12

Framework for formal methods in conformance testing

Prof. Dr. Dieter Hogrefe

Universität Bern, Institut für Informatik, Länggassstrasse 51,

CH-3012 Bern, Switzerland, e-mail: hogrefe@iam.unibe.ch

This paper presents the latest developments in the Formal Methods in Conformance Testing (FMCT) project of ITU and ISO. The project has been initiated to study the role of formal description techniques in the conformance testing process. The goal is to develop a standard that defines the meaning of conformance in the context of FDTs. A general framework which has been developed in the FMCT project will be presented.

1. INTRODUCTION

The primary objectives of standardization is to allow systems developed by different vendors to work together, to exchange and handle information. Conformance testing, i.e. the assessment of a product on conformance with it's specification, is an important issue to compatibility. To achieve this, a standard way has to be defined to look at products. This has been done in a joint project of the standardization organizations ISO and ITU which has been started around 1983. The effort resulted in a standard called „Conformance Testing Methodology and Framework (CTMF)" [9].

CTMF is a very general framework which should be applicable to the widest possible range of specifications and products.

Recently, more and more specifications in the communications world are being made by use of a formal description technique (FDT). This makes it necessary to define the meaning of conformance in this formal context.

ISO and ITU-T started another joint project around 1989, called project 1.21.54 in ISO and question 8/10 in ITU-T. This new effort should result in a standard on „Formal Methods in Conformance Testing (FMCT)" [3]. The current planing of the project extends to May 1997. At this date the publication of an International Standard in ISO and the respective standard in ITU-T is desirable. This means that there needs to be a Committee Draft around middle of 1995 and a Draft International Standard around 1996. Two main meetings of the project are planned per year, but there may be additional meetings in the subgroups.

Given a formal specification, how can one find out whether an implementation conforms to this specification?

The motivation of this activity is the existence of formally described standards and therefore the need to define the meaning of conformance with respect to formal specifications.

T. Mizuno et al. (eds.), *Protocol Test Systems*
© Springer Science+Business Media Dordrecht 1995

As FMCT is a very general problem in software engineering, it's principles should be applicable to any application area, in particular any application area in communications, e.g. OSI, ODP and IBCN.

Since CTMF defines conformance in a very general informal sense, it serves as a natural basis for the more formal work on FMCT. By nature there is a close relationship between CTMF and FMCT. In particular, the concepts and their definitions ought to be compatible between the two documents.

One has to distinguish between three different types of concepts:

a) concepts of CTMF not subject to FMCT

b) concepts of FMCT with no relationship to CTMF

c) concepts shared by FMCT and CTMF

Concepts of type a) of course remain valid in the FMCT framework. There is no additional formal interpretation of these concepts.

Category b) contains concepts which exist due to the formal nature of FMCT. In many cases these concepts have not been handled by CTMF because they have a generally agreed upon informal meaning sufficient in the CTMF context and can be looked up in standard dictionaries. Examples are the terms verification and validation. In a formal context, though, these terms need further precision, specifically adapted to formal specification and testing of communication protocols.

Category c) contains those concepts which have been defined in the CTMF context but need further interpretation in FMCT. Of course, the formal interpretation needs to be consistent with the informal one. The following example shall illustrate the definition of concepts in category c).

A protocol implementation conformance statement (PICS), according to CTMF, is a statement made by the supplier of an OSI implementation or system, stating which capabilities have been implemented, for a given OSI protocol.

In the formal context, PICS defines actual values for the parameters of the formal specification *S* for a given implementation I. A particular PICS indirectly determines a subset of the conformance requirements defined for *S*.

The FMCT project is currently maintaining a working document which ultimately should become the standard or recommendation, respectively.

The main part of the document contains the definition of the terminology which is used to define the framework for formal methods in conformance testing.

Besides the main part there are 2 rather elaborated annexes.

Annex A explains the relationship between the FMCT concepts and the existing FDTs: Estelle [8], LOTOS [7] and SDL [10]. The annex explains for example the concept of the implementation relation w.r.t. the three languages and gives an interpretation of PICS proforma and PICS.

Annex B deals with test generation methods as they have been developed for finite state machine and other models, in particular [1], [11], [2], [13], and [6]. A common example, the INRES protocol and service [5] is used to show the applicability of existing test generation

methods to an OSI protocol specification. INRES is not a real protocol, but it contains many features of OSI protocols. It is an abridged version of the Abracadabra protocol used in [12] for illustration.

Besides these annexes there are two other activities reflected in Annex C and D on specification styles for testability and measures for test suite coverage respectively. These are activities for further study at the moment.

The elements described in this paper have been carefully developed by investigation of a number of formal test case generation methods and tools some of which are briefly described in Annex B of [3].

The following sections will give an overview of the main ideas of [3].

