SAPx.a') = C(<SAPx.a',SAPy._>)$;

this also contradicts the assumption.

By repeating the above procedure, we can derive a sequence of contradictions of the assumption that there exist more than two global events which have the same global relative time.

Lemma 3.3: Each local event must appear once and only once in the sequence of global events in one conversation.

Proof: From Definition 3.3, each local event is assigned a natural number based on local relative clock; From Definition 3.6, the natural number is also its global relative time, so it must appear once in a global event. If we assume that a local event appears more than once in the sequence, then all global events which contain the event must have the same global relative time (Theorem 3.2). This contradicts the assumption; that is , each local event must appear only once in the sequence.

From the above theorem and lemmas, we can see that the GE model can represent and interpret both sequential and concurrent features of a communications system adequately. Also, the relative clock based specification technique is simpler and more applicable than either physical clock based or interleaved semantics based techniques in capturing the nature of concurrency. A testability-directed ETAL specification based on relative clock to the OSI transport service is presented in Section 4.2.

3.3. Formal Definition of Collisions

In this section, we apply the relative clock based specification technique to one of most interesting concurrent scenarios - collision - to show its expressive power.

The concept of collisions in communications protocols has been informally, but not formally, discussed in other work [Boch90b, Gotz90]. In this section, we discuss the nature of

collisions, then we present a formal definition of collisions based on our global events model.

3.3.1. The Nature of Collisions

Collisions are those scenarios in which more than one process at different SAPs initiate actions independently of each other. In these scenarios, one or more of these processes may not receive the expected responses. Collision scenarios are also called "cross overs" (of internal interactions) from an internal system structure point of view. Typical collision scenarios are shown in Figures 3.1(a) and (b).

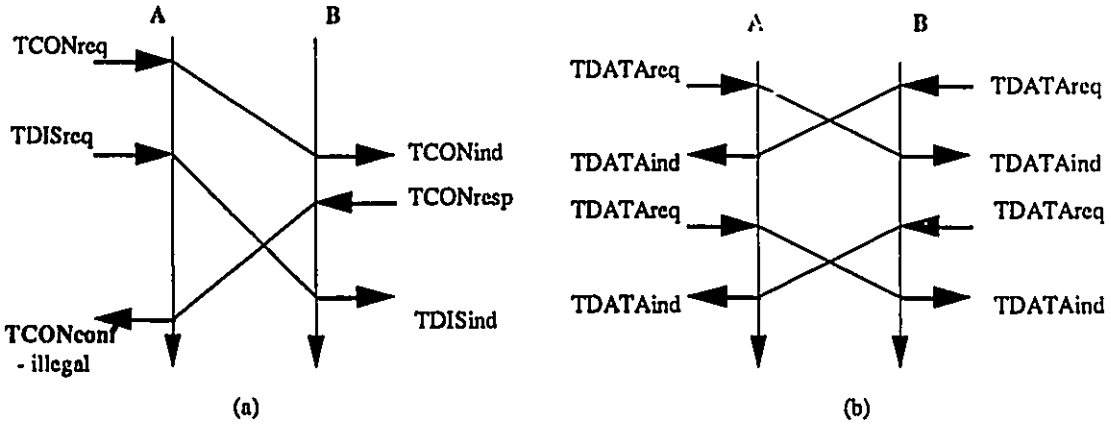

Figure 3.2: Typical Collision Scenarios

3.3.2. Formal Definition of Collisions

There are several reasons why we have to take collisions into account in the specification of a distributed system: i) each user in a purely distributed system should have absolutely equal right of access the system. This is desirable from the (system) service point of view; ii) Because there is no sharing of real clock and memory, and no priority assigned to a single entity associated with the SAPs in a communications (distributed) system, it is extremely difficult to design an efficient protocol which can avoid collisions. Therefore we need to characterize collisions to enable protocols to be designed with the capability to fairly handle

collisions; iii) for the sake of the completeness of the testing of the distributed interface, test cases handling these extreme cases must be easily derivable from the specifications.

As mentioned in [Boch90b], there are two types of collisions: one will cause some problems while the other will not. We detail this point with a practical assumption that there exists a definition on the legality of the order of event sequences (traces) of a systems (for instance, [ISO8072]). We then define the unacceptable collision and acceptable collision based on the local legality and the global events model, respectively. Moreover, we further assume that the order of occurrences of events at any one SAP can be determined based on real (local) clock. Therefore the order of the events at one SAP can be totally ordered.

Definition 3.7: Legal (order of a trace) at SAPi is defined as {a1, a2, ... , ak} | $k \geq 2$ and the sequence a1, a2, ... , ak is allowed at SAPi by the specifications.

Notation 3.5: Order(L,e_m,e_n) denotes the order of local events sequence L.e_m, L.e_{m+1} , ..., L.e_n at SAP L where m, n are integer, such that n>m>0.

Definition 3.8: An **unacceptable collision scenario** consists of a sequence of global events: <SAPx.a_1, SAPy.b_1> .<SAPx.a_2, SAPy.b_2> .... <SAPx.a_k, SAPy.b_k>, such that the sequence satisfies the conditions: i) Order(SAPx, a_1, a_{k-1}) and Order(SAPy, b_1, b_{k-1}) are legal; ii) either Order(SAPx, a_1, a_k) or Order(SAPy, b_1, b_k) is illegal; and iii) there exists at least one global event, say < SAPx.a_m, SAPy.b_n>, for any m and n from 1 to k, provided that both SAPx.a_m and SAPy.b_n are nonempty.

Definition 3.9: An **acceptable collision scenario** consists of a sequence of global events: <SAPx.a_1, SAPy.b_1> .<SAPx.a_2, SAPy.b_2> .... <SAPx.a_k, SAPy.b_k>, such that the sequence satisfies the condition: i) for all p|1<p≤k, Order(SAPx, a_1, a_p) and Order(SAPy, b_1,

b_p) are legal, and ii) there exists at least one global event, say < $SAPx.a_m$, $SAPy.b_n$>, for any m and n from 1 to k, provided that both $SAPx.a_m$ and $SAPy.b_n$ are nonempty.

The collision scenario in Figure 3.2(b) is a simple case of an acceptable collision which does not affect the service (data transfer) reached by the system.

Theorem 3.4: A collision scenario is a sequence of event-tuples containing at least one tuple of relatively concurrent events.

Proof: This is obvious from Definitions 3.8, 3.9 and Theorem 3.1.

It is worthwhile to point out that the concept of relative concurrency is sufficient to reflect the characteristics of collision behaviour precisely from Theorem 3.4. That is, the relative concurrency rather than the traditional (physical time) concurrency stresses the physical meaning of collisions.

The collision scenario in Figure 3.2(a) is an unacceptable collision. The representations of global trace and local traces of this scenario are as follows:

Global (events) Trace:

<A!TCONreq, B. _ > . <A!TDISreq, B?TCONind> .< A. _ , B!TCONresp>.<A?TCONconf, B?TDISind>

Local (events) Traces:

at A: A!TCONreq . A!TDISreq . A?TCONconf -- locally illegal Order;

at B: B!TCONind, B!TCONresp, B?TDISind -- locally legal Order

A formal specification of this collision scenario based on the trace assertions in ETAL is presented in Appendix B.

It would be desirable to consider how to handle unacceptable collisions at the time of specification. However, this is not easy to propose a general method to handle the collision. More discussion on this issue can be found in Sections 4.2.2 and Section 6.3.

CHAPTER 4

EXTENDED TRACE ASSERTION LANGUAGE AND APPLICATIONS

4.1. The Extended Trace Assertion Language (ETAL)

We have briefly introduced Parnas's trace assertion language (TAL) in Section 2.1.2. One of the foundations of the language and the method is the ordering relation among (local) events. This relation is automatically determined if the system to be specified by TAL is non-distributed. However, to specify a **geographically distributed** system such as a communications system, a formal model should first be established to interpret the ordering relation among *distributed* events. Therefore, to describe the global and concurrent aspects of the services provided by a communications (distributed) system and for subsequent testing/verification of the system, the trace assertion language is extended in two ways based on the global events (GE) model (introduced in Section 3.2) . These extensions are first to **event-tuples**, and secondly, to include **temporal assertions.**

4.1.1. Event-tuple Extension

Although the trace assertion language (TAL) discussed in Section 2.1.2 is capable of representing scenarios and specifying the legality of a local service scenario, it is not very suitable for describing the concurrent behaviours such as collisions based on a global view of the distributed system. This is since TAL specifies the system only in terms of sequentially observable local events. Therefore, we first extend TAL by employing the global event

representation introduced in Section 3.2, instead of a local event representation, to capture the global view of events and to explicitly describe concurrent behaviours.

A global event is represented by an event tuple. An example is <A!TCONreq, B!CONreq>, which describes that two events A!TCONreq and B!CONreq are said to take part in a relatively concurrent scenario if and only if they happen at the same relative time. The relationships among local events and global events, and the format of event-tuples are described in detail in Section 3.2. A detailed example is given in Section 4.2 for illustrating how trace assertions expressed on global events can specify concurrent aspects.

4.1.2. Temporal Assertions Extension

As discussed in Section 2.1.3, event based temporal assertions are capable of easily representing liveness properties of both global (system) and local (unit) views of services provided by a co.. ·unications (distributed) system. However, temporal assertions are not suitable for representing safety properties. On the other hand, the specification of liveness properties is very useful for verification purposes (see Section 6.1 for more detail), and the original trace assertion language is not suitable for describing liveness properties (although it is powerful enough to describe safety properties). Our approach is to extend TAL by adding simple temporal operators.

For example, to consider the concurrent data transfer service in the OSI transport layer. It is quite difficult use trace assertions to explicitly specify liveness properties such as

> A TDTreq in a sender side should be eventually delivered and presented as the corresponding TDTind in the receiver side in all possible scenarios.

This difficulty is due to that the legality of a trace defined in a trace assertion does not refer to future but the past.

A temporal assertion however easily and explicitly expresses such a future-oriented property:

[] (SAP1!TCONreq -> <> SAP2?TCONind)

This asserts that a connection request at SAP1 will result in a connection indication appearing at SAP2 sometime in the future. This is quite a concise specification, which is more useful than the trace assertion method for verification purposes.

4.1.3 Observations on the Extensions of TAL (ETAL)

Because of the complementariness of temporal logic and trace assertions, the idea of combining the two languages is quite natural. Both are property-oriented and based on the same mathematical foundation (first order logic).

The **Extended Trace Assertion Language** (ETAL), is obtained from TAL by incorporating both extensions described above, namely event-tuples and temporal assertions, based on the global events model. Not only can ETAL represent concurrent scenarios but it also can specify both liveness and safety properties according to a global view of services provided by a communications (distributed) system. Thus, the extended trace language is a scenario/property-oriented language based on a set of assertions about external (observable) behavior of a system. With respect to a "solely abstract" specification capability [BaPa78, Hoff85, Gotz90], none of the FDTs [FORTE] (perhaps due to the intended purpose of these FDTs) are well suited, particularly for describing totally abstract scenarios/properties of the services of a totally distributed system. In the next section, the format of the extended trace specification is described in detail.

The basic idea behind the extended trace assertion language is "*specifications are predicates*", similar to the idea "programs are predicates" made by Hoare [Hoar85]. In Chapter 5 and Section 6.1, we will discuss a more important applications of the language derived from this idea, namely to test result analysis and to verification.

4.2. Applications of ETAL to Transport Example

In this section, we show how to use the extended trace assertion language to specify the Transport Service (TS) introduced in Section 2.1. A complete ETAL specification of both local and global transport services is given in Appendix B for reference for the discussion in this section.

4.2.1. The Format of ETAL in the Application

The format of a specification in ETAL is similar to that in TAL (recall Section 2.1.1). The only differences are that the syntax of the expression of events is changed by using event-tuples in specification of global services and that temporal assertions are added into the semantics part. For sake of simplification, only some of parameters are presented in the specification since what are stressed in this thesis is on the relative timing (ordering) relations among the events in traces and those parameters are more interesting when exploring such ordering relations. Also, to focus on canonical traces and equivalence functions which are more interesting in illustrating our techniques for specification and testing, the sections related to the output variables and their values in the specification are omitted. Thus, an ETAL specification consists of:

SYNTAX:

1> Access Program ,

SEMANTICS:

2> Canonical Traces, /* event-tuple expression */

3> Auxiliary Functions, /* optional */,

4> Equivalences,

5> Temporal Assertions. /* our addition*/;

Table 4.1: The Format of ETAL

The definition of function of each part in the format above is same as that of each corresponding part in TAL.

In order to express the canonical unacceptable collision traces (Definition 3.7), the definitions of canonical traces in TAL are modified as follows:

A " ©T " denotes an unacceptable collision canonical trace. (see also Section 4.2.3)

A "T" denotes a sequential or an acceptable collision canonical trace.

A "%T" denotes an erroneous canonical trace.

An "E" stands for any global event.

An "e" stands for any local event.

Note that none of "©T"s or "T"s above could be overlapped with any "%T". However, one of "©T"s may be overlapped with one of "T"s. Also, a collision scenario corresponds (or part of) a canonical trace in the specification.

4.2.2. The Specification of OSI Transport Service

The transport service specification specifies OSI transport service (TS) in terms of the observable behavior, i.e., observable events, of the service provider in terms of its standardized definition [ISO8072]. Particularly, it includes the specification of acceptable and unacceptable collision scenarios when the services defined in the standard are provided. Notice that some of those scenarios such as collision scenarios are not defined in the standard. In this specification, we also assume that the service is provided based on class 0 transport protocol.

As discussed in previous chapters, specifications play an important role in test result analysis for conformance testing. Therefore the specification of the services includes two parts for test

results analysis. In the Part I of Appendix B, the local services of TS are specified in terms of a set of assertions on the legal and illegal local event traces; in the Part II of Appendix B, the global services defined in the standard are specified in terms of a set of assertions on the globally legal and sequential global event traces. Moreover, the global collision scenarios are also specified in terms of our definitions of acceptable and unacceptable collisions (see Section 3.3).

The specification of global services is represented in terms of two-SAPs (users) of the transport service. The style of the specification is based on the service users point of view (recall the discussion in the beginning of Chapter 3). In the temporal assertions part of the specification, we just employ the basic standard temporal operators described in Section 2.3. We use binary event-tuples to describe the global view of the two-users services in both trace and temporal assertions. This can be extended to n-tuples to describe multiple connections.

In the specification, one type of access program corresponds to one type of primitive defined in the standard. The notations of the access programs are as follows:

TCONreq denotes the primitive "T_CONNECTrequest".
TCONind denotes the primitive "T_CONNECTindication".
TCONresp denotes the primitive "T_CONNECTresponse".
TCONconf denotes the primitive "T_CONNECTconfirm".
TDTreq denotes the primitive "T_DATArequest".
TDTind denotes the primitive "T_DATAindication".
TDISreq denotes the primitive "T_DISCONNECTrequest".
TDISreq denotes the primitive "T_DISCONNECTindication".

The notations of the parameters of each TS primitive are given in the Table 3 of the standard. The representation of an event is defined to Section 3.2.