2. BASELINE

In order to be able to establish a formal relationship between a specification and an implementation under test it is necessary to have a complete and completely formal specification of an IUT. Usually this is not given. The normal case is to have a specification consisting of a mixture of description techniques, such as prose text, MSC, FDTs, ASN.1, etc. Furthermore the system specification may make reference to other specification, e.g. standards, without explicitly incorporating them. In these cases a formal specification of the IUT has to be developed before conformance can be checked by formal means. Fig. 1 shows that scenario.

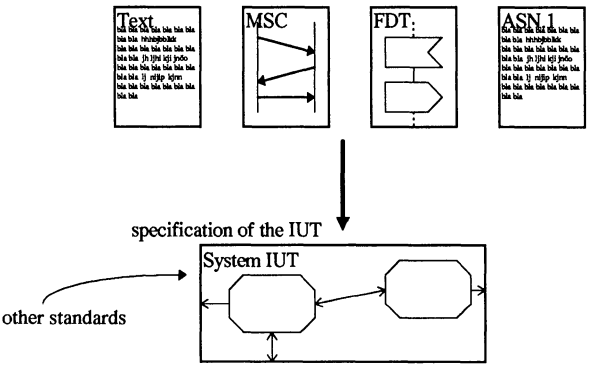

Fig.1: A formal and complete specification of the IUT is needed

When we talk about a specification in the following, we always talk about a complete formal specification of the IUT. During the development process of this specification it may have been necessary to choose among options. If the FDT allows it, it may be possible to develop a more general **parameterized specification**. This does no longer specify one particular IUT but a set of IUTs according to the different combinations of options which are possible. In order to get the specification of a particular IUT it is necessary to instantiate the parameterized specification with actual values for the formal parameters. This is then called it

an **instantiated specification**. If in the following the term **specification** is used, usually an instantiated specification is meant.

Under these assumptions a formal meaning can be assigned to the term „conformance".

3. FORMAL MEANING OF CONFORMANCE

We will develop a framework in this section which can be used to characterize „conformance" precisely. We will see that there is not only a single „conformance" between a specification and an implementation, but that there are many different possibilities to define it, depending on what degree of similarity one wants between the specification and the implementation.

The set of instantiated specifications is denoted by *SPECS*.

An implementation consists of hardware devices or a combination of hardware and software that is executed by it. An implementation has physical connectors or programming interfaces for communication with other implementations.

The set of implementations is called *IMPS*.

The elements of *SPECS* are mathematical objects while the elements of *IMPS* are not. Therefore it is not possible to define a formal relation between specifications and implementations.

We need to transform an implementation into a mathematical object. It is assumed (**test assumption**) that any implementation *IUT* can be modeled by an element m_{IUT} in a formalism *MODS* (e.g. labeled transition systems, finite state machines). The model m_{IUT} of an implementation *IUT* may simply be the set of traces that can be observed while experimenting with *IUT*.

Usually it is not possible to actually calculate m_{IUT} for a particular *IUT*. We only need this concept to define the meaning of conformance. The formalism *MODS* may be the same as *IMPS*. We will later see an example where for both SDL is used. This may facilitate the comparison of both.

For a particular choice of *MODS* one implementation may have many different models in *MODS*. The test assumption implies that these cannot be distinguished by testing. Therefore it is sufficient to model an implementation *IUT* by a single model $m_{IUT} \in MODS$.

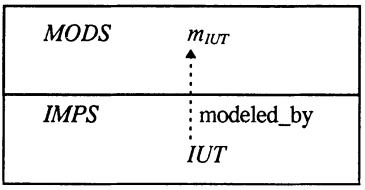

Fig. 2: The relation between elements of *IMPS* and *MODS*

Conformance between an implementation and a specification consists of two parts: static conformance and dynamic conformance.

In the FMCT context **static conformance** involves the correct instantiation of a parameterized specification such that the actual values of the parameters are an allowed combination of implementation options according to the ICS proforma (see [9]).

Dynamic conformance involves the permitted observable behavior of an implementation in instances of communication as described by the specification. Dynamic conformance between an implementation and a specification is formally characterized by a relation between the model of the implementation and the specification. This relation is called an **implementation relation**. It will be denoted as imp with the following signature:

$\text{imp} \subseteq MODS \times SPECS$

This relation is not a priori given just by the knowledge of *MODS* and *SPECS*. We will see in the following section by an example, that for one formalism, SDL in the example, there can be different meaningful implementation relations. Which one to choose may be application dependent. There has been no implementation relation proposed so far which is accepted as the universal one under all circumstances.

The lesson to learn from this is that it is not sufficient just to give a formal specification, but it is also necessary to give the implementation relation. An implementation *IUT* conforms dynamically to an instantiated specification s with respect to the relation imp, if the model m_{IUT} of *IUT* relates to s as m_{IUT} imp s. In this case m_{IUT} is a conforming model of s with respect to imp.

One specification can have many conforming implementations. For a specification $s \in SPECS$ and an implementation relation imp the set M_s denotes the set of all conforming models in *MODS*:

$M_s = \{m \in MODS \mid m \text{ imp } s\}$

Fig. 3 shows how an instantiated specification $s \in SPECS$ determines a set of conforming implementations I_s. The set I_s denotes the set of implementations that can be modeled by models in M_s. Therefore the set I_s is the set of implementations that implement s correctly.

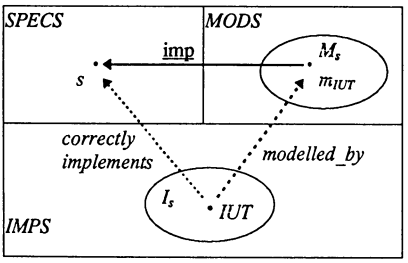

Fig. 3: Relations between *IMPS*, *MODS* and *SPECS*

4. EXAMPLE

In this section we will give an example to illustrate some of the concepts introduced in the previous section.

Figures 8-12 show an SDL specification. It is part of the initiator side of the INRES protocol [5]. the specification is not parameterized in order to keep the example simple. For an example of a parameterized specification see [3].

Only the external behavior is considered in the context of conformance. An SDL specification does not explicitly express the external behavior. Therefore to test for dynamic conformance a representation of the observable behavior derived from the SDL specification is needed. A suitable representation for the external behavior for an SDL specification is by an ACT (Asynchronous Communication Tree, see [4]) or an LTS (Labeled Transition System).

If SDL is used for specification of systems the set *SPECS* is the set of all ACTs or LTSs which can be derived from any SDL specification. For the example a small part of an ACT representation of the observable behavior is illustrated in Fig. 4. The ACT representation associates to each state information signals in the different channels. In the figure only channels that convey a signal in the current state are shown. If all channels of the system are empty this is denoted $Q = \langle\rangle$. The state information is however not needed when test cases are defined from the ACT representation.

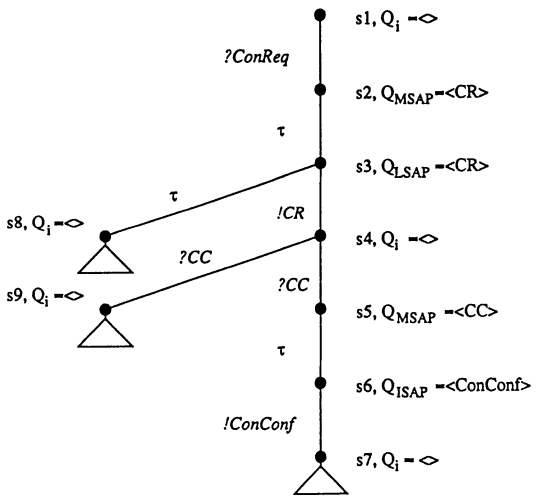

Figure 4: An ACT representation of part of the observable behavior

The concepts of the formal framework are now illustrated by giving three different implementations I_1, I_2 and I_3. The implementations are represented by their models from the set *MODS* for which SDL is used. Please note that the SDL specifications I_1, I_2 and I_3 actually define an ACT, i.e. a number of traces. SDL is only used to define these traces.

I_1 is an implementation that has a model that coincide with the model of the specification. This means that the block description of the protocol shown in Figures 9 and 10 can be seen as a model for this implementation.

I_2 implements the protocol as specified in Figure 10 as well, except for the behavior that can be expressed in SDL as shown in Figure 5. The implementation sends a signal *DT* instead of a *CR* when a connect request is received.

I_3 similarly implements the protocol as defined in the specification. In addition in state *connected* it may receive a new connect request and initiate again the connection phase as shown in Figure 5.

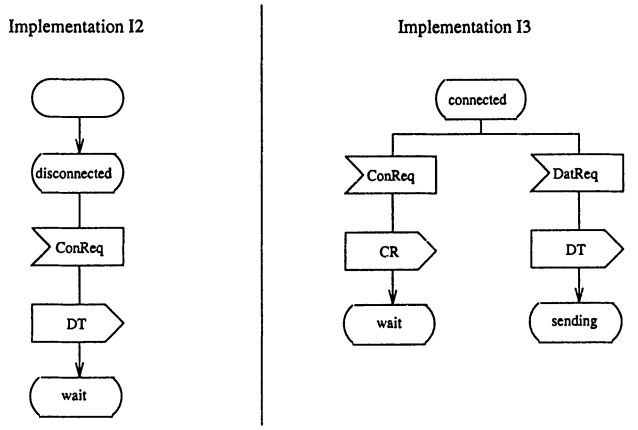

Figure 5: The changed behavior of implementations I_2 and I_3

The criterion for conformance of an implementation and an SDL specification is defined in terms of an implementation relation imp between the model of an implementation and a specification.