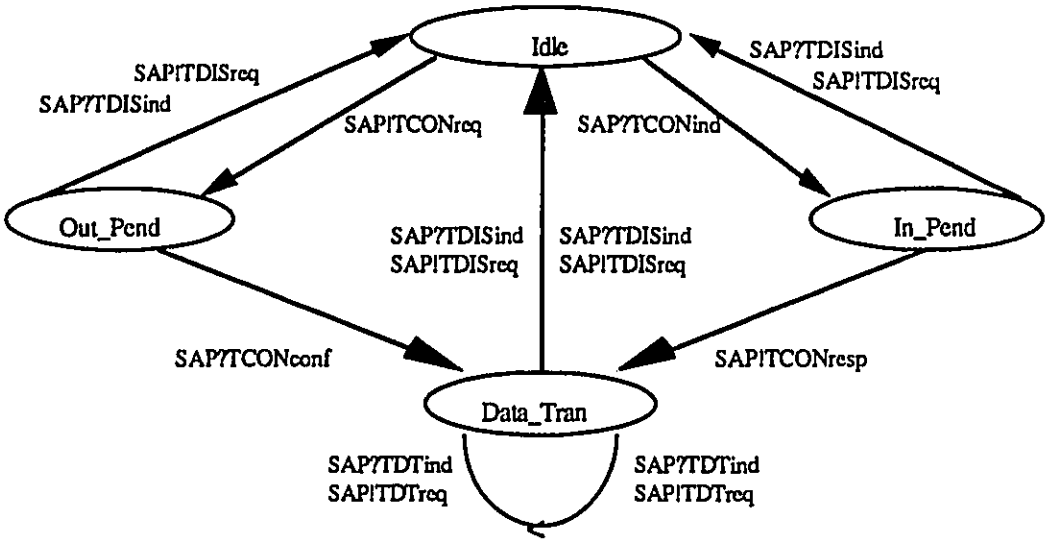

Figure 4.1: FSM Representation of Local TS

Figure 4.1 shows the local transport services in FSM form. For instance, one of locally legal scenarios corresponding Figure 4.1 is as follows:

SAP!*TCONreq* . SAP?*TCONconf* .*K*

where subtrace K can be any composition of SAP?*TDTind* and SAP!*TDTreq*.

Note: In fact, if parameters of primitives are considered, the number of events in K must be finite.

In the Part I of Appendix B, The (legal) canonical trace corresponding to the above legal scenario (canonical trace #6) is :

T= Tp.Tq

$\wedge$(Tp = SAP!TCONreq.SAP?TCONconf)

$\wedge$(Tq= SAP?TDTind V Tq= SAP!TDTreq V Tq= Tq.SAP?TDTind V Tq= Tq.SAP!TDTreq)

where the recursive expressions, Tq= Tq.SAP?TDTind and Tq= Tq.SAP!TDTreq, are used to represent the expression "*K*" in that scenario.

Moreover, since we can regard the protocol entity as a kind of "service providers" which provides protocol through Usap (Upper SAP) and Lsap (Lower SAP), the protocol specification can also be written in ETAL in much the same manner as we did for services.

4.2.3 Discussion on Specifying Unacceptable Collisions in ETAL

In Part II of the Appendix B, different from Part I, we explicitly specify the collision scenarios as one of contributions in this thesis. The sequential scenarios and acceptable collision scenarios are specified in terms of their definitions in the TS standard. However, the unacceptable collisions are not defined in the standard since they are actually not global services. One of the reasons why we specify the unacceptable collisions in the specification is that test results conclusions can be correctly drawn even though unacceptable collisions occur during testing of a service provider (see also Section 3.3 and Chapter 5).

Both specification and validation of complete unacceptable collision behaviours are meaningless since such behaviours are actually not global services. This is probably the reason why unacceptable collision scenarios are not defined in the standard. Therefore, the specification of the unacceptable collisions presented in this thesis is based only on our understanding and definition of unacceptable collisions. As shown in Part II of Appendix B, we specify an unacceptable collision scenario in terms of an assertion on the future of the scenario based on part of the scenario. That is, on one hand, we do not need a specification for the whole unacceptable collision scenario. In practice, there does not exist a definition of a "whole" unacceptable collision scenario. On the other hand, we should, in the specification, express the fact that both the service users and provider did nothing wrong within any unacceptable

collision scenario. Therefore we specify these scenarios as a different kind of canonical trace from illegal canonical traces (%T).

In the specification, "©T" is used to express the assertion on a canonical unacceptable collision scenario. A ©T asserts that some error (i.e. an unacceptable collision) will eventually occur in the future if last event of this canonical unacceptable collision trace occurs; it also asserts that the trace before the event must be legal in the past. We claim that this method is not only meaningful in specification theory but practical in the applications of the specification (see Section 5.2).

We now take one of canonical unacceptable collision traces, canonical trace#12, from the Part II to illustrate the specification technique discussed above.

©T = <A!TCONreq, B. _ >.< A!TDISreq, B?TCONind>.< A. _ ,B!TCONresp>

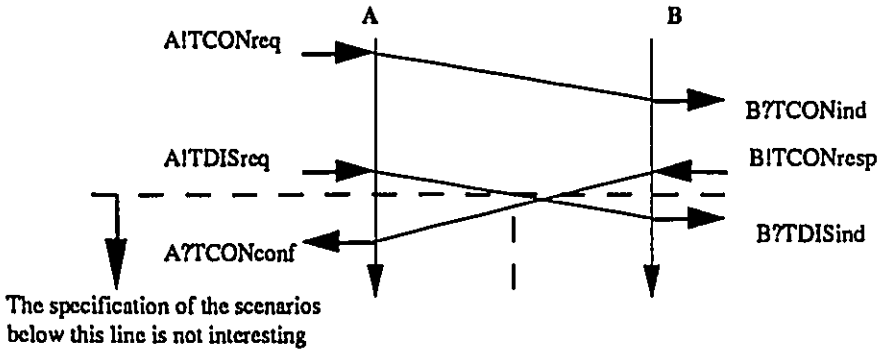

Figure 4.2: An Unacceptable Collision Scenario and its Specification

Figure 4.2 shows the time-sequence diagram interpretation for this trace based on the global event model. As shown in Figure 4.2, the trace

<A!TCONreq, B. _ >.< A!TDISreq, B?TCONind>

will become an unacceptable collision scenario once the last event in the canonical trace, "< _ ,B!TCONresp>", occurs. This because the A!TCONconf will be eventually received by the user A, which, however, is locally illegal at SAP A. That is, the trace

<A!TCONreq, _ >.< A!TDISreq, B?TCONind>.< _ ,B!TCONresp>

asserts that the error (ie,. the illegal event A?TCONconf) caused by the collision will eventually occur.

Note that we do not need to specify the scenario after event "< A. _ ,B!TCONresp>" in the canonical trace since the error (i.e. the collision) caused by the event will eventually occur and that scenario is not part of the service. Also, note that the trace before the event "<A. _ , B!TCONresp>" should be specified as another legal canonical trace (#11) in the specification. The legal trace expresses that both the service users and provider work correctly even though the global scenario is unacceptable. The way for specifying unacceptable collision scenario in ETAL described above precisely captures the nature of unacceptable collision scenarios. In Section 5.2, we will show that such specifications are very useful for test result analysis.

It is worthwhile to mention that a specification of a whole unacceptable collision scenario is meaningful if we propose and specify how to handle the unacceptable collision. In Chapter 6, some discussions on the issue of handling collision are presented. The rest of this thesis shows that the specification of unacceptable collision scenarios presented in Appendix B is precise enough for illustrating our contributions to specification and test result analysis methods which are the theme of this thesis.

CHAPTER 5

A NEW APPROACH FOR TEST RESULT ANALYSIS AND TESTABILITY-DIRECTED SPECIFICATIONS

The testing procedure for conformance testing of communications protocols and services [UrPr86, Rayn87, BoDsZh89] usually includes two major steps: test generation and test result analysis (recall Section 2.1.5). In this chapter, we present a relative clock-based approach for test results analysis including time-ordering information based on global events model. In this approach, relative clocks are implemented in a testing architecture to test the conformance of System Under Test (SUT) to its corresponding relative clock-based specification. Finally, we will give a study on testability-directed specifications and show that a relative clock based extended trace assertion specification is testability-directed.

5.1. The New Approach for Test Result Analysis

Basically, there are two main tasks to be accomplished in test result analysis using trace analysis: i) correctly recording traces including time-ordering information, and ii) analyzing the recorded traces with respect to the corresponding timed reference specifications to obtain test conclusions. However, there is no practical technique to accomplish the first task, and how to derive reference specifications for second task has not been well specified in previous research (recall Section 2.2.2). In this section, we present a new, specification-based test result analysis approach for accomplishing the above two tasks.

5.1.1. The Steps and Testing Architecture for the Approach

In our approach, there are three steps involved in the test results analysis:

i) check legality of ordering of all local traces at each SAP against the (local) formal specification of that SAP. Once all the local traces are shown to conform to the corresponding local specification, the second step is begun.

ii) use the "trace merger" module in the global analyzer merges the locally timestamped events in local traces at all SAPs to form a global event sequence (list).

iii) use global analyzer to check whether or not the relative ordering relations among those events in the sequence (list) are legal with respect to the ETAL specification of the global view of the system. Based on this checking, test verdicts are assigned and test results are reported.

The basic idea that test result analysis be divided into the three steps in our approach is not entirely novel. Nevertheless, the practical difference is that the timestamp for each local event is based on relative clocks rather than real clocks in our approach. We now focus on the description of the latter steps with more detailed discussion on the use of relative clock-based specifications in testing, particularly in test result analysis.

Implementation testing activities based on global/local service specifications are denoted **service-directed conformance testing** (see also Section 2.2.2). Global / local service specification are defined to Section 2.1.1. The examples of global/local service specification can be found in Appendix B.

Figure 5.1 shows a **service-directed external testing architecture** based on a coordinated method [Rayn87, BoDsZh89] (ignore the message shown in the figure for now). The system (service provider) consisting of protocol entities A and B is the system under test (SUT). The "Global Tester & Analyzer" sends test events to the SUT and analyzes the events received from the SUT only through SAPs A and B, remotely. Notice that the test events based on this testing architecture are specified in terms of TS-PDUs. Since we explicitly employ relative clock for service-directed conformance testing, we also identify the locations of the relative clocks, Ca and Cb in the testing architecture.

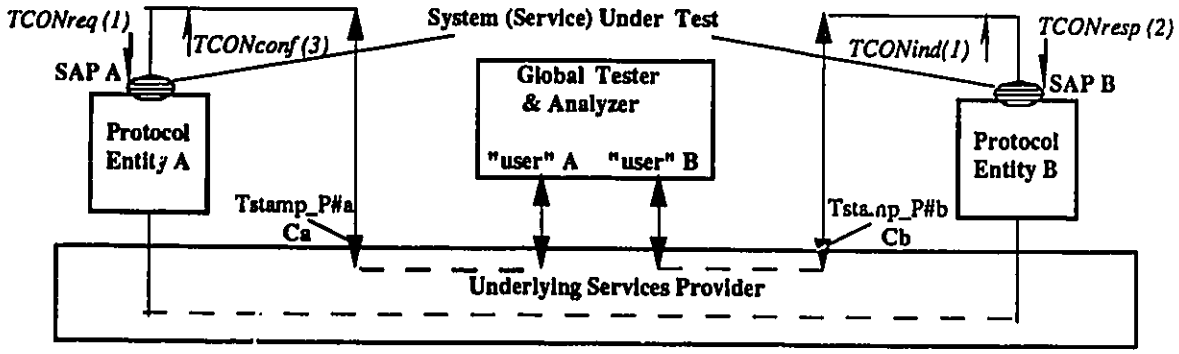

Figure 5.1: Service-Directed External Testing Architecture

Although the cost of adding relative clocks into SUT is quite low, one may argue that it may not be practical if the SUT already exists. In this case, we specify virtual relative clocks Ca and Cb which virtually located at the SAPs A and B in the ETAL specification of the services. We may think of these clocks as subprocesses of user A (tester) and user B (test responder). The local points of implementation of the clocks are called timestamp points, Tstamp_P#a and Tstamp_P#b (see Figure 5.1), respectively. The IRs (Implementation Rules) for implementing C_a and C_b are same as those in Section 3.1.2.

In addition, we need the assumption that each entity in the SUT can correctly deliver a T(m) (see IR3(i)). That is, the T(m) can always be "attached" to the corresponding PDUs

(De/Encoding of *m*) within the layers through which the message *m* is delivered (from the sending user to the receiving user). Note that the notation of a message could be changed; e.g., a message expressed by notation *TCONresp*, sent at sender's side, will be expressed by notation *TCONconf* at receiver's side of that message.

We assume that the functions of the virtual relative clocks used in the specification are equivalent to their implementations at Tstamp_P#a and Tstamp_P#b. We also assume that there is no nondeterministic interference outside the environment consisting of the service provider and its users.

In a similar way as for the service-directed external testing architecture, we build relative clocks into the coordinated testing architecture for protocol conformance testing. Figure 5.2 shows the relative clock based coordinated testing architecture (ignore the messages shown in the figure for now).

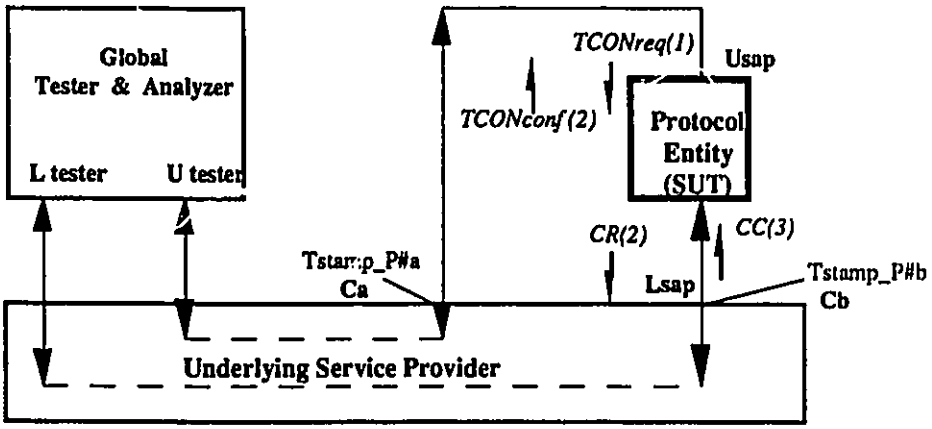

Figure 5.2: Protocol-Directed External Coordinated Testing Architecture

5.1.2. Key Algorithms

As mentioned in the last section, two modules "trace merger" and "test result analyzer" are needed in the global analyzer for accomplishing the tasks in the latter two steps of our test result analysis.

Trace_Merger (TM) Module

The function of the trace merger module in the global analyzer is that it takes observed local events in the received local traces as parameters and outputs a totally ordered global events list based on the global events model (i.e., merges local traces to a global trace).

Based on the definitions of relative clock and event-tuple given in Chapter 3, the algorithm is straightforward. The Trace_Merger Module consists of a set of submodules:

- *n* **e_snapshot** modules, where *n* is the number of SAPs involved in a testing procedure; each e_snapshot stores all events which were observed at SAPi, but not yet processed (see process_event module definition) in a queue, called e_queue#i;

- one **process_event** module which processes each observed event to form a new or modify an existing global event, and then modifies the global_event_list. That is, the Trace_Merger module takes each observed event as a parameter value and outputs a modified global_event_list in ascending order by global relative time. The order of the global events in the list is total (the proof is similar to the proof of Theorem 3.2, given in Section 3.2). Algorithms for TM module have been tested and a detailed pseudo code of the algorithms can be found in Appendix A.

Test Result Analyzer (TRA) Module

The function of TRA in the global analyzer is that it takes a merged trace (i.e., a global_event_list) as a parameter and automatically searches the trace in the trace assertion specification of the SUT to assign verdicts (fail or pass the test) correctly for the test results associated with the trace. The trace is expressed in a global_event_list which was provided by TM module.