Several of the implementation relations defined are based on the existence of similar sequences of events in the implementation and the specification. Sequences of observable events can be derived from an ACT representation of an SDL specification. A sequence of observable events is denoted a trace and a trace set *Tr(S)* denotes the set of all possible traces of a specification *S*. A trace of the ACT shown in Figure 4 is *<ConReq,CR,CC,ConConf>*.

Only a very limited class of SDL specifications specify a behavior that is expressed by a finite set of traces. The reason is the unbounded queue property of an SDL channel. Even for simple systems specified in SDL it should be possible to send any number of a signal conveyed by the channel from the environment to the system. E.g. in the example specification it should

be possible to send any number of the signal *ConReq* to the system. Hence in practice exhaustive test of the observable behavior of an SDL system is not possible. Still the trace models are useful in the definition of a formal requirement for conformance of the dynamic behavior of an implementation.

A relation that is often used as an implementation relation is the trace inclusion relation $\leq_{tr}$ (trace preorder). Two trace sets T_1 and T_2 satisfy the relation $T_1 \leq_{tr} T_2$ if and only if T_1 is a subset of T_2.

The trace inclusion relation may be used as an implementation relation in two ways dependent on how the requirements of the specification are interpreted. If the specification is assumed to specify the maximal allowed behavior of an implementation, the trace set of a conforming implementation is a subset of the trace set of the specification. For a conforming implementation I and a specification S this is denoted $I \leq_{tr} S$ or $(I,S) \in \leq_{tr}$.

The other interpretation of requirements specified by an SDL specification is that it defines the minimal required behavior of an implementation. In this case the trace set of a specification S is a subset of a conforming implementation I, $I \geq_{tr} S$.

The maximal allowed behavior implementation relation implies a very weak requirement on a conforming implementation. An implementation that cannot perform any external action is a conforming implementation of any specification, as the empty set is a subset of any trace set. It is not possible to test for the minimal required behavior implementation relation due to the infinite trace sets of most SDL specifications.

In Table 1 it is shown which implementations are conformant with respect to the specified example (here denoted S) and the two implementation relations.

(Impl,Spec) satisfies	$\leq_{tr}$	$\geq_{tr}$
(I_1,S)	true	true
(I_2,S)	false	false
(I_3,S)	false	true

Table 1: Overview of the conformance of I_1, I_2 and I_3 with respect to different implementation relations

As the trace set of implementation I_1 is identical to that of the specification both the maximal allowed behavior and minimal required behavior relation are satisfied for I_1. Implementation I_2 does not satisfy $\leq_{tr}$ as the trace <*ConReq*,*DT*> is not a member of the trace set of the specification. Similarly, the implementation relation $\geq_{tr}$ is not satisfied either, as the trace set of I_2 does not include the trace <*ConReq*,*CR*>. For implementation I_3 the implementation relation $\leq_{tr}$ is not satisfied as the implementation can for example perform the trace <*ConReq*,*CR*,*CC*,*ConConf*,*ConReq*,*CR*>, that is not a trace of the specification. As I_3 can perform every trace of the specification it conforms according to the implementation relation $\geq_{tr}$.

The model of an implementation is not known in advance for conformance testing. It can only be approximated by performing experiments on the implementation and observing responses. Non-deterministic system specifications make it impossible to ensure that an implementation model is a complete description of the possible behavior.

In the example specification it may not be possible to determine if an implementation can perform a specific trace. This is the case for trace <*ConReq*,*CR*> . The unreliable channel may always discard the signal *CR* such that it never occurs as an observable event in the environment. However it is not possible to derive from a number of experiments in which the signal *CR* has not been observed whether the trace has been implemented. So the trace preorder implementation relations may provide a sound basis as a conformance criterion only if additional assumption are made on the specification and/or implementation. For instance it may be assumed that information is provided on how non-determinism of the specification is resolved in the implementation.

5. CONFORMANCE REQUIREMENTS

In the previous sections conformance has been defined in an abstract way by means of implementation relations on the sets *MODS* and *SPECS*.

However, in [9] the definition of conformance is based on the more concrete concept of conformance requirements. This second characterization of conformance is a possible refinement of the previous definition, and both approaches define the same concept with the same expressive power.

[9] distinguishes two types of conformance requirements: static and dynamic. The meaning is analog to the distinction of static and dynamic conformance introduced above. In the FMCT context static conformance requirements are requirements on the possible combinations of values of the formal parameters of the specification. Dynamic conformance requirements are requirements implied by the formal specification and the implementation relation. Both static and dynamic conformance requirements are properties that need to be satisfied by (the model of) an implementation in order to conform.

In a formal context conformance requirements are expressed in a formal requirement language. This may for example be temporal logic. *REQS* denotes the set of all requirements that can be expressed in a particular language. In the requirements approach an instantiated specification $s \in SPECS$ is expressed as a set of requirements $R_s \subseteq REQS$. The set of all possible specifications is the powerset of the set of possible requirements: $SPECS = P(REQS)$. One element $r \in R_s$ represents a single dynamic conformance requirement. In general. the set R_s can be infinite.

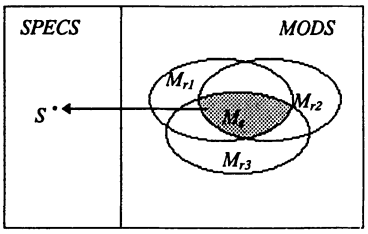

Figure 6: Conformance requirements and conforming implementations

The approach using conformance requirements to characterize conformance is closer to the methodology defined in [9] than the approach using implementation relations.

Dynamic conformance between an implementation and a specification in the requirement approach is formally characterized by a relation between the model of the implementation and the specification. This relation is called a **satisfaction relation**. It will be denoted as sat, where sat has the signature:

$\underline{\text{sat}} \subseteq MODS \times SPECS$

An implementation *IUT* conforms dynamically to specification $s \in SPECS$ if the model m_{IUT} satisfies all conformance requirements in R_s. The set Ms of models of conforming implementations in the requirements approach is given by

$M_s = \{m \in MODS \mid \forall r \in R_s : m \ \underline{\text{sat}}\ r\}$

For a particular conformance requirement $r_i \in R_s$, the set M_{ri} denotes the set of all models in *MODS* satisfying requirement r_i, i.e. $M_{ri} = \{m \in MODS \mid m \ \underline{\text{sat}}\ r_i\}$. Figure 6 shows how intersection of the sets M_{ri} determines the set M_s of conforming implementations.

6. TEST ARCHITECTURE

Since conformance testing involves experimenting with and observing the IUT it is necessary to consider the way how the IUT can be accessed. If, for example, the IUT can only be accessed by means of a channel that acts as a queue with delay the control and observation is different from as if the IUT could be accessed directly (synchronously, e.g. by function call). To take this into account, a test context has to be defined through which the IUT is accessed. We have to distinguish between the PCOs and IAPs as depicted in Fig. 7.

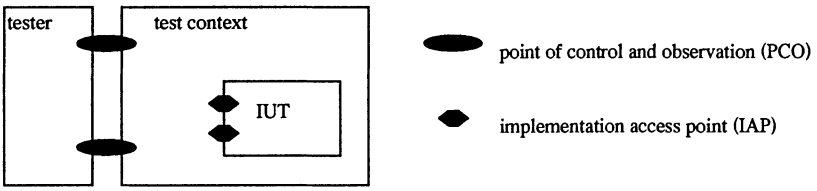

Figure 7: The test architecture

The tester is the implementation of a test suite. It is the entity that carries out the experiments by executing the test cases and observing the results. The tester communicates with the test context via the PCOs, and indirectly with the IUT via the test context. It can be structured into an Upper Tester and a Lower Tester according to [9]. The experiments that constitute the test suite are called test cases. The execution of a test case leads to a verdict. The formal specification of a test case t is expressed in a formal language called the test notation and denoted by *TESTS*: $t \in TESTS$. It follows that a test suite, being a set of test cases, is in *P(TESTS)*, the powerset of *TESTS*.

The test context is the system in which the IUT is embedded, and via which the IUT communicates with the tester. It relates events that occur at the PCOs in the communication between the test context and the tester, to events that occur at the IAPs in the communication between the test context and the IUT. The formal model of a test context is a transformation of behavior as it is modeled at the IAPs to behavior as it is modeled at the PCOs. It is here described as a function C on *MODS*:

$C : MODS \rightarrow MODS$

It follows that the behavior of the IUT as observed at the PCOs is formally expressed as $C(m_{IUT})$.

The PCO is an interaction point in the test architecture where the tester interacts with the test context, and via the test context (indirectly) with the IUT. The test events are controlled and observed via the PCOs.

The IAP is an interaction point in the test architecture where the IUT interacts with it's environment, i.e. the test context, and via the test context (indirectly) with the tester.

Ideally the PCOs and IAPs coincide and the test context is empty. In this case the tester interacts directly with the IUT and $C(m_{IUT}) = m_{IUT}$.

PCOs and IAPs are formally modeled as interaction points of m_{IUT}. Their concrete representation depends on the formalism used for *MODS*.

7. TEST EXECUTION

During the run of a test case $t \in T \subseteq TESTS$, where T is a test suite, observations are made. An observation is an element from a set of all possible observations *OBS*. It can include a log of occurring interactions, a (preliminary) verdict, or anything which is considered important for determining the result of the test case execution. The result of a test case execution which leads to an observation $\sigma \in OBS$ is defined by a verdict assignment $\underline{verd}_t$ which must exist for each test case $t \in T$:

$\underline{verd}_t : OBS \rightarrow \{pass, fail\}$

The assignment of pass to an execution of a test case indicates that the test purpose could be achieved. It does not imply that the IUT is conforming under all circumstances. For example, execution of the same test case at a different occasion may yield the verdict fail. Test cases have to be designed in such a way that the test verdict pass implies with a high probability that the IUT conforms to the specification from which the test case has been derived.