The procedure for searching a merged trace in given trace assertions is achieved by a set of mapping procedures, each of which is associated with one global event in the merged trace. Based on concepts in ETAL, the first global event in any given trace must be a canonical trace and any subtrace starting from the first global event should be mapped into a canonical trace. To map the current canonical trace into a new canonical trace in each mapping procedure, TRA module has a variable, called "current_canonical_trace" to record the current canonical trace. Then each *mapping procedure* proceeds as follows:

TRA module records and then removes the global event, say Ea, in the head of given global_event_list once a time. It then searches the equivalence function entitled with "T.Ea ≡" in the equivalence section of the ETAL specification. Once "T.Ea ≡ " is found, TRA module compares any one of conditions in the equivalence function "T.Ea ≡ " with the trace in "current_canonical_trace". There are two cases for the result of a comparison:

i) If a condition and the trace are found matched, then TRA module maps the trace "T.Ea" into a new canonical trace given in the column "equivalences" at the row of the matched condition; meanwhile, the value "current_canonical_trace" is changed to be the new canonical trace.

ii) If there is no match between them until all conditions are compared, then the mapping procedure fails.

During the procedure for searching the merged trace, TRA draws its conclusion in terms of the results of the above comparisons as follows:

i) If all the canonical traces during the mapping procedures after processing all global events in the list are canonical acceptable traces, then the conclusion is that the SUT passes the test.

ii) If a "©T" is reported when mapping to a canonical trace, then the conclusion is that the testing procedure has reached an unacceptable collision scenario, and the test result analysis procedure halts at this point. We then assume that testers decide the next step at this moment. In this case, the SUT has no error under this test at the halting point.

iii) If neither "T" nor "©T" could be mapped after processing a global event, then "%T" is reported. The conclusion in this case is that the SUT **fails** to pass the test. Note that there is no inconclusive test result resulting from collisions using our approach.

TRA can be implemented in Prolog, so the analysis procedure can be carried out automatically. During the mapping procedures in TRA, the function of backtracking is crucial for guaranteeing that all possible canonical traces for given global event list have been automatically searched. Therefore, by optimizing the backtracking capability of Prolog, the efficiency of the procedure of test result analysis would be enhanced. More studies on relating TRA to Prolog are needed for further research. In this thesis, we only focus on the basic techniques for realizing TRA for test result analysis.

5.2. Examples

In this section, we show how the relative clock based test result analysis approach can achieve a complete error detection power for service and protocol conformance testing by three examples.

5.2.1. Example I

To show how relative clocks based approach can be used for test result analysis for service-direct conformance testing (see Section 2.2.2), we present an OSI transport service example based on the testing architecture made in Figure 5.1. In this example, we want to test the service of the establishment of a (remote) connection between two users. The global tester, through two different communications channels, simulates the two users A and B, who want to establish connection with each other. The SAPs A and B at protocol entities A and B are associated with the users A and B, respectively.

We suppose that there are two errors in the SUT (i.e., the service provider) when providing the above service:

i) Protocol entity A fails to generate A_CR (connection request) to the underlying service provider through SAP A upon receiving *TCONreq* from its "user" (i.e., user A);

ii) Protocol entity B generates *TCONind* spontaneously (to user B through SAP B) without any stimuli from the underlying service provider. The notation for an event-tuple is defined to Section 3.2.

The global view of the above abnormal scenario is shown in the sequence diagram in Figure 5.3.

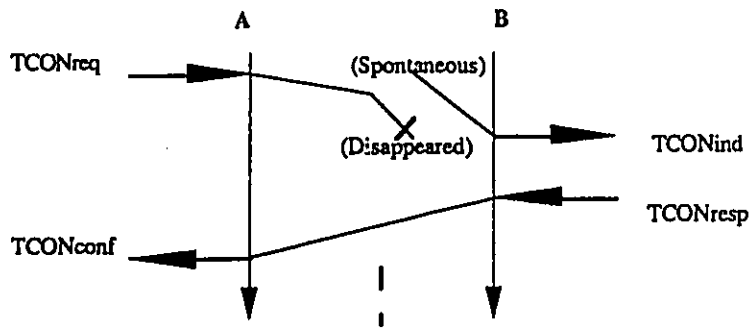

Figure 5.3: The Sequence Diagram of the Abnormal Scenario of Example I

Recorded observed local traces of this scenario are:

at *Tstamp_P#a*: A!TCONreq . A?TCONconf ;

at *Tstamp_P#b*: B?TCONind . B!TCONresp ;

The above two local traces are both legal service scenarios from a local view point and therefore the above errors could not be detected locally without global time-ordering information. The errors will only be detected by a global trace analyzer once the local events with time-ordering information have been transferred to the analyzer. As discussed in Sections 1.1 and 2.2, real clock-based time-ordering information may not be globally correct; we thus, use relative clock as a solution. The testing procedure with global relative time-ordering information proceeds as follows:

First, C_a and C_b are initialized to be "0". Then, following the IRs (see Section 3.1), the relative time of all local events and the changes of values of the local relative clocks are as follows:

Through SAP A:

new message, *TCONreq* will be sent ,

$C_a=C_a+1=1$ (IR2), C_a(A!TCONreq)=1 (IR5), T(TCONreq)= 1 (IR3);

Through SAP B:

new message, *TCONind* is received,

T(TCONind)= 0 (IR4), C_b= 1 + Max {C_b,T(TCONind)} = 1 (IR3),

C_b(B?TCONind)= 1;

Through SAP B:

new message, *TCONresp* is sent by "user B",

C_b =C_b+1 =2, **C_b(B!TCONresp)=2**, T(TCONresp) = 2;

Through SAP A:

new message, *TCONconf* is received,

(since TCONconf and TCONresp are same message and T(TCONresp) = 2),

T(TCONconf) = 2 (IR3),

C_a= 1 + Max{C_a , T(TCONconf) } = 3, **C_a(A?TCONconf)= 3 ;**

Thus, in terms of Definition 3.5, the global relative time of each event is:

C(A!TCONreq) = C(B?TCONind) = 1; C(B!TCONresp)=2; C(A?TCONconf)= 3.

Figure 5.1 identifies the messages exchanged during testing with their relative time.

Thus, the global event trace by the trace merger algorithm (see Section 5.1.2) in terms of relative time is as follows:

<A!TCONreq, B?TCONind> . < A._ , B!TCONresp> . <A?TCONconf, B._ >

However, the (legal) canonical trace (#5.A in the Part_II of Appendix B) for establishing connection should be:

<A!TCONreq, B._ > . < A._ , B?TCONind> . < A._ , B!TCONresp> . <A?TCONconf, B. _ >

That is, no legal canonical trace in the ETAL specification of the SUT (see Part II of Appendix B) can be matched by the merged trace. Therefore, a "%T" is reported, which makes a correct conclusion that the SUT fails to pass the test.

We now analyze the results of this example from another angle. Note that the absence of a timestamp in an observed message implies that the message is produced spontaneously. Since the event *?TCONind* must not be spontaneous in terms of the service definition, one may pursue an argument that the second error can be detected by only the "local" tester (i.e., "user B") with the absence of timestamp in the message, *TCONind*. In other words, the analysis with global information is not necessary to detect this error. However, the argument may not be true since detection power is provided by our global time-ordering approach implicitly. To understand our test result analysis approach based on relative clock more precisely, we should explain this fact from the following two aspects:

i) the information of "the absence of a timestamp" itself is **not local but global** since the global relative clock was used to provide this information. Therefore we may not say that the detection of the error (in terms of the absence) is made locally;

ii) more generally, absence of a timestamp within an observed message is not necessarily an error; for instance, spontaneous event *?TDISind* is allowed in the definition of transport services. Thus, the second phase, ie. an analysis on all related local traces with global time-ordering information is necessary for detecting (globally incorrect) errors.

5.2.2. Example II

We now illustrate by an example how our approach can be used to reconcile the "locally inconclusive" test results resulting from (global) collisions. In the context of this thesis, we define **locally inconclusive test results** as globally legal but locally illegal traces with respect to specifications. The testing architecture used in this example is the same as the one shown in Figure 5.1. The assumptions on the implementation of relative clocks in this example are also the same as those in the last example.

We consider an interesting case that the two "users" (i.e., the testers) send the TCONreq to each other independently of each other when both users are executing a test for testing the service of connection establishment. As shown in Figure 5.4, both testers receive TCONind from its peer later on.

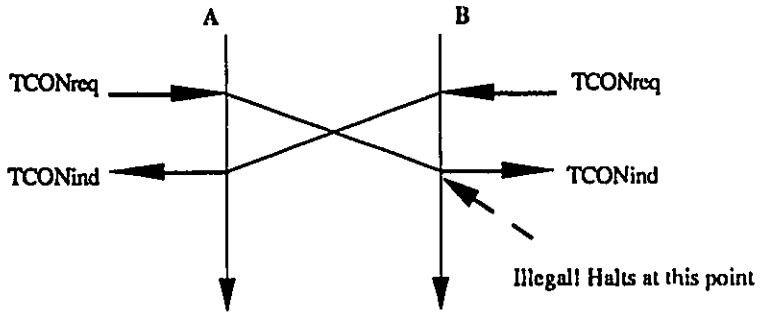

Figure 5.4: The Sequence Diagram of the Abnormal Scenario of Example II

Here we assume that the testing procedure halts in both SAPs A and B since receiving TCONind is illegal after sending TCONreq locally. Then, the local traces in the two SAPs are same and as follows:

at A: A!TCONreq.A?TCONind; at B: B!TCONreq.B?TCONind;

The conclusion on the test result of this test is either inconclusive or that the SUT fails to pass the test if the conclusion is made without global time-ordering information since the above

two traces are locally illegal. This is because the testers cannot judge who is the producer of the message *TCONind*. This obviously limits the error detection power of tests. However, if each local tester sends its observed local trace to the global analyzer with time-ordering information based on relative clock, then we can easily obtain the correct conclusion. The testing procedure proceeds as follows:

Based on the relative clocks shown in Figure 5.1, the local relative time of each local event is: Ca(A!TCONreq) = 1; Ca(A?TCONind) = 2; Cb(B!TCONreq) =1; Cb(B?TCONind) = 2.

Thus, the global relative time of each event (Definition 3.5)

C(A!TCONreq) = C(B!TCONreq) =1: C(A?TCONind) = C(B?TCONind) = 2.

In terms of the global relative time of each local event. By TM module, the global trace is:

<A!TCONreq, B!TCONreq>.<A?TCONind, B?TCONind >

Then, by TRA module, the observed global trace is found matched to an unacceptable canonical trace (see Definition 3.7 and canonical trace#31 of the Part_II, Appendix B):

©T = <A!TCONreq, B!TCONreq>.<A?TCONind, B?TCONind >

Therefore, by TRA module with the ETAL specifications, a reconciled conclusion is that there is no error in both the local service users and the service provider. That is, the test result (expressed in a global event trace) are globally valid (legal) and the locally unexpected (illegal) test results (expressed in the local event traces) were caused by the collision.

5.2.3. Example III

Finally, we show an OSI transport example [BoDsZh89, ISO8073] for protocol (entity - the SUT) conformance testing based on protocol entity model shown in Figure 5.2. We have the same assumptions on the implementation of relative clocks as those in the previous examples. We consider the example, given by [BoDsZh89], namely, trace "t3" as well as similar notation of the error in t3. The errors are as follows:

i) Upon receiving *TCONreq* from its "user" (ie. U tester) , the SUT sends a *CR* PDU, but invokes the *TCONconf* after spontaneously. ii) Upon receiving the *CC* PDU later from its peer (ie. L tester), the SUT fails to pass on a corresponding the *TCONconf* to its user. The second error is not assumed in [BoDsZh89]. However it should be assumed, otherwise the first error could be detected locally since the tester will receive two *TCONconf (s).*

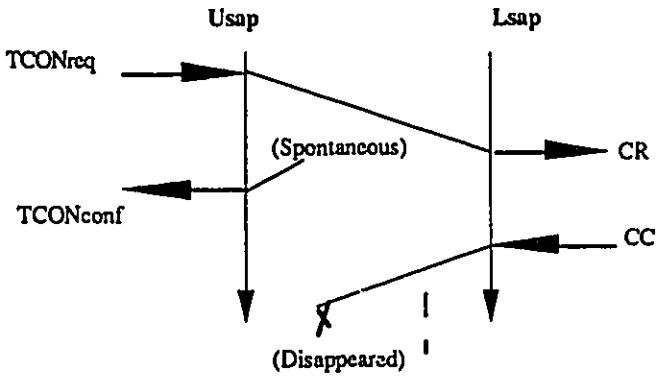

Figure 5.5: The Sequence Diagram of the Abnormal Scenario of Example III

The global view of this scenario is shown in Figure 5.5. The local traces of the scenario are

At Usap: Usap!TCONreq . Usap?TCONconf ;

At Lsap: Lsap?CR . Lsap!CC;

The above two local traces are both legal locally. However, the global observation of the order of the events occurred at the two SAPs is supposed to be the following:

Tg = Usap!TCONreq . Usap?TCONconf . Lsap?CR . Lsap!CC;

This trace is obviously illegal globally. The reason why the error cannot be detected in [BoDsZh89] is that there is no way to form Tg if time-ordering (or ordering) information is not taken into account. However, correctness of time-ordering information cannot be guaranteed if a real clock is used. Next, we show that the errors can easily be detected by using the relative clock based approach. The testing procedure with global relative timing information proceeds as follows:

In terms of the IRs described in Section 3.1, the relative time of each of the above events at Usap and Lsap based on the relative clock are:

Through Usap:

new message, *TCONreq* will be sent,

$C_a=C_a+1=1$ (IR2), C_a**(Usap!TCONreq)=1** (IR5), T(TCONreq)= 1 (IR3(i)) ;

Through Usap:

new message, *TCONconf* is received,

T(TCONconf) = 0 (IR4), $C_a= 1 + \{C_a + T(TCONconf)\} = 2$ (IR3),

C_a**(Usap?TCONconf)=2;**

Through Lsap:

new message, *CR* is received by "L tester" (at Lsap),

Since CR and TCONreq are same message and T(TCONreq) = 1, T(CR)= 1 (IR3) ;

$C_b= 1 + \{C_b.T(CR)\} = 2$ (IR3), C_b**(Lsap?CR)= 2;**

Through Lsap;

new message, *CC* is sent by "L tester",

$C_b = C_b + 1 = 3$, C_b(Lsap!CC)= 3, T(CC) = 3;

The message, *TCONconf* is not invoked by the SUT although the SUT (protocol entity) received the *CC* . That is, U tester would never receive this message.

Hence, the global relative time of each event (Definition 3.5) :

C(Usap!TCONreq)=1, C(Usap?TCONconf)= C(Lsap?CR)= 2, C(Lsap!CC)= 3.

Figure 5.2 shows the messages exchanged during the test procedure with their relative time.

By TM module described in Section 5.1.2, The merged trace expressed in event-tuples is:

Te= < Usap!TCONreq, Lsap._ > . < Usap?TCONconf, Lsap?CR > . <Usap. _ , Lsap!CC > ;

However, the (legal) canonical trace for establishing connection should be:

Tc = < Usap!TCONreq, Lsap. _ > . < Usap._ , Lsap?CR >

.<Usap. _ , Lsap!CC > . < Usap?TCONconf, Lsap. _ >;

That is, no legal canonical trace in the ETAL specification of the SUT can be matched with the trace "Te" by TRA module. Therefore the correct conclusion is that the SUT fails to pass the test.