A test suite T passes if all test case executions of test cases $t \in T$ yield the verdict pass. It fails otherwise.

8. HOW TO COMPLY WITH THIS STANDARD

To comply with this standard the parties involved in the formal conformance testing process shall identify the following items:

- a formalism of instantiated specifications *SPECS*;
- a parameterized standardized specification *S*;
- a modelling formalism *MODS*;
- an implementation relation imp and/or satisfaction relation sat;
- if both imp and sat are given, m imp s <=> $\forall r \in R_s$: m sat r must be demonstrated;
- a tester, an IUT and a test context and their mutual interfaces IAP and PCO in *MODS*;
- a test suite $T \subseteq TESTS$, where *TESTS* is a test notation;
- a function $C : MODS \rightarrow MODS$ modelling the test context;
- a set of observations *OBS*;
- for each $t \in T$ a verdict assignment

If automatic test generation is done, there is a need to identify a limiting function for the test suite and a coverage function that identifies the degree to which the specified functionality of the IUT is tested by the test suite.

9. CONCLUSIONS

The paper describes a framework for formal methods in conformance testing as it is in the process of standardization within ITU-T and ISO. It identifies a number of issues which are important when defining a formal method for conformance testing. The implementation relation and the testing context are two of them. Being a framework it does not describe a concrete method. Therefore the user is anyone who develops tools or formal methods for conformance testing. The framework gives some guidance on which important issues exist.

10. ACKNOWLEDGMENTS

The contents of this paper is not the work of a single person. The ideas and some of the text are taken from the FMCT working document [3] which has been developed by the joint ITU/ISO project on FMCT. Numerous persons were and are still involved in this work. In particular I want to mention Ana Cavalli, Anne Rouger, Jan Ellesberger, Jean-Philippe Favreau, Lex Herink, Pim Kars, Finn Kristoffersen, Jan Kroon, Marc Phalippou, Thomas Robles, Jan Tretmans and Umit Uyar who built the core of the project and supplied major contributions to the current version of the document.

REFERENCES

[1] Cavalli, A., et al: Automated Protocol Conformance Test Generation Based on Formal Methods for LOTOS Specifications, Proceedings of the IFIP 5th International Workshop on Protocol Test Systems, Montreal, Canada, September 1992.

[2] Ek, A., Ellesberger, J., Wiles, A.: Computer supported Test Generation from SDL Specifications, Technical Report, Telia Research, Sweden, 1993.

[3] ISO/ITU-T: Formal Methods in Conformance Testing, Working Draft, July 1994.

[4] Hogrefe, D.: Automatic generation of test cases from SDL specifications, SDL Newsletter, no.12, 1988.

[5] Hogrefe, D.: OSI formal specification case study: the INRES protocol and service, report IAM-91-012, University of Bern, 1991.

[6] Hogrefe, D., Grabowski, J., Nahm, R.: Test case generation with test purpose specification by MSCs, (in O. Faergemand: 6th SDL Forum, SDL'93), North-Holland, 1993.

[7] ISO TC97/SC21: LOTOS: Language for the temporal ordering specification of observational behavior, 1991.

[8] ISO TC97/SC21: ESTELLE: A formal description technique based on an extended state transition model, 1991.

[9] ISO/IEC: Information Technology - OSI conformance testing methodology and framework, IS 9646, 1989.

[10] ITU-T SG 10: Recommendation Z.100: Specification and Description Language SDL, 1992.

[11] Phalippou, M., Groz, R.: From Estelle specifications to industrial test suites, (in J. Quemada: FORTE'90), North-Holland, 1991.

[12] ISO TC97/SC21: Guidelines for the application of Estelle, LOTOS and SDL, technical report TR 10167, 1990.

[13] Ural, H., Williams, A.: Test generation by exposing control and data dependencies within system specifications in SDL, (in R. Tenney: FORTE'93), North-Holland, 1994.

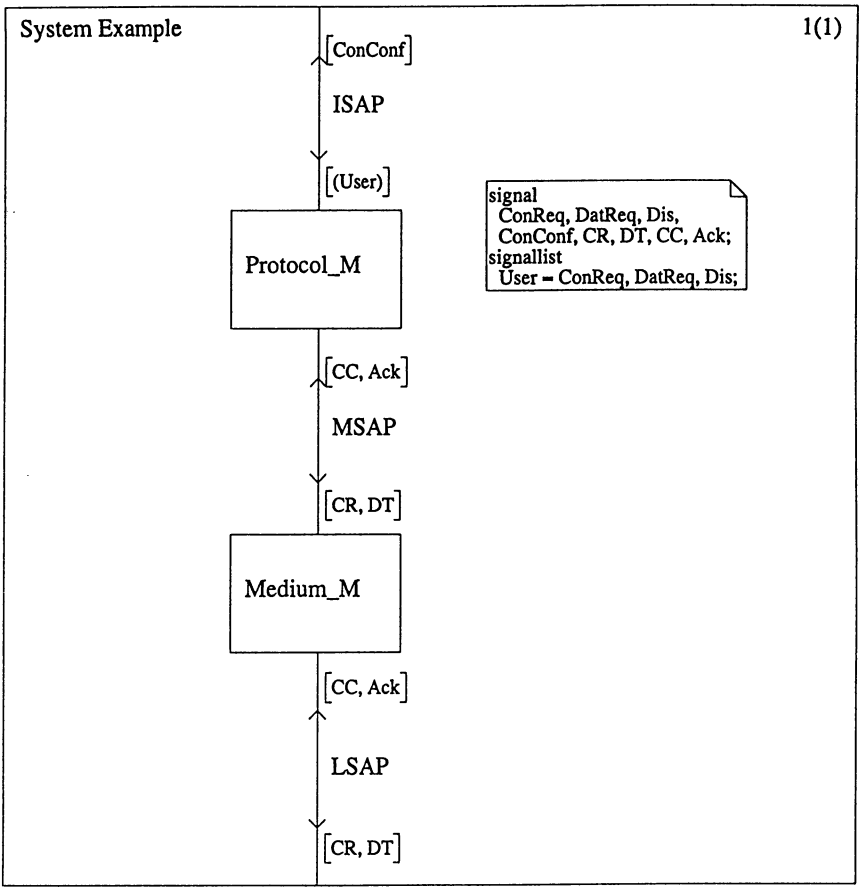

Figure 8: The system diagram for the protocol

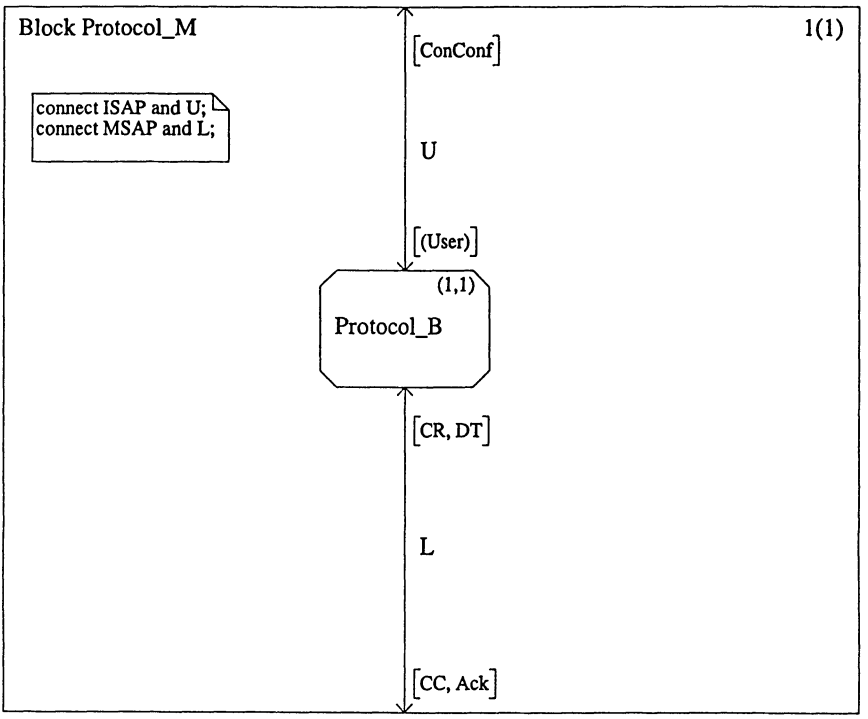

Figure 9: The block of the sending process

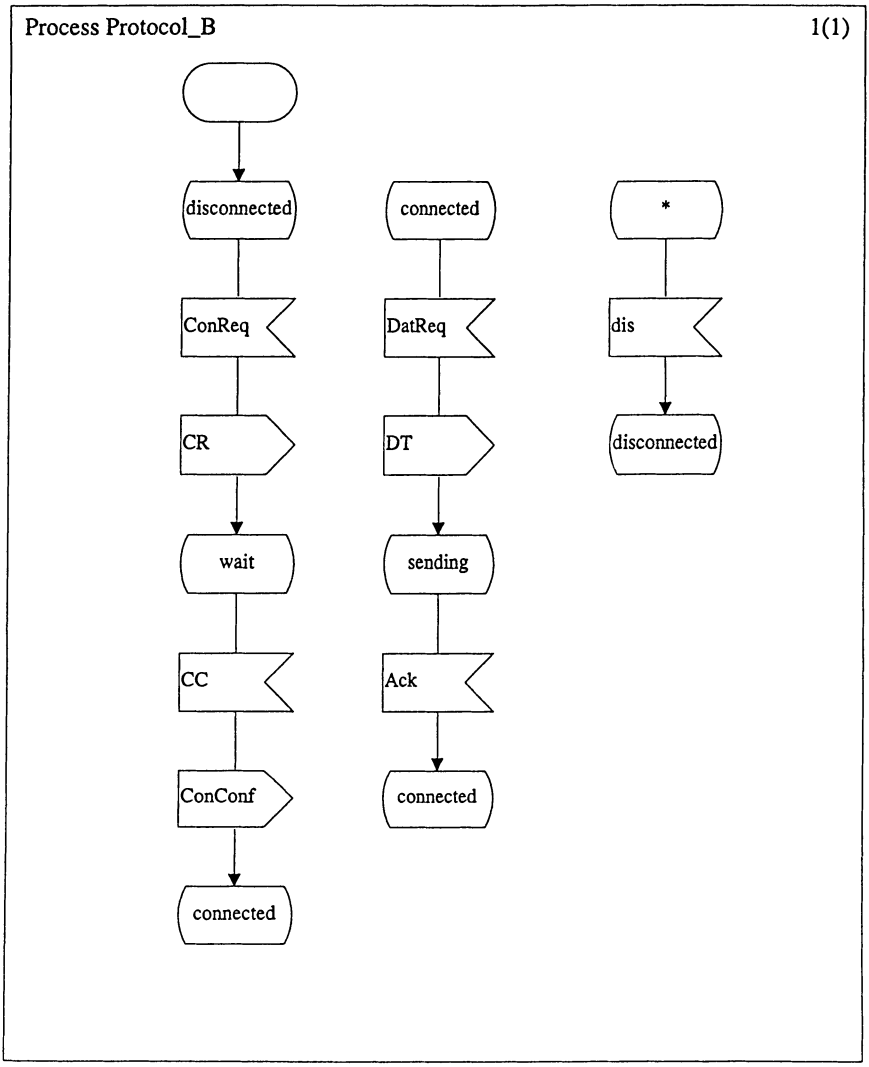