5.2.4. Error Detection Power and Conclusions

In the previous sections, we have presented the set of realistic examples. With these examples, we have shown that our test result analysis approach is feasible for the following applications:

i) Test result analysis for testing communications services including concurrent aspects.

ii) Reconciliation of locally inconclusive test results (observed traces) from testing local services.

iii) Test result analysis for testing communications services and protocols including spontaneous aspects.

Typically, a complete error detection capability of a tests result analysis procedure would not need to accomplish more than the above requirements. Therefore, we claim that this relative clock based test result analysis approach achieves complete error detection capability.

Moreover, since we can regard the protocol entity as a kind of "service provider" (providing the protocol features through Usap and Lsap), we can state that the relative clock based approach for test result analysis can be used in any external architecture. That is, the approach could be "end-to-end-directed" for acceptance testing [Sale91], based on the above examples.

In Section 6.1, we will give a brief explanation of how the relative clock approach can be used for verification purposes. The approach is very similar to the test result analysis approach above.

5.3. Testability-Directed Specification

As stated in Chapter 1, there exist some relationships between the specification and testing phases in developing a communications (distributed) system. A study exploring such relationships may yield some enhancements to the efficiency of each phase. An industrial

study on testability-directed issues at the design phase has been carried out in [UOBNR]. However, little work has been done on testability at the specification phase. In this section, we focus on the testability-directed specification to explore the relationships between specification and testing.

5.3.1. Requirements on Testability-Directed Specifications

One of our goals in this thesis is to define a means of specifying a communications (distributed) system that can facilitate subsequent testing of the implementation of the system. We say that such specifications are testability-directed specifications. In the context of this thesis, a **testability-directed specification** should fulfill the following requirements:

Firstly, the specification should have strong expressive power such as the capability to specify concurrent behaviours of a communications (distributed) system in a formal and abstract way (i.e., solely in terms of user-observable behaviours of the system [BaPa78, Hoff85, Gotz90]). Then, formal specification based conformance testing can achieve full error detection capability in terms of observed behaviors of the system. That is, the testing activity can correctly detect the errors from observable behaviours allowing for concurrency.

Secondly, the timing model of the specification should facilitate the testing activity. The timing model should be used as guidance for establishing a practical testing environment and for ease of recognizing and generating tests directly and automatically. As a consequence of this requirement , the model of specification should be same as the timing model for testing.

5.3.2. ETAL Specifications are Testability-Directed

In this section, we claim and show that an extended trace specification is a testability-directed specification in terms of the above requirements. Some comparisons of ETAL with other

formal specification languages are also provided with respect to the testability-directed requirements.

First of all, consider the aspect of expressiveness. The specification in ETAL can specify the concurrent behaviours of a communications (distributed) system based on the global events model. Therefore, on one hand, the tests (for instance, using a criterion of covering all canonical traces) could be generated from the specification for testing globally concurrent behaviours. However, more discussion on this issue is beyond the scope of this thesis. On the other hand, the concurrent information in test results (traces) can be recognized directly by the global analyzer from the specification. This is because the time-ordering information in test results is expressed in terms of the same model (i.e. GE model) as that of the specification. However, none of the FDTs can be used for testing globally concurrent behaviours since they lack the capability of specifying global concurrency in a testable way (recall Section 2.2.1).

Secondly, as described in Section 4.1, the specification in ETAL includes trace assertions. Because trace assertions can be interpreted and executed via Prolog (see discussion in Section 5.1.2), tests can be generated from and test results can be recognized with the specification automatically. In addition, as shown in the examples of Section 5.1.3, a reference file [BoDsZh89] for test result analysis is simply the trace specification itself; i.e., the trace assertions represent "trace knowledge" directly. This is because

i) the format of a trace used for test result analysis is the same as that of any canonical trace in trace assertions. However, in FDT-based test result analysis approaches, test results should be first translated into FDT-recognizable form (for instance, [BoBe89]) before analysis. It might either introduce new errors since the translation or reduce the efficiency of test result procedure since the translation step has to be taken.

ii) both the test results and the specification are expressed in terms of observable behaviours, and

iii) assertions could be the "knowledge" since they are logic-based. However, none of FDTs [FORTE] can be used as reference file directly since the specifications in a FDT should first be transformed into the corresponding reference file (for instance, in [BoDsZh89]). Also, directly building the "knowledge" from FDT specifications is still only vaguely described.

Finally, we consider the factor of the establishment of a testing environment such as an external testing architecture. If time-ordering information expressed in the specification of a communications system is based on real clocks and we further require test result analysis based on global time-ordering information, then a synchronization mechanism should be built in the testing environment to fully synchronize the communications (local) real clocks for testing *globally* real-time features of the system. However, experience has shown that such a mechanism can be too costly or even too complex to be realized [Lamp78, BoDsZh89, Fidg91]. We can carry out global test results analysis with respect to a relative clock based specification simply by setting up a set of relative clocks in the testing environment. That is, we do not need to build complex synchronization mechanisms such as would be needed for real clocks; this obviously enhances testability by keeping the cost of implementing clocks quite low.

In summary the relative clock based specification approach can justifiably be described as testability-directed.

CHAPTER 6

EXTENSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

6.1. Verification-directed Specification and ETAL

In this section, we summarize some preliminary results on the relationships between the verifiability-directed specification issues and specification in ETAL. After relating ETAL to testing activities in the previous chapter, relating ETAL to verification is a natural consideration. The definition of verification may vary in different contexts and applications. Our definition is as follows:

> **Implementation verification** is to verify whether an implementation meets the correctness requirements (see Section 2.1.4) as expressed in an extended trace assertion specification.

We now briefly discuss some previous work on verification-directed specification. For verification purposes, [Gotz90] discusses how to specify a communications (distributed) system in a "solely" abstract way by using temporal logic. However, the properties related to global relative concurrent services (e.g. two distributed users send TDTreq to each other, independently of each other, what good things should be happen in the future) cannot be verified based on specification since such properties cannot be specified in local event-based temporal logic. In addition, using past operators in temporal logic reduces the expressiveness of specifications. This is because expressing all possible legal histories (safety properties) by just a few past operators is so complicated. Therefore such specifications are hard to be understood. The method of using traces for describing safety property in [Vogt82] is similar to ours, but it is neither formal compared to ETAL nor detailed as far as validation

(verification and testing) of both sequential and relatively concurrent related safety properties is concerned. Although the verification technique in [ClEmSi86] can be used to verify the liveness property for our method, the safety properties are not naturally addressed since neither past operators nor explicit histories are given for representing past [Gotz90]. Their technique can be modified, albeit somewhat unnaturally, to address safety issues; further discussion is beyond the scope of this thesis.

As in Section 5.2, we now show that specifications in ETAL appear to be verifiability-directed based on some preliminary studies [YuSaPr91].

A specification is said to be **verifiability-directed** if the specification can be directly used as a verification model to automatically verify any implementation. At present, FDs - Formal Descriptions (specifications written according to FDTs, recall also Section 5.3.2 about FDTs) cannot be said to be verifiability-directed (perhaps due to the original intended purpose of FDTs). Examples of problems (refer also the Section 5.3) include:

i) The properties related to global relative concurrent services cannot be verified based on specification since such properties cannot be specified in FDTs.

ii) Automated verification is hard to be done with FDs since they are not assertion-based.

Some papers discussed a related issue, namely verification of the specification itself, "better style" of specification is proposed in [Najm87]. Although ETAL supports verification of the specification [McLe84a, McLe84b], discussion of this issue is beyond the scope of this thesis.

To verify global services provided by a communications system, two steps are needed. The **first step** is to verify local services provided at local access points against local service specifications; the **second step** is to verify global service using the verified local services against the global service specification.

We now show why specifications in ETAL are verifiability-directed. As described in Section 4.1, the service specification in the extended trace language includes two parts: trace assertions and temporal assertions. As the first step of service-directed verification (verification of local services) is similar to but much simpler than the second step (verification of global service), it will not be described here.

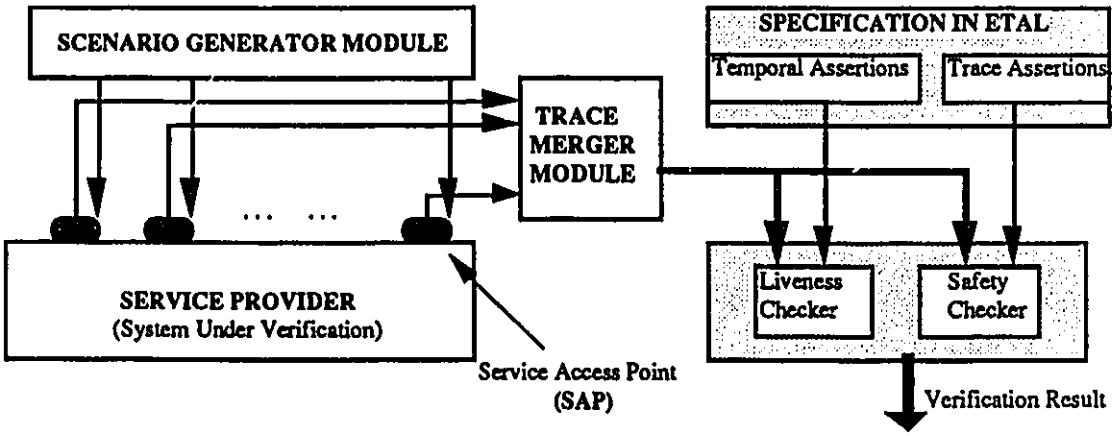

Figure 6.1: Abstract Service-directed Implementation Verification Architecture

Global service verification is illustrated in the Figure 6.1. We assume that the scenario generator module can generate all possible scenarios (since the set of legal service behaviours is finite) and the relative clocks are implemented as in testing. Traces observed at each SAP are merged in the trace merger module based on the global event model. Then liveness properties are verified by checking temporal assertions of the global (ETAL) specification of the service. Finally, safety properties are verified against the trace assertions of the specification, so, we can verify merged implementation traces automatically against the

specification itself. Studies on designing "Safety Checker" module and "Liveness Checker" module (see Figure 6.1) used for verification activities above by relating them to PROLOG are therefore needed for further research. Compared to an earlier method [ClEmSi86], our method appears to be more direct and natural. This is because it is not necessary to construct a "model" since liveness and safety properties can be verified separately with straightforward algorithms based on temporal assertions and trace assertions, respectively. However, more investigation of the relationships needs to be done; this is beyond the scope of this thesis.

In addition, the properties related to global relative concurrent services can be verified based on ETAL specification since such properties can be specified in ETAL. Finally, automated verification can be done with ETAL specifications themselves by relating ETAL specifications to PROLOG since both of them are assertion-based.

Therefore, we can state that specifications in ETAL are verifiability-directed. Combined with our results from Chapter 5, we claim that ETAL specifications are both testability/ verifiability - directed.

One of our intended goals in this topic for further research is linking testing / verification activities by considering testing/verification requirements at the specification phase. We believe that this should give testers support for achieving higher degrees of validation (verification/testing) criteria. However more study on this issue is needed.

6.2. Relationships between ETAL and TTCN

In this section, we describe some initial studies on the relationship between ETAL and TTCN. TTCN (Tree and Tabular Combined Notation) is an abstract standard comprised of a

tabular notation for test specification notation [PrMo91] (see also Section 2.1.5 on conformance testing).

Firstly, we observe that there is a strong similarity in format; the service specification in the extended trace language includes two parts: trace assertions and temporal (event) assertions. The trace assertion part of ETAL is similar to the dynamic behaviour descriptions (tree) part of abstract TTCN specifications of tests. The ETAL temporal assertion part corresponds to the test purposes (tables). Although TTCN is better for representing individual test cases, ETAL is better for representing canonical forms [PrDe90] (classes) of tests than TTCN.

Secondly, in the extended trace language, scenarios of services are still described by traces, not by trees, as in TTCN, which would be a better way to describe scenarios. Work on how to apply TTCN tree-based concepts to extended trace assertions needs to be done. On the other hand, a mathematical foundation (formal semantics) is needed to make TTCN more formal. Therefore a study on how to build the formal semantics of the trace assertion language into TTCN might be useful. Moreover, we note that traces describe infinite scenarios while trees describe finite scenarios. Thus, this approach might be best for relating specification semantics and test semantics, then providing an important link between formal specification, verification and testing.

Thirdly, we observe that the mechanism of error token (or class of error token) in the extended trace language might be useful to specify verdicts in TTCN. Finally, the concept of event-tuple can support concurrency in TTCN for multi-party testing [ISO5076], e.g., the messages between the master tester and other testers may be described by event-tuples in a natural way.

6.3. On the Issue of Handling Collisions

As mentioned in Section 4.1.2, it might be desirable to consider how to handle an unacceptable collision when we are going to specify it. The current approaches on how to handle or avoid collisions are summarized in [Boch90b]. For example, in approach 2, collisions are handled by simply giving priority to incoming (or outgoing) calls over outcoming (or incoming) calls whatever the types of calls are. However it is not fair enough to each user of a system if a purely distributed system is desired (recall the discussion in Section 2.2.1). In approach 4, collisions are avoided by imposing a specific run-time environment of interpretation rule and making an assumption. These rules and the assumption assure that the system contains at most one queued interaction at any given time, and therefore no collision can occur. However, approach 4 can avoid only local collisions* that two events are simultaneous locally since it is assumed that there exists at most one queued interaction at the initial state of the system; that is, this assumption implies that the global collisions caused by relatively concurrent events are not of concern and therefore need not be avoided.

As pointed out in [Boch90b], an approach for handling unacceptable collisions must include requirements for the implementation of a system. A solution for handing unacceptable collisions therefore would be useful in the near future and appears to be straightforward given a few practical assumptions on the implementation details. For example, in the following, we show how to handle the collision (canonical trace#31 of Part_II, Appendix B) resulting from that two distributed users A and B send TCONreq(QoS_A), TCONreq(QoS_B) to each other, respectively, independently of each other. Here, QoS_A

---

* We assume that a system contains a (global) relative clock, and that which event happened first between two events at the same SAP can be measured based on local clock. Then, local collisions can be handled or transformed to the global case implicitly. Therefore it was not considered in our definition of collision (see Definitions 3.8 and 3.9).

(quality of service) is a parameter of the primitive TCONreq [ISO8072]. We also assume that QoS is represented as a natural number. ETAL specification of this collision scenario can be found in Part II of Appendix B. A practical rule to handle this collision based on QoS is as follows:

When user A receives TCONind(QoS_B) from B after sending TCONreq(QoS_A), it compares QoS_A and QoS_B: if QoS_A ≥ QoS_B, then user A does not send TCONresp since the quality which it requested is higher than user B's and the quality of service of two users should be consensus to the lower one; if QoS_A < QoS_B, then user A sends TCONresp to user B.

Meanwhile, follows the same rule as user A does, User B does the same thing as user A does. This rule assures that there is only one TCONresp will be produced and therefore the collision is handled.

Furthermore, the specification of a mechanism for handling unacceptable collisions is desirable for obtaining a complete specification since the tasks for handling such collisions will become services of the system. It is useful to note that the relative clock based specification method is expressive enough to represent the mechanisms for handling collisions, since such mechanisms are just a special kind of relatively concurrent scenarios. ETAL's ability to specify relatively concurrent scenarios has already been discussed in previous chapters and is illustrated with an example presented in Appendix B.

Numerous other extensions are possible, but these extensions appear to be most relevant for achieving a complete testability/verifiability-directed specification methodology.