Figure 10: The sending process

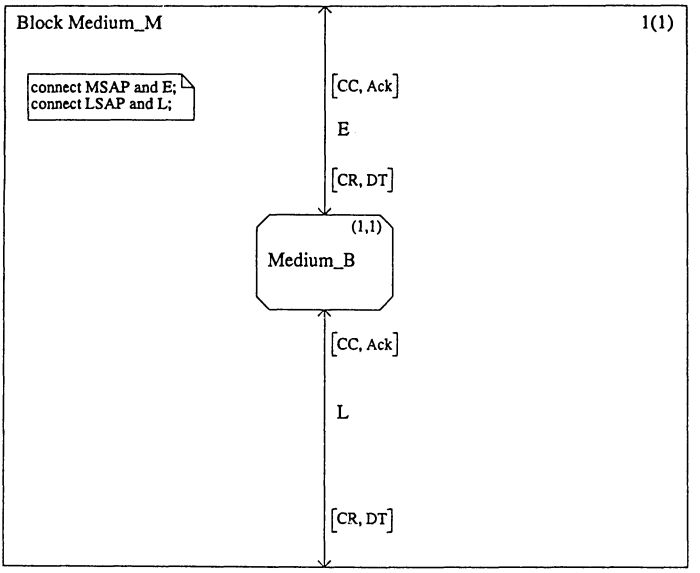

Figure 11: The block modeling the unreliable medium

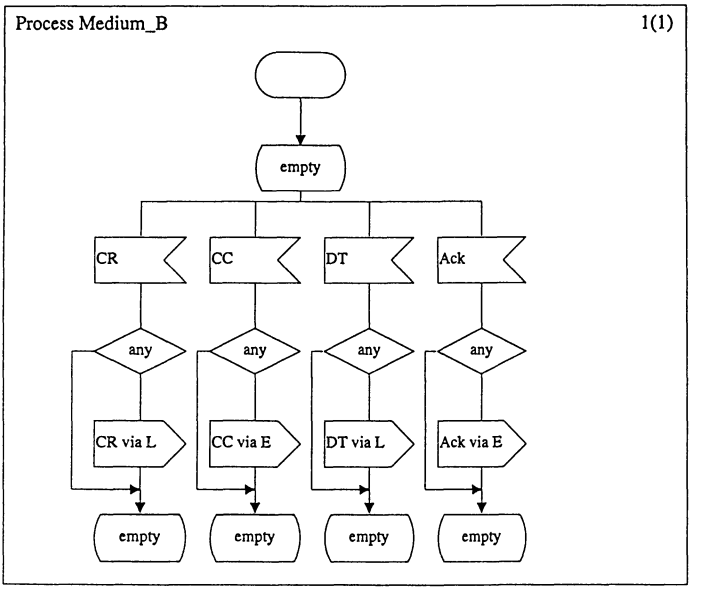

Figure 12: The behavior of the unreliable medium