CHAPTER 7

CONCLUDING REMARKS

In this thesis, we have proposed a new interpretation of Lamport's notion of concurrency of distributed events by introducing the concept "relative concurrency". Based on a global relative clock, one can gain an insight of the nature of distributed scenarios, including collisions, better from this concept than from either traditional (real clock-based) concurrency or interleaved concurrency. This concept has been shown to be beneficial for relating specification and testing activities. Specifically, we have illustrated that specification and testing techniques based on relative clocks appear to be adequate and useful for developing reliable communications software in practice. The key is the life-cycle use of testability-directed, relative clock based reference specifications.

In particular, based on the global event model, we presented a language, namely the extended trace assertion language (ETAL), to represent concurrent aspects of transport services and related the language to a new test result analysis approach.

The relative clock based reference specification directly supports the relationship between specification and testing activities. We have shown specifically that relative clock-based specifications in ETAL facilitate testing procedures and test result analysis. Therefore relative clock-based specifications in ETAL are **testability-directed.** Moreover, when combine this with observations on verifiability-directed specification presented in Section 6.1, we claim that "useful specifications are sets of predicates or assertions" (e.g., specifications in

ETAL). This is the fundamental characterization of testability / verifiability - directed specifications.

We have applied our specification and test result analysis approaches to the set of realistic examples. As one of further studies after the above activities, the researches on the techniques for automatically deriving all possible traces including relative concurrent aspects are needed. Such techniques are crucial for our specification approach to achieve the requirements of completeness and to be further scaled up for specifying large communications services and protocols. Moreover, although our test result analysis approach is efficient for conformance testing of time-ordering aspects of large communications software in practice (Section 6.3), it may not be useful for time-related performance testing. Performance testing may only be based on real clocks rather than relative clocks.

Finally, we believe that testability (and verifiability) directed studies such as this work will contribute significantly to the effectiveness of the software development process, both by enhancing the productivity of testers, and by improving the accessibility of reference specifications by supporting the derivation and validation of relative timed scenarios. It is our opinion that scenario-directed techniques and toolware will yield the greatest improvements in software quality and productivity over this decade.

REFERENCES

[BaPa78] A. Bartussek and D. Parnas, "Using Assertions about Traces to Write Abstract Specifications for Software Modules". *Lecture Notes in Computer Science* (65), Information Systems Methodology, pp. 211-236. 1978.

[Boch90a] G. Bochmann, " Protocol Specification for OSI". *Computer Networks and ISDN Systems*, v.18, pp. 167-184. 1989-90.

[Boch90b] G. Bochmann, "Specifications of a Simplified Transport Protocol Using Different Formal Description Techniques". *Computer Networks and ISDN Systems*, v.18, pp. 335-377. 1989-90.

[BoBe89] G. Bochmann and O. Bellal, "Test Result Analysis in respect to Formal Specifications ". *Proceedings of the Second International Workshop on Protocol Test Systems*. pp. 272-294. Oct., 1989.

[BoDsZh89] G. Bochmann, R. Dssouli and J-R. Zhao, "Trace Analysis for Conformance and Arbitration Testing". *IEEE Tran. Soft. Eng.*, pp. 1347-1356. Nov. 1989.

[ChLa85] K. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", *ACM Tran. Computer Sys.*, v.3, n.1, pp. 63-75. Feb., 1985.

[ClEmSi86] E. Clarke, E. Emerson and A. Sistla , "Automatic Verification of Finite-State Concurrent System Using Temporal Logic Specification". *ACM Tran. on Prog. Lang. and Sys.*, pp. 244-263. 1986.

[Gotz90] R. Gotzhein, "Specifying Communication Services with Temporal Logic". *Protocol Specification, Testing, and Verification* X (ed. L. Logrippo, R. Probert and H. Ural), pp. 287-304. 1991. North-Holland.

[Fidg91] C. Fidge, "Logical Time in Distributed Computing Systems". *IEEE Computers*, pp. 28-33, Aug., 1991.

[FORTE] Formal Description Techniques for Distributed Systems, *Proceedings of International Conference on Formal Description Techniques for Distributed Systems*, 1988-1991.

[HaWe88] D. Haban and W. Weigel, "Global Events and Global Breakpoints in Distributed Systems". *Proc. of 21st International Conference on System Science*. v. II, pp. 166-175. 1988.

[Hoar85] R. Hoare, "Mathematical Logic and Programming Languages", Prentice-Hall International, 1985.

[Hoff85] D. Hoffman, "The Trace Specification of Communications Protocols". *IEEE Trans. Computers*, Vol. c-34, No.12, pp. 1102-1113. Dec., 1985.

[HoSn88] D. Hoffman and R. Snodgrass, "Trace Specifications: Methodology and Models". *IEEE Tran. Soft. Eng.*, v.14, n.9., pp.1243-1251. Sept., 1988.

[ISO7498] ISO7498, "IPS - OSI - Basic Reference Model". 1984.

[ISO5076] ISO/IEC JTC1/SC21 N5076 "Working Draft for Multi-Party Testing". Aug., 1990.

[ISO8072] ISO 8072, IPS-OSI-Transport Service Definitions. 1986.

[ISO8073] ISO 8073, IPS-OSI Connection Oriented Transport Protocol Specification. 1985.

[ISO9646] ISO-DIS 9646, ISO/IEC JCT1/SC21, "IPS - OSI - OSI Conformance Testing Methodology and Framework". 1989.

[Jeff85] D. Jefferson, "Virtual Time". *ACM Tran. on Prog. Lang. and Sys.*, v.7, n.3, pp. 404-425. July, 1985.

[Lamp78] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System". *Comm. of ACM*, v.21,n.7, pp. 558-565, July, 1978.

[Liu89] M-L. Liu, "Protocol Engineering", *Advances in Computers*, v.27, pp. 79-195. July, 1989.

[LoSiUr82] L. Logrippo, D. Simon and H. Ural, "Executable Description of the Transport Service in Prolog". *Protocol Specification, Testing and Verification IV* , North-Holland, pp. 279-293. 1985.

[McLe84a] J. McLean, "A Formal Method for the Abstract Specification of Software", *Journal of ACM*, Vol. 31, No. 3, pp. 279-293. July, 1984.

[McLe84b] J. McLean, "A Complete System of Temporal Logic for Specification Schemata", *Lecture Notes in Computer Science* , pp. 360-370. 1984.

[McWeLa86] J. McLean, D. Weiss and C. Landweher, "Executing Trace Specifications Using PROLOG". *NRL Report 8940*, 1986.

[Morg85] C. Morgan, "Global and Logical Time in Distributed Snapshots". *Info. Proc. Lett.* 20 (1985) pp. 189-194. May, 1985.

[Najm87] E. Najm, "A Verification oriented specification in LOTOS of the transport protocol". *Protocol Specification, Testing and Verification VII*, pp. 181-203. North-Holland, 1988.

[PaMa90] D. Parnas and J. Madey, "Functional Documentation for Computer Systems Engineering, *TR 90-287*, Queen's University, Canada. Sept., 1990.

[PaWa89] D. Parnas and Y. Wang "The Trace Assertion Method of Module Interface Specification". *TR 89-261* , Queen's University, Canada. 1989. Also, to appear in *IEEE Tran. Soft. Eng.*.

[PrDe90] R. Probert, M. Desjardins, "Improving Quality and Interoperability of Protocol Implementations via Conformance Testing Standardization". *Proceedings of IEEE International Conference on Communications*, v.4, pp. 1374-1381. Apr., 1990.

[PrMo91] R. Probert, O. Monkewich, "TTCN - The International Notation For Conformance Testing of Communications Systems". TR 90-38, Sept., 1990. Also, to appear in *Computer Networks & ISDN Systems*. 1991.

[PrSa91] R.Probert, K. Saleh, "Synthesis of Communications Protocols: Survey and Assessment", (Special Issue on Protocol Engineering), *IEEE Tran. Computers*, v.40, n.4, pp. 468-476. Apr., 1991.

[PrYuSa92] R. Probert, H-L. Yu and K. Saleh, "Relative-Clock-Based Specification and Test Result Analysis of Distributed Systems". To appear in the *Proceedings of 11th IEEE Phoenix International Conference on Computers and Communications (accepted in Sept. 1991)*. Apr. 1992. Also in *TR-91-38, Dept. of Computer Science*, University of Ottawa. Oct., 1991.

[PTS] Protocol Test Systems, *Proceedings of International Workshop on Protocol Test Systems*. 1988-1991.

[PSTV] Protocol Specification, Testing and Verification, *Proceedings of the IFIP WG 6.1 International Workshop (81-85) or Symposium (86-91) on Protocol Specification, Testing and Verification I - XI*. 1981-1991.

[Rayn87] D. Rayner, "OSI Conformance Testing" , *Computer Networks & ISDN Systems*. v.14, pp. 79-92. 1987.

[Sale91] K. Saleh, "Synthesis Methods For the Design and Validation of Communication Protocols". *Ph.D.Thesis*, University of Ottawa. Jan., 1991.

[Scho90] A. Schouwen, "The A-7 Requirements Model: Re-examination for Real-Time Systems and an Application to Monitoring Systems". *TR 90-276, Queen's University*. May, 1990. Also, to appear Comm. of ACM. 1991.

[SpBr87] The SPECS Consortium and J. Bruijing, "Evaluation and Integration of Specification Languages". *Computer Networks and ISDN Systems*, v.13, pp. 75-89, 1987.

[Schw87] M. Schwarts, Telecommunication Networks, Addison-Wesley. 1987.

[UOBNR] A Join Project of University of Ottawa and Bell Northern Research ltd., "Design System for Testability". Ottawa. 1990-1991.

[UrPr86] H. Ural and R. Probert , "Step-Wise Validation of Communication Protocols and Services". *Computer Networks and ISDN Systems*, v.11, n.3, pp. 183-202, Mar., 1986.

[UrYa91] H. Ural and B. Yang, "A Test Sequence Selection Method for Protocol Testing". *IEEE Trans. Comm.* v. 39, n. 4 , pp. 514-523, Apr., 1991.

[ViLo86] C. Vissers and L. Logrippo, "The Importance of the Service Concept in the Design of Data Communication Protocols". *Protocol Specification, Testing and Verification V*, pp. 3-17, North-Holland. 1986.

[Vogt82] F. Vogt, "Even-Based Temporal Logic Specifications of Services and Protocols". *Protocol Specification, Testing and Verification II* , pp. 63-73, North-Holland. 1982.

[YuSaPr91] H-L. Yu, K. Saleh and R. Probert, "Testability-Directed Trace Specifications of Communications Services", *Official Proceedings of Telecommunications Research Institute of Ontario (TRIO) Researchers Conference'91*, Peterborough, ON. May, 1991. Also in *TR 91-28, Department of Computer Science*, University of Ottawa. 1991.

[Zimm80] H. Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection ". *IEEE Trans. Comm.* , v. COM-28, pp. 425-432. Apr., 1980.

Appendix A: Trace_Merger Module (pseudo code)

Data Structures:

```
Type
        number_of_sap, relative_time : unassigned integer;
        event_type ={...}              /* varies for different applications; for the TS example,
                                          event_type = {!TCONreq, ?TCONind, !TCONresp,
                                          ?TCONconf, !TDTreq,  ?TDTind, !TDISreq, ?TDISind}  */

        ID_of_service_access_point = 1..number_of_sap;
        event_tuple = arrary[1..number_of_sap] of event_type;
        local_event = record of
                        ID_sap: ID_of_service_access_point;
                        type_e: event_type;
                        timestamp: relative_time;
                        end;
        global_event = record of
                        e_tuple: event_tuple;
                        order_no: relative_time;
                        end;
```

Bodies of Algorithms:

e_snapshot#i:

```
var     new_event: local_event
begin
    when occurrence of a event  SAPi.m  is observed
        create new record new_event ;
        assign a natural number to the field "new_event.timestamp" based on IRs (Section 3.1.2);
        new_event.type := .m;
        add new_event  at the end of  queue#i;
      end.
end  (of e_snapshot#i)
```

```
process_event:
var
        end_T_conversation : boolean;                   /* signal of the end of a
                                                        testing conversation (group) */
        e_buffer: local_event;
        new_g_event, current_g_event: global_event;
        i: integer;
begin
        end_T_conversation := false;
        i := 0;
        Repeat
                i := i +1;
                check queue #i;
                if queue#i is not empty
                then
                  begin
                    e_buffer := head of queue#i;
                    remove the head from the queue;
                    current_g_event:= head of global_event_list;
                    while current_g_event ≠ NULL        /* search the global_event_list */
                        begin                           /* for the same value of timestamp */

                            if  e_buffer.timestamp = current_g_event.order_no
                              then                      /* modify the current_g_event  */
                                current_g_event.e_tuple[e_buffer.ID_sap] := e_buffer.type_e;
                                exit;       /* from while */
                               else
                                current_g_event := next  of current_g_event;
                        end;  /* of while */
```

```
            if current_g_event = NULL          /* no global event's timestamp is
                                                  same as the local event's timestamp,
                                                  which is being processed */
                    then                       /* So, create a new global event  */

                        begin
                            create new record new_g_event;
                            new_g_event.e_tuple[e_buffer.ID_sap]:= e_buffer.type_e;
                            new_g_event.order_no := e_buffer.timestamp;
                            insert new_g_event into the global_event_list in an
                                    ascending order by its global relative time;
                        end;
                Output (global_event_list);    /* a kind of output , e.g. to be sent to "trace
                                                  analyzer module" for analyzing legality of the
                                                  trace */
        end;        /* of the first "if" */

        if i = number_of_sap then i := 0;      /*check each sequence in recycle way)
        read_control(end_T_conversation);       /* tester or analyzer control a testing
                                                   conversation by keyboard*/
    Until end_T_conversation
End ( of module)
```

Note: This module has been tested based on functional criteria against its specification given in Section 5.1.2. A set of tests has been successfully passed.

Appendix B: Specification of OSI Transport Service in ETAL

Part I: ETAL specification of local transport services

SYNTAX:

1> Access-Programs

Program Name	Value	Arg.
!TCONreq	(Called Addr., Calling Addr., QoS, TS user-data)	TPDU
?TCONconf	(Called Addr., Calling Addr., QoS, TS user-data)	TPDU
?TCONind	(Called Addr., Calling Addr., QoS, TS user-data)	TPDU
!TCONresp	(Called Addr., Calling Addr., QoS, TS user-data)	TPDU
!TDTreq	(TS user-data)	TPDU
?TDTind	(TS user-data)	TPDU
!TDISreq	(TS user-data)	TPDU
?TDISind	(Disc_reason_ts ,TS user-data) (from peer) or (Disc_reason_tp ,TS user-data) (from s. provider)	TPDU

Table A.I.1

SEMANTICS:

2> Canonical(T) <->

(T = _)
V (T = SAP!TCONreq)
V (T = SAP!TCONreq.SAP?TCONconf)
V (T = SAP?TCONind)
V (T = SAP?TCONind.SAP!TCONresp)
V (T= Tp.Tq
∧(Tp = SAP!TCONreq.SAP?TCONconf V Tp = SAP?TCONind.SAP!TCONresp)
∧(Tq= SAP?TDTind V Tq= SAP!TDTreq V Tq= Tq.SAP?TDTind V Tq= Tq.SAP!TDTreq)
)

3> Auxiliary Functions

Function Name	Value	Arg
Tdatatran	<boolean>	<trace>

Table A.I.2

Tdatatran(T) =

Condition	Value
(T= Tp.Tq ∧(Tp = SAP!TCONreq. SAP?TCONconf) ∧(Tq= SAP?TDTind V Tq= SAP!TDTreq V Tq= Tq.SAP?TDTind V Tq= Tq.SAP!TDTreq)) V (T= Tp.Tq ∧(Tp = SAP?TCONind . SAP!TCONresp) ∧(Tq= SAP?TDTind V Tq= SAP!TDTreq V Tq= Tq.SAP?TDTind V Tq= Tq.SAP!TDTreq))	ture
(T≠ Tp.Tq ∧(Tp = SAP!TCONreq. SAP?TCONconf) ∧(Tq= SAP?TDTind V Tq= SAP!TDTreq V Tq= Tq.SAP?TDTind V Tq= Tq.SAP!TDTreq)) ∧(T≠ Tp.Tq ∧(Tp = SAP?TCONind . SAP!TCONresp) ∧(Tq= SAP?TDTind V Tq= SAP!TDTreq V Tq= Tq.SAP?TDTind V Tq= Tq.SAP!TDTreq))	false

Table A.I.3

4> Equivalence:

T.SAP!TCONreq ≡

conditions	equivalences
T= "_"	SAP!TCONreq
T≠ "_"	%T

Table A.I.4

T.SAP?TCONind ≡

conditions	equivalences
T= "_"	SAP!TCONind
T≠ "_"	%T

Table A.I.5

T.SAP!TCONresp ≡

conditions	equivalences
T= SAP?TCONind	SAP?TCONind.SAP!TCONresp
T ≠ SAP?TCONind	%T

Table A.I.6

T.SAP?TCONconf ≡

conditions	equivalences
T= SAP?TCONreq	SAP?TCONreq.SAP!TCONconf
T ≠ SAP?TCONreq	%T

Table A.I.7

T.SAP?TDTind ≡

conditions	equivalences
T = SAP!TCONreq.SAP?TCONconf	SAP!TCONreq.SAP?TCONconf.SAP?TDTind.
T = SAP?TCONind.SAP!TCONresp	SAP?TCONind.SAP!TCONresp.SAP?TDTind
Tdatatran(T)	T
T ≠ SAP!TCONreq.SAP?TCONconf ∧T ≠ SAP?TCONind.SAP!TCONresp ∧T ≠ Tdatatran(T)	%T

Table A.I.8

T.SAP!TDTreq ≡

conditions	equivalences
T = SAP!TCONreq.SAP?TCONconf	SAP!TCONreq.SAP?TCONconf.SAP?TDTreq.
T = SAP?TCONind.SAP!TCONresp	SAP?TCONind.SAP!TCONresp.SAP?TDTreq
Tdatatran(T)	T
T ≠ SAP!TCONreq.SAP?TCONconf ∧T ≠ SAP?TCONind.SAP!TCONresp ∧T ≠ Tdatatran(T)	%T

Table A.I.9

T.SAP!TDISreq ≡

conditions	equivalences
T= "_"	%T
T≠ "_"	"_"

Table A.I.10

T.SAP?TDISind(ts) ≡

conditions	equivalences
T= "_"	%T
T≠ "_"	"_"

Table A.I.11

T.SAP?TDISind(tp) ≡

conditions	equivalences
T= "_"	%T
T≠ "_"	"_"

Table A.I.12

5> Temporal Assertions (Liveness Properties only):

i) [] (SAP!TCONreq => <> (SAP?TCONconf V SAP?TDISind(ts) V SAP?TDISind(tp) V SAP!TDISreq))

ii) [] (SAP!TDTreq =><> (SAP?TDTreq V SAP?TDTind V SAP!TDISreq V SAP?TDISind(ts) V SAP?TDISind(tp)))

(end of specification of local view of TS service)

Part II: Specification of global view of OSI transport service

SYNTAX:

1> Access-Programs

Program Name	Value	Arg.
!TCONreq	(Called Addr., Calling Addr., QoS, TS user-data)	TPDU
?TCONconf	(Called Addr., Calling Addr., QoS, TS user-data)	TPDU
?TCONind	(Called Addr., Calling Addr., QoS, TS user-data)	TPDU
!TCONresp	(Called Addr., Calling Addr., QoS, TS user-data)	TPDU
!TDTreq	(TS user-data)	TPDU
?TDTind	(TS user-data)	TPDU
!TDISreq	(TS user-data)	TPDU
?TDISind	(Disc_reason_ts ,TS user-data) (from peer) or (Disc_reason_tp ,TS user-data) (from s. provider)	TPDU

Table: A.II.1

SEMANTICS:

2>Canonical(T) <->

1 ═══ (T= < _ , _ >) /* Idle global canonical trace of TS global service */

Connection request from A:

2.A ⟹ V (T = <A!TCONreq, _ >

3.A ⟹ V (T = <A!TCONreq, _ >.< _ ,B?TCONind>)

4.A ⟹ V (T = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>)

5.A ⟹ V (T = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>.<A?TCONconf, _ >)

6.A ⟹ V (T = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>.<A?TCONconf, B!TDTreq >)

7.A ⟹ V (T = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>.<A?TCONconf, _ >
.<A!TDTreq, _ >)

8.A ⟹ V (T = Tp.Tq
∧ Tp= <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>.<A?TCONconf, B!TDTreq>
∧ (Tq = <A!TDTreq, _ > V Tq = < _ ,B!TDTreq>
V Tq=<A?TDTind, _ > V Tq = <A?TDTind ,B!TDTreq> V Tq = <A!TDTreq ,B!TDTreq>
V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq>
V Tq=Tq.<A?TDTind, _ >V Tq = Tq.<A?TDTind ,B?TDTind>
V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq ,B!TDTreq>
V Tq = Tq.<A!TDTreq ,B?TDTind>))

9.A ⟹ V (T = Tp.Tq
∧ Tp= <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>.<A?TCONconf, _ >
.<A!TDTreq, _ >
∧ (Tq = <A!TDTreq, B?TDTind> V Tq = < _ ,B?TDTind>
V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq>
V Tq=Tq.<A?TDTind, _ >V Tq = Tq.<A?TDTind ,B?TDTind>
V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq ,B!TDTreq>
V Tq = Tq.<A!TDTreq ,B?TDTind>))

10.A ⟹ V ((T = Tm.<A!TDISreq, _ > ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp) ∧ E = <A?TDTind, _ >
V Tm = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>.<A?TCONconf, _ >
V Tm= <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>
.<A?TCONconf, B!TDISreq>
V Tm= < _ , B!TCONreq>.< A?TCONind, B!TDISreq>))

11.A ≡≡ V (T = <A!TCONreq, _ >.< A!TDISreq, B?TCONind>)

12.A ≡≡ V (©T = <A!TCONreq, _ >.< A!TDISreq, B?TCONind>.< _ ,B!TCONresp>)

13.A ≡≡ V (©T = <A!TCONreq, _ >.< A!TDISreq, B?TCONind>.<A!TCONreq, B!TCONresp>)

14.A ≡≡ V (T = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>.<A?TCONconf, B!TDISreq>)

15.A ≡≡ V (©T = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>.<A?TCONconf, B!TDISreq>
.<A!TDTreq, _ >)

16.A ≡≡ V (©T = Tm.<A!TDISreq, _ > ∧
((Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp) ∧ E = <A?TDTind, B!TDTreq>)
V (Tm = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>
.<A?TCONconf, B!TDTreq>)))

17.A ≡≡ V (©T = Tm.<A!TDISreq, B!TDTreq > ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp) ∧
(E = <A?TDTind ,B?TDTind> V E = <A?TDTind ,B!TDTreq>))
V (Tm = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>
.<A?TCONconf, B!TDTreq>)))

18.A ≡≡ V (©T = Tm.<A!TDISreq, B!TDTind > ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp) ∧ E = <A!TDTreq ,B?TDTreq>))

19.A ≡≡ V (T = <A!TCONreq, _ >.<A?TDISind(tp), B?TCONind>

20.A ≡≡ V (©T = <A!TCONreq, _ >.<A?TDISind(tp), B?TCONind>.< _ ,B!TCONresp>

21.A ≡≡ V (©T = Tm.< A?TDISind(tp), _ > ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp)
∧ (E = <A?TDTind, B!TDTreq> V E = <A!TDTreq, _ >)
V Tm = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>
.<A?TCONconf, B!TDTreq>))

22.A ≡≡ V (©T = Tm.< A?TDISind(tp), B!TDTreq> ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp)
∧ (E = <A?TDTind ,B!TDTreq> V E = <A?TDTind ,B?TDTind>)
V Tm = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>
.<A?TCONconf, B!TDTreq>))

23.A ⟹ V (©T = Tm.< A?TDISind(tp), B?TDTind> ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp) ∧ E = <A!TDTreq ,B!TDTreq>))

24.A ⟹ V (T = <A!TCONreq, _>.<_ ,B?TCONind>.< _ ,B!TCONresp>.<A?TCONconf, B?TDISind(tp)>)

25.A ⟹ V (©T = <A!TCONreq, _>.<_ ,B?TCONind>.< _ ,B!TCONresp>
.<A?TCONconf, B?TDISind(tp)>.<A!TDTreq, _>)

26.A ⟹ V (T = Tm.<A?TDISind(tp), B?TDTind> ∧ Tm = Canonical(Tm) ∧ Tm= Tp. E
∧ (E = <A!TDTreq, _ > V E = <A!TDTreq, B?TDTind >))

27.A ⟹ V (©T = Tm.< _ , B!TDTreq > ∧ Tm = Canonical (Tm) ∧ Tm= Tp. E
∧ E = <A?TDISind(tp), B?TDTind>)

28.A ⟹ V (T = Tm.<A?TDISind(tp), B?TDISind> ∧ Tm = Canonical(Tm) ∧ Tm= Tp. E
∧ E = <A!TDISreq, _ >)

29.A ⟹ V (©T = Tm.< A?TDISind(tp), B!TDISreq> ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp)
∧ (E = <A?TDTind ,B!TDTreq> V E = <A?TDTind ,B?TDTind>)
V Tm = <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp>
.<A?TCONconf, B!TDTreq>))

Connection request from B:

2.B ≡ V (T = < _ ,B!TCONreq>)

3.B ≡ V (T = < _ ,B!TCONreq>.<A?TCONind, _ >)

4.B≡ V (T = < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ >)

5.B ≡ V (T = < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ >.< _ ,B?TCONconf>)

6.B ≡ V (T = < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ >.<A!TDTreq, B?TCONconf>)

7.B ≡ V (T = < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ >.< _ ,B?TCONconf>)
.< _ ,B!TDTreq>)

8.B ≡ V (T = Tp.Tq
∧ Tp= < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ >.<A!TDTreq, B?TCONconf>
∧ (Tq = <A!TDTreq, _ > V Tq = < _ ,B!TDTreq>
V Tq=< _ ,B?TDTind> V Tq = <A!TDTreq ,B?TDTind> V Tq = <A!TDTreq ,B!TDTreq>
V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq>
V Tq=Tq.<A?TDTind, _ >V Tq = Tq.<A?TDTind ,B?TDTind>
V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq ,B!TDTreq>
V Tq = Tq.<A!TDTreq ,B?TDTind>))

9.B ≡ V (T = Tp.Tq
∧ Tp= < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ >.< _ ,B?TCONconf>)
.< _ ,B!TDTreq>)
∧ (Tq = <A?TDTind ,B!TDTreq> V Tq = <A?TDTind, _ >
V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq>
V Tq=Tq.<A?TDTind, _ >V Tq = Tq.<A?TDTind ,B?TDTind>
V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq ,B!TDTreq>
V Tq = Tq.<A!TDTreq ,B?TDTind>))

10.B ≡ V ((T = Tm.< _ ,B!TDISreq> ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp) ∧ E = < _ ,B?TDTind>
V Tm = < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ >.< _ ,B?TCONconf>
V Tm= < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ >
.<A!TDISreq, B?TCONconf>)
V Tm= <A!TCONreq, _ > . <A!TDISreq, B?TCONind>

11.B ≡ V (T = < _ ,B!TCONreq>.<A?TCONind, B!TDiSreq>)

12.B ≡ V (©T = < _ ,B!TCONreq>.<A?TCONind, B!TDISreq>.<A!TCONresp, _ >)

13.B ≡ V (©T = < _ ,B!TCONreq>.<A?TCONind, B!TDISreq>.<A!TCONresp, B!TCONreq>)

14.B ≡ V (T=< _ ,B!TCONreq>.<A?TCONind, _ >.<A!TCONresp, _ >.<A!TDISreq, B?TCONconf>)

15.B ≡ V (©T=< _ ,B!TCONreq>.<A? ΓCONind, _ >.<A!TCONresp, _ >.<A!TDISreq,B?TCONconf>
.< _ , B!TDTreq>)

16.B ≡ V (©T = Tm.< _ ,B!TDISreq> ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp) ∧ E = <A!TDTreq, B?TDTind>
V Tm = < _ ,B!TCONreq>.<A?TCONind, _ >.<A!TCONresp, _ >
.<A?TCONconf, B!TDTreq>))

17.B ≡ V (©T = Tm.< A!TDTreq, B!TDISreq> ∧
(Tm = Tp.E ∧ Canonical (Tm) = Canonical(Tp)
∧ (E = <A!TDTreq ,B?TDTind> V E = <A?TDTind ,B?TDTind>)
V Tm = < _ ,B!TCONreq>.<A?TCONind, _ >.<A!TCONresp, _ >
.<A?TCONconf, B!TDTreq>))

18.B ≡ V (©T = Tm.< A?TDTind, B!TDISreq> ∧
(Tm = Tp.E ∧ Canonical (Tm) = Canonical(Tp) ∧ E = <A?TDTind ,B!TDTreq>))

19.B ≡ V (T = < _ ,B!TCONreq>.<A?TCONind, B?TDISind(tp)>

20.B ≡ V (©T = < _ ,B!TCONreq>.<A?TCONind, B?TDISind(tp)>.<A!TCONresp, _ >

21.B ≡ V (©T = Tm.< _ , B?TDISind(tp)> ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp) ∧
(E =<A!TDTreq, B?TDTind> V E = <_ , B!TDTreq>))
V Tm = < _ ,B!TCONreq>.<A?TCONind, _ >.<A!TCONresp, _ >
.<A!TDTreq, B?TCONconf>))

22.B ≡ V (©T = Tm.< A!TDTreq, B?ΓDISind(tp)> ∧
(Tm = Tp.E ∧ Canonical (Tm) = Canonical(Tp)
∧ (E = <A!TDTreq ,B?TDTind> V E = <A?TDTind ,B?TDTind>)
V Tm = < _ ,B!TCONreq>.<A?TCONind, _ >.<A!TCONresp, _ >
.<A!TDTreq, B?TCONconf>))

23.B ≡≡≡ V (©T = Tm.< A?TDTind, B?TDISind(tp)> ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp) ∧ E = <A!TDTreq ,B!TDTreq>))

24.B ≡≡≡ V (T =<_ ,B!TCONreq>.<A?TCONind, _>.<A!TCONresp, _ >.<A?TDISind(tp), B?TCONconf>)

25.B ≡≡≡ V (T =<_ ,B!TCONreq>.<A?TCONind, _ >.<A!TCONresp, _ >
.<A?TDISind(tp), B?TCONconf>.<_ ,B!TDTreq>)

26.B ≡≡≡ V (T = Tm.<A?TDTind, B?TDISind(tp)> ∧ Tm = Canonical(Tm) ∧ Tm= Tp. E
∧ (E = < _ ,B!TDTreq> V E = <A?TDTind, B!TDTreq>))

27.B ≡≡≡ V (T = Tm. <A!TDTreq , _ > ∧ Tm = Canonical(Tm) ∧ Tm= Tp. E
∧ E = <A?TDTind, B?TDISind(tp)>)

28.B ≡≡≡ V (T = Tm.<A?TDISind, B?TDISind(tp)> ∧ Tm = Canonical(Tm) ∧ Tm= Tp. E
∧ E = < _ ,B!TDISreq>)

29.B ≡≡≡ V (©T = Tm.< A!TDISreq, B?TDISind(tp)> ∧
(Tm = Tp.E ∧ Canonical(Tm) = Canonical(Tp)
∧ (E = <A!TDTreq, B?TDTind> V E = <A?TDTind ,B?TDTind>)
V Tm = < _ ,B!TCONreq>.<A?TCONind, _ >.<A!TCONresp, _ >
.<A!TDTreq, B?TCONconf>))

Connection requests from both sides:

30 ≡ V (T =< A!TCONreq, B!TCONreq>)

31 ≡ V (©T =< A!TCONreq, B!TCONreq>.< A?TCONind, B?TCONind>)

32 ≡ V (©T =< A!TCONreq, B!TCONreq>.< A!TDISreq, B!TDISreq>)

Disconnection requests from both sides:

33 ≡ V (T = Tm.<A!TDISreq, B!TDISreq>
∧ Tm = Canonical(Tm) ∧ Tm = Tp.E ∧ Tp = Canonical(Tp)
∧ (E = <A!TDTreq, B?TDTind> V E = <A?TDTreq, B!TDTind>
V E = <A!TDTreq, B?TCONconf>V E = <A?TCONconf, B!TDTreq>
V E = <A!TDTreq ,B!TDTreq>))

Illegal canonical trace:

34 ≡ V (%T = T ∧ T ≠ T ∧ T ≠ ©T)

3> Auxiliary Functions

Function name	Value	Arg
Tcon_a_tran1	<boolean>	<trace>
Tcon_b_tran1	<boolean>	<trace>
Tcon_a_tran2	<boolean>	<trace>
Tcon_b_tran2	<boolean>	<trace>
Lost(T)	<boolean>	<trace>

Table A.II.2

Tcon_a_tran1(T) =

Condition	Value
(T = Tp.Tq ∧Tp= < A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf, B!TDTreq> ∧(Tq = <A!TDTreq, _ > V Tq = < _ ,B!TDTreq> V Tq=< _ ,B?TDTind> V Tq = <A!TDTreq ,B?TDTind> V Tq = <A!TDTreq ,B!TDTreq> V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq> V Tq=Tq.<A?TDTind, _ >V Tq = Tq.<A?TDTind ,B?TDTind> V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq,B!TDTreq> V Tq = Tq.<A!TDTreq ,B?TDTind>))	ture
T ≠ Tp.Tq ∧Tp= < A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf, B!TDTreq> ∧(Tq = <A!TDTreq, _ > V Tq = < _ ,B!TDTreq> V Tq=< _ ,B?TDTind> V Tq = <A!TDTreq ,B?TDTind> V Tq = <A!TDTreq ,B!TDTreq> V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq> V Tq=Tq.<A?TDTind, _ >V Tq = Tq.<A?TDTind ,B?TDTind> V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq,B!TDTreq> V Tq = Tq.<A!TDTreq ,B?TDTind>))	false

Table A.II.3

Tcon_a_tran2(T) =

Condition	Value
T = Tp.Tq ∧Tp= <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp> .<A?TCONconf, _ >.<A!TDTreq, _ > ∧(Tq = <A!TDTreq, B?TDTind> V Tq = < _ ,B?TDTind> V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq> V Tq=Tq.<A?TDTind, _ >V Tq = Tq.<A?TDTind ,B?TDTind> V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq ,B!TDTreq> V Tq = Tq.<A!TDTreq ,B?TDTind>))	ture
T ≠ Tp.Tq ∧Tp= <A!TCONreq, _ >.< _ ,B?TCONind>.< _ ,B!TCONresp> .<A?TCONconf, _ >.<A!TDTreq, _ > ∧(Tq = <A!TDTreq, B?TDTind> V Tq = < _ ,B?TDTind> V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq> V Tq=Tq.<A?TDTind, _ >V Tq = Tq.<A?TDTind ,B?TDTind> V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq ,B!TDTreq> V Tq = Tq.<A!TDTreq ,B?TDTind>))	false

Table A.II.4

Tcon_b_tran1(T) =

Condition	Value
T = Tp.Tq ∧Tp= < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ > .<A!TDTreq, B?TCONconf> ∧(Tq = <A!TDTreq, _ > V Tq = < _ ,B!TDTreq> V Tq=< _ ,B?TDTind> V Tq = <A!TDTreq ,B?TDTind> V Tq = <A!TDTreq ,B!TDTreq> V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq> V Tq=Tq.<A?TDTind, _ >V Tq = Tq.<A?TDTind ,B?TDTind> V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq ,B!TDTreq> V Tq = Tq.<A!TDTreq ,B?TDTind>))	ture
T ≠ Tp.Tq ∧Tp= < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ > .<A!TDTreq, B?TCONconf> ∧(Tq = <A!TDTreq, _ > V Tq = < _ ,B!TDTreq> V Tq=< _ ,B?TDTind> V Tq = <A!TDTreq ,B?TDTind> V Tq = <A!TDTreq ,B!TDTreq> V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq> V Tq=Tq.<A?TDTind, _ >V Tq = Tq.<A?TDTind ,B?TDTind> V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq ,B!TDTreq> V Tq = Tq.<A!TDTreq ,B?TDTind>))	false

Table A.II.5

Tcon_b_tran2(T) =

Condition	Value
T = Tp.Tq ∧Tp= < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ > .< _ ,B?TCONconf>) .< _ ,B!TDTreq>) ∧(Tq = <A?TDTind ,B!TDTreq> V Tq = <A?TDTind, _ > V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq> V Tq=Tq.<A?TDTind, _ > V Tq = Tq.<A?TDTind ,B?TDTind> V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq ,B!TDTreq> V Tq = Tq.<A!TDTreq ,B?TDTind>))	ture
T ≠ Tp.Tq ∧Tp= < _ ,B!TCONreq>.<A?TCONind, _ >.< A!TCONresp, _ > .< _ ,B?TCONconf>) .< _ ,B!TDTreq>) ∧(Tq = <A?TDTind ,B!TDTreq> V Tq = <A?TDTind, _ > V Tq = Tq.<A!TDTreq, _ > V Tq = Tq.< _ ,B?TDTind> V Tq = Tq.< _ ,B!TDTreq> V Tq=Tq.<A?TDTind, _ > V Tq = Tq.<A?TDTind ,B?TDTind> V Tq = Tq.<A?TDTind ,B!TDTreq> V Tq =Tq. <A!TDTreq ,B!TDTreq> V Tq = Tq.<A!TDTreq ,B?TDTind>))	false

Table A.II.6

Lost(T) =

Condition	Value
number of A!TDTreq ≠ number of B?TDTind V number of B!TDTreq ≠ number of A?TDTind	true
number of A!TDTreq = number of B?TDTind ∧number of B!TDTreq =number of A?TDTind	false

Table A.II.7

4>Equivalence

T.<A!TCONreq, _ > ≡

conditions	equivalences
T= <_,_>	<A!TCONreq,_>
T ≠ <_,_>	%T

Table A.II.8

T.< _ ,B?TCONind> ≡

conditions	equivalences
T= <A!TCONreq, _ >	<A!TCONreq, _ >.< _ , B?TCONind>
T ≠ <A!TCONreq, _ >	%T

Table A.II.9

T.<A!TDISreq ,B?TCONind> ≡

conditions	equivalences
T= <A!TCONreq, _ >	<A!TCONreq, _ >.<A!TDISreq, B?TCONind>
T ≠ <A!TCONreq, _ >	%T

Table A.II.10 (11.A)

T.< _ ,B!TCONresp> ≡

conditions	equivalences
T= <A!TCONreq, _ >.< _, B?TCONind>	<A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>
T= <A!TCONreq, _ >.<A!TDISreq, B?TCONind>	©T
T ≠ <A!TCONreq, _ >.< _, B?TCONind> ∧T≠ <A!TCONreq, _ >.<A!TDISreq, B?TCONind>	%T

Table A.II.11 (12.A)

T.<A!TCONreq, B!TCONresp> ≡

conditions	equivalences
T= <A!TCONreq, _ >.<A!TDISreq, B?TCONind>	©T
T≠ <A!TCONreq, _ >.<A!TDISreq, B?TCONind>	%T

Table A.II.12 (13.A)

T.<A?TCONconf, _ > ≡

conditions	equivalences
T=<A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>	<A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>.<A?TCONconf, _ >
T ≠ <A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>	%T

Table A.II.13

T.<A?TCONconf, B!TDISreq> ≡

conditions	equivalences
T=<A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>	<A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>.<A?TCONconf, B!TDISreq>
T ≠ <A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>	%T

Table A.II.14 (14.A)

T.<A?TCONconf, B!TDTreq> ≡

conditions	equivalences
T=<A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>	<A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>.<A?TCONconf, B!TDTreq>
T ≠ <A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>	%T

Table A.II.15 (6.A)

T.< _ , B!TCONreq> ≡

conditions	equivalences
T=< _ , _ >	< _ , B!TCONreq>
T ≠ < _ , _ >	%T

Table A.II.16

T.<A?TCONind, _ > ≡

conditions	equivalences
T=< _ , B!TCONreq>	< _ , B!TCONreq>.<A?TCONind, _ >
T ≠ < _ , B!TCONreq>	%T

Table A.II.17

T.<A?TCONind, B!TDISreq> ≡

conditions	equivalences
T=< _ , B!TCONreq>	< _ , B!TCONreq>.<A?TCONind, B!TDISreq>
T ≠ < _ , B!TCONreq>	%T

Table A.II.18

T.<A!TCONresp, _ > ≡

conditions	equivalences
T=< _ , B!TCONreq>.<A?TCONind, _ >	< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >
T=< _ , B!TCONreq>.<A?TCONind, _B!TDISreq>	©T
T ≠ < _ , B!TCONreq>.<A?TCONind, _ > ∧T≠ < _ , B!TCONreq>.<A?TCONind, _B!TDISreq>	%T

Table A.II.19

T.<A!TCONresp, B!TCONreq> ≡

conditions	equivalences
T=< _ , B!TCONreq>.<A?TCONind, _B!TDISreq>	©T
T≠ < _ , B!TCONreq>.<A?TCONind, _B!TDISreq>	%T

Table A.II.20

T.< _ ,B?TCONconf> ≡

conditions	equivalences
T=< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >	< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >.< _ ,B?TCONconf>
T ≠ < _ , B!TCONreq>.<A?TCONind, _ > ∽A!TCONresp, _ >	%T

Table A.II.21

T.<A!TDISreq, B?TCONconf> ≡

conditions	equivalences
T=< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >	< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >.<A!TDISreq, B?TCONconf>
T ≠ < _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >	%T

Table A.II.22

T.<A!TDTreq, B?TCONconf> ≡

conditions	equivalences
T=< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >	< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >.<A!TDTreq, B?TCONconf>
T ≠ < _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >	%T

Table A.II.23

T.<A!TDTreq, _ > ≡

conditions	equivalences
T=<A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>.<A?TCONconf, _ >	<A!TCONreq, _ >.< _ ,B?TCONind> .< _ , B!TCONresp>.<A?TCONconf, _ > .<A!TDTreq, _ >
T=<A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>.<A?TCONconf, B!TDISreq>	©T
T = Tcon_a_tran1 (T)	T
T = Tcon_a_tran2 (T)	T
T = Tcon_b_tran1 (T)	T
T = Tcon_b_tran2 (T)	T
T ≠ <A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>.<A?TCONconf, _ > ∧T ≠ <A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>.<A?TCONconf, B!TDISreq> ∧ ¬ Tcon_a_tran1 (T) ∧¬ Tcon_a_tran2 (T) ∧ ¬ Tcon_b_tran1 (T) ∧¬ Tcon_b_tran2 (T)	%T

Table A.II.24 (15.A)

T.< _ ,B!TDTreq> ≡

conditions	equivalences
T=< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >.< _ ,B?TCONconf>	< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >.< _ ,B?TCONconf> .< _ ,B!TDTreq>
T=< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >.<A!TDISreq, B?TCONconf>	©T
T = Tcon_a_tran1 (T)	T
T = Tcon_a_tran2 (T)	T
T = Tcon_b_tran1 (T)	T
T = Tcon_b_tran2 (T)	T
T ≠ < _, B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >.< _ ,B?TCONconf> ∧ T ≠ < _, B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >.<A!TDISreq, B?TCONconf> ∧ ¬ Tcon_a_tran1 (T) ∧ ¬ Tcon_a_tran2 (T) ∧ ¬ Tcon_b_tran1 (T) ∧ ¬ Tcon_b_tran2 (T)	%T

Table A.II.25

T.<A?TDTind, _ > ≡

conditions	equivalences
T = Tcon_a_tran1 (T)	T
T = Tcon_a_tran2 (T)	T
T = Tcon_b_tran1 (T)	T
T = Tcon_b_tran2 (T)	T
¬ Tcon_a_tran1 (T) ∧ ¬ Tcon_a_tran2 (T) ∧ ¬ Tcon_b_tran1 (T) ∧ ¬ Tcon_b_tran2 (T)	%T

Table A.II.26

T.< _ ,B?TDTind> ≡

conditions	equivalences
T = Tcon_a_tran1 (T)	T
T = Tcon_a_tran2 (T)	T
T = Tcon_b_tran1 (T)	T
T = Tcon_b_tran2 (T)	T
¬ Tcon_a_tran1 (T) ∧ ¬ Tcon_a_tran2 (T) ∧ ¬ Tcon_b_tran1 (T) ∧ ¬ Tcon_b_tran2 (T)	%T

Table A.II.27

T.<A?TDTind, B?TDTind> ≡

conditions	equivalences
T = Tcon_a_tran1 (T)	T
T = Tcon_a_tran2 (T)	T
T = Tcon_b_tran1 (T)	T
T = Tcon_b_tran2 (T)	T
¬ Tcon_a_tran1 (T) ∧ ¬ Tcon_a_tran2 (T) ∧ ¬ Tcon_b_tran1 (T) ∧ ¬ Tcon_b_tran2 (T)	%T

Table A.II.28

T.<A!TDTreq, B?TDTind> ≡

conditions	equivalences
T = Tcon_a_tran1 (T)	T
T = Tcon_a_tran2 (T)	T
T = Tcon_b_tran1 (T)	T
T = Tcon_b_tran2 (T)	T
¬ Tcon_a_tran1 (T) ∧ ¬ Tcon_a_tran2 (T) ∧ ¬ Tcon_b_tran1 (T) ∧ ¬ Tcon_b_tran2 (T)	%T

Table A.II.29

T.<A!TDTreq, B!TDTreq> ≡

conditions	equivalences
T = Tcon_a_tran1 (T)	T
T = Tcon_a_tran2 (T)	T
T = Tcon_b_tran1 (T)	T
T = Tcon_b_tran2 (T)	T
¬ Tcon_a_tran1 (T) ∧ ¬ Tcon_a_tran2 (T) ∧ ¬ Tcon_b_tran1 (T) ∧ ¬ Tcon_b_tran2 (T)	%T

Table A.II.30

T.<A?TDTind, B?TDTind> ≡

conditions	equivalences
T = Tcon_a_tran1 (T)	T
T = Tcon_a_tran2 (T)	T
T = Tcon_b_tran1 (T)	T
T = Tcon_b_tran2 (T)	T
¬ Tcon_a_tran1 (T) ∧ ¬ Tcon_a_tran2 (T) ∧ ¬ Tcon_b_tran1 (T) ∧ ¬ Tcon_b_tran2 (T)	%T

Table A.II.31

T.<A?TDTind, B!TDTreq> ≡

conditions	equivalences
T = Tcon_a_tran1 (T)	T
T = Tcon_a_tran2 (T)	T
T = Tcon_b_tran1 (T)	T
T = Tcon_b_tran2 (T)	T
¬ Tcon_a_tran1 (T) ∧ ¬ Tcon_a_tran2 (T) ∧ ¬ Tcon_b_tran1 (T) ∧ ¬ Tcon_b_tran2 (T)	%T

Table A.II.32

T.<A!TDISreq, _ > ≡

conditions	equivalences
(T= Tp.E ∧ Canonical (T) = Canonical (Tp) ∧ E = <A?TDTind, _ >) V (T= <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf, _ >) V (T= <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf, B!TDISreq>) V (T= < _ , B!TCONreq>.<A?TCONind, B!TDISreq>)	T.<A!TDISreq, _ >
(T= Tp.E ∧ Canonical (T) = Canonical (Tp) ∧ E = <A?TDTind, B!TDTreq>) V (T= <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf,B!TDTreq>)	©T
(T= Tp.E ∧ Canonical (T) = Canonical (Tp) ∧ E ≠ <A?TDTind, _ > ∧ E ≠ <A?TDTind, B!TDTreq> ∧ E ≠<A?TDTreq, B!TDTreq>) ∧ (T≠ <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf, _ >) ∧ (T≠ <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf.B!TDISreq>) ∧ (T≠ <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf.B!TDTreq>) ∧ (T≠ < _ , B!TCONreq>.<A?TCONind, B!TDISind>)	%T

Table A.II.33 (10.A, 16.A)

T.<A?TDISind(ts), _ > ≡

conditions	equivalences
¬ Lost (T) ∧ (T = Tp.E ∧ E = < _ , B!TDISreq>)	<_,_>
¬ Lost (T) ∧ T = <A!TCONreq, _ >.< _ ,B?TCONind> .< _ ,B!TCONresp>.<A?TCONconf, B!TDISreq>	<_,_>
¬ Lost (T) ∧ T = < _ ,B!TCONreq>.<A?TCONind, B!TDISreq>	<_,_>
Lost (T) V (T = Tp.E ∧ E ≠ < _ , B!TDISreq>) ∧ (T ≠ <A!TCONreq, _ >.< _ ,B?TCONind> .< _ ,B!TCONresp>.<A?TCONconf, B!TDISreq> ∧ (T ≠ < _ ,B!TCONreq>.<A?TCONind, B!TDISreq>)	%T

Table A.II.34

T.<A?TDISind(tp), B?TCONind> ≡

conditions	equivalences
(T =<A!TCONreq, _ >)	<A!TCONreq, _ >.<A?TDISind(tp), B?TCONind>
(T ≠<A!TCONreq, _ >)	%T

Table A.II.35 (19.A)

T.<A?TCONconf, B?TDISind(tp)> ≡

conditions	equivalences
T = <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>	<A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp> .<A?TCONconf, B?TDISind(tp)>
T ≠ <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>	$\sigma_c \cdot \Gamma$

Table A.II.36 (24.A)

T.<A?TDISind(tp), B?TDTreq> ≡

conditions	equivalences
T=<A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>.<A?TCONconf, B!TDTreq>	©T
T = Tp.E Canonical (Tm) = Canonical (Tp) ∧ (E = <A?TDTind, B!TDTreq> V E = <A?TDTind, B?TDTind>)	©T
T ≠ <A!TCONreq, _ >.< _ ,B?TCONind> < _ , B!TCONresp>.<A?TCONconf, B!TDTreq > ∧T = Tp.E Canonical (Tm) = Canonical (Tp) ∧ E ≠ <A?TDTind, B!TDTreq> ∧ E ≠ <A?TDTind, B?TDTind>)	%T

Table A.II.37 (22.A)

T.<A?TDISind(tp), B?TDTind> ≡

conditions	equivalences
T= Tp.E ∧ (E=<A!TDTreq, _ > V E=<A!TDTreq, B?TDTind>)	T.<A?TDISind(tp), B?TDTind>
T= Tp.E ∧ E=<A!TDTreq, B!TDTreq>)	©T
T= Tp.E ∧ E≠<A!TDTreq, _ > ∧ E≠<A!TDTreq, B?TDTind> ∧ E=<A!TDTreq, B!TDTreq>	%T

Table A.II.38 (23.A, 26.A)

T.<A?TDISind(tp), B?TDISreq> ≡

conditions	equivalences
(T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧ E = < _ ,B?TDTind>) V (T= < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDISreq, B?TCONconf>) V (T =<A!TCONreq, _ >.< A!TDISreq, B?TCONind>)	< _ , _ >
(T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧(E = <A!TDTreq ,B?TDTind> V E = <A!TDTreq ,B!TDTreq>) V (T= < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDTreq, B?TCONconf>)	©T
(T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧ E ≠ < _ ,B?TDTind> ∧E ≠ <A!TDTreq ,B?TDTind> ∧E ≠ <A!TDTreq ,B!TDTreq>) ∧(T≠ < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDISreq, B?TCONconf>) ∧(T≠ < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDTreq, B?TCONconf>) ∧(T≠ <A!TCONreq, _ >.< A!TDISreq, B?TCONind>)	%T

Table A.II.39

T.<A?TDISind(tp), B?TDISind> ≡

conditions	equivalences
(T= Tp.E ∧E=<A!TDISreq, _ >)∧¬ Lost(T)	< _ , _ >
(T= Tp.E ∧ E≠<A!TDISreq, _ >) ∨ Lost(T)	%T

Table A.II.40

T.<A?TDISind(tp), _ > ≡

conditions	equivalences
(T= Tp.E ∧(E=<A!TCONreq, _ > V E = <A?TCONconf, _ > VE=<A!TCONind, _ > VE=<A!TCONind, B?TDISind(tp)> V E = <A?TCONconf, B?TDISind(tp)> V E=<A!TDTind, B?TDISind(tp)> V E=<A!TDISind, B?TDISind(tp)>) ∧¬ Lost (T)	< _, _ >
(T= Tp.E ∧(E≠<A!TCONreq, _ > V E≠ <A?TCONconf, _ > VE≠<A!TCONind, _ > VE≠<A!TCONind, B?TDISind(tp)> V E ≠<A?TCONconf, B?TDISind(tp)> V E≠<A!TDTind, B?TDISind(tp)> V E≠<A!TDISind, B?TDISind(tp)>)	%T

Table A.II.41

T.< _ ,B!TDISreq> ≡

conditions	equivalences
(T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧ E = < _ ,B?TDTind>) V (T= < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ > .< _ , B?TCONconf>) V (T= < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDISreq, B?TCONconf>) V (T =<A!TCONreq, _ >.< A!TDISreq, B?TCONind>)	T.< _ , B!TDISreq>
(T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧E = <A!TDTreq ,B?TDTind>) V (T= < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDTreq, B?TCONconf>)	©T
(T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧ E ≠ < _ ,B?TDTind> ∧E ≠ <A!TDTreq ,B?TDTind> ∧E ≠ <A!TDTreq ,B!TDTreq>) ∧(T≠ < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ > .< _ , B?TCONconf>) ∧(T≠ < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDISreq, B?TCONconf>) ∧(T≠ < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDTreq, B?TCONconf>) ∧(T≠ <A!TCONreq, _ >.< A!TDISreq, B?TCONind>)	%T

Table A.II.42

T.< _ ,B?TDISind(ts)> ≡

conditions	equivalences
¬ Lost (T) ∧(T = Tp.E ∧ E = <A!TDISreq, _ >)	< _ , _ >
¬ Lost (T) ∧T = < _ , B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.<A!TDISreq, B?TCONconf>	< _ , _ >
¬ Lost (T) ∧T = <A!TCONreq, _ >.<A!TDISreq, B?TCONind>	< _ , _ >
Lost (T) V (T = Tp.E ∧ E ≠ <A!TDISreq, _ >) ∧(T ≠ < _ , B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.<A!TDISreq, B?TCONconf> ∧(T ≠ <A!TCONreq, _ >.<A!TDISreq, B?TCONind>)	%T

Table A.II.43

T.<A?TCONind, B?TDISind(tp)> ≡

conditions	equivalences
T =< _ ,B!TCONreq>	< _ ,B!TCONreq> .<A?TCONind, B?TDISind(tp)>
T ≠< _ ,B!TCONreq>	%T

Table A.II.44

T.<A?TDISind(tp), B?TCONconf> ≡

conditions	equivalences
T = < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >	< _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.<A?TDISind(tp), B?TCONconf>
T ≠ < _ ,B!TCONreq>.<A?TCONind, _ > . <A!TCONresp, _ >	%T

Table A.II.45

T.<A!TDTreq, B?TDISind(tp)> ≡

conditions	equivalences
T=< _ , B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >.< A!TDTreq ,B?TCONconf>	©T
T=Tp.E ∧ Canonical (Tm) = Canonical (Tp) ∧ (E=<A!TDTreq, B?TDTind> ∨ E = <A?TDTind, B?TDTind>)	©T
T ≠ < _, B!TCONreq>.<A?TCONind, _ > <A!TCONresp, _ >.< A!TDTreq ,B?TCONconf> ∧ T = Tp.E ∧ Canonical (Tm) = Canonical (Tp) ∧ (E ≠<A!TDTreq, B?TDTind> ∧ E ≠ <A?TDTind, B?TDTind>)	%T

Table A.II.46

T.<A?TDTind, B?TDISind(tp)> ≡

conditions	equivalences
T= Tp.E ∧ E=< _ ,B!TDTreq>	T.<A?TDTind, B?TDISind(tp)>
T= Tp.E ∧ E≠< _ ,B!TDTreq>	%T

Table A.II.47

T.<A?TDISreq, B?TDISind(tp)> ≡

conditions	equivalences
(T= Tp.E ∧ Canonical (T) = Canonical (Tp) ∧ E = <A?TDTind, _ >) V (T= <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf, B!TDISreq>) V (T= < _ , B!TCONreq>.<A?TCONind, B!TDISind>)	< _ , _ >
(T= Tp.E ∧ Canonical (T) = Canonical (Tp) ∧ (E = <A!TDTreq, B?TDTind> V E = <A?TDTind, B?TDTind>) V (T= <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf,B!TDTreq>)	©T
(T= Tp.E ∧ Canonical (T) = Canonical (Tp) ∧ E ≠ <A?TDTind, _ > ∧ E ≠ <A?TDTind, B!TDTreq> ∧ E ≠<A?TDTreq, B!TDTreq>) ∧ (T≠ <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf.B!TDISreq>) ∧ (T≠ <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf.B!TDTreq>) ∧ (T≠ < _ , B!TCONreq>.<A?TCONind, B!TDISind>)	%T

Table A.II.48

T.<A?TDISind(ts), B?TDISind(tp)> ≡

conditions	equivalences
(T= Tp.E ∧ E=< _ , B!TDISreq>) ∧ ¬ Lost(T)	< _ , _ >
(T= Tp.E ∧ E≠< _ ,B!TDISreq>) V Lost(T)	%T

Table A.II.49

T.< _ , B?TDISind(tp)> ≡

conditions	equivalences
(T= Tp.E ∧ (E=< _ ,B!TCONreq> V E = < _ ,B?TCONconf> VE=< _ ,B!TCONind V E=<A?TDISind(tp), B!TCONind> V E = <A?TDISind(tp), B?TCONconf> V E=<A?TDISind(tp), B!TDTind> V E=<A?TDISind(tp), B!TDISind>) ∧ ¬ Lost (T)	< _ , _ >
(T= Tp.E ∧ (E≠< _ ,B!TCONreq> V E ≠< _ ,B?TCONconf> VE≠< _ ,B!TCONind V E≠<A?TDISind(tp), B!TCONind> V E≠<A?TDISind(tp), B?TCONconf> V E≠<A?TDISind(tp), B!TDTind> V E≠<A?TDISind(tp), B!TDISind>)	%T

Table A.II.50

T.<A!TCONreq, B!TCONreq> ≡

conditions	equivalences
T= <_,_>	<A!TCONreq, B!TCONreq>
T ≠ <_,_>	%T

Table A.II.51

T.<A!TCONind, B!TCONind> ≡

conditions	equivalences
T= <A!TCONreq, B!TCONreq>	©T
T ≠ <A!TCONreq, B!TCONreq>	%T

Table A.II.52

T.<A!TDISreq, B!TDISreq> ≡

conditions	equivalences
T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧ (E = <A?TDTind ,B?TDTind>)	T.<A!TDISreq, B!TDISreq>
(T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧ (E = <A!TDTreq ,B?TDTind> ∨ E = <A!TDTreq ,B!TDTreq>) ∨ (T= = <A?TCONreq, B?TCONreq>) ∨ (T= < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDTreq, B?TCONconf>) ∨ (T= <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf,B!TDTreq>)	©T
(T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧ E ≠ <A?TDTind, B?TDTind> ∧ E ≠ <A!TDTreq ,B!TDTreq>) ∧ (T ≠ <A?TCONreq, B?TCONreq>) ∧ (T≠ < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDISreq, B?TCONconf>) ∧ (T≠ < _ ,B!TCONreq>.<A?TCONind, _ > .<A!TCONresp, _ >.< A!TDTreq, B?TCONconf>) ∧ (T≠ <A!TCONreq, _ >.< A!TDISreq, B?TCONind>) ∧ (T≠ <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf.B!TDISreq>) ∧ (T≠ <A!TCONreq, _ >.< _ , B?TCONind> .< _ , B!TCONresp>.<A?TCONconf.B!TDTreq>) ∧ (T≠ < _ , B!TCONreq>.<A?TCONind, B!TDISind>)	%T

Table A.II.53

T.<A!TDISind(ts), B!TDISind(ts)> ≡

conditions	equivalences
(T = Tp.E ∧ E= <A!TDISreq, B!TDISreq>) ∧¬ Lost(T)	< _ , _ >
(T = Tp.E ∧ E≠ <A!TDISreq, B!TDISreq>) ∨ ¬ Lost(T)	%T

Table A.II.54

T.<A!TDISind(tp), B!TDISind(tp)> =

conditions	equivalences
T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧ (E = <A?TDTind ,B?TDTind>) ∧¬ Lost(T)	<_ , _>
T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧ (E = <A?TDTind ,B?TDTind>) ∧ Lost(T)	%T
T= Tp.E ∧ Canonical(T) = Canonical (Tp) ∧ (E ≠ <A?TDTind ,B?TDTind>)	©T

Table A.II.55

5> Temporal Assertions (Liveness Properties only):

Note: ϕ stands for any event which we do not care in specification

i) [] (<A!TCONreq, _ > => <> (< _ ,B?TCONind> V <A?TDISind(tp), _>))

ii) [] (< _ ,B!TCONresp> => <> (<A?TCONconf, _ > V <A?TDIS(tp), B.ϕ>))

iii) [] (<A!TDTreq, _ > => <> (< _ ,B?TDTind> V <A.ϕ, B?TDISreq(tp)>))

iv) [] (< _ ,B!TDTreq > => <> (<A?TDTind, _ > V <A?TDIS(tp), B.ϕ>))

v) [] (<A!TDISreq, _ > => <> < _ ,B?TDTind(ts)> V <A.ϕ, B?TDISreq(tp)>))

vi) [] (< _ ,B!TDISreq> => <> <A?TDISind(ts), _ > V <A?TDIS(tp), B.ϕ>))

(end of specification of global view of TS service)