Trace Analysis for Conformance and Arbitration Testing

GREGOR V. BOCHMANN, SENIOR MEMBER, IEEE, RACHIDA DSSOULI, AND J. R. ZHAO

Abstract—**There are two aspects to testing: 1) the selection of appropriate test inputs and 2) the analysis of the observed interactions of the implementation under test (IUT) in order to determine whether the observed input/output trace conforms to the IUT's specification. The paper analyzes the second aspect with particular attention to testing of communication protocol implementations. Various distributed test architectures are used for this purpose, where partial input/output traces are observable by "local observers" at different interfaces. The error detection power of different test configurations is determined, based on the partial trace visible to each local observer and their global knowledge about the applied test case. The automated construction of trace analysis modules from the formal specification of the protocol is also discussed. Different transformations of the protocol specification may be necessary to obtain the "reference specification" which can be used by a local or global observer for checking the observed trace. This checking possibly involves the "nondeterministic execution" of the reference specification. Experience with the construction of an arbiter for the OSI Transport protocol is described.**

Index Terms—**Communication protocols, conformance testing, distributed systems, formal specifications, protocol testing, testing, test oracle, test result analysis.**

I. INTRODUCTION

A WELL-KNOWN problem in system testing is the realization of a reference, sometimes called "oracle," which determines whether a given interaction sequence observed during the test of an implementation under test (IUT) is valid or not. Such an oracle must clearly be related to the specification of the IUT. This paper deals with the construction of such oracles, and their use in the protocol testing process.

In the area of communication protocol development and implementation, the protocol specification has an important role to play [7]. It is the basis for the protocol implementations in the different systems which are designed to communicate with one another, and helps for the selection of test cases and the analysis of test results. For the description of OSI protocols and services, the use of formal specifications is seriously considered [29]. The availability of such specifications make it possible to apply formal methods to the testing of protocol implementations [4].

We explore in the paper a testing approach where the concern for selecting the appropriate test input provided to the implementation under test (IUT) is separated as much as possible from the analysis of the observed output [11]. We have therefore the following two concerns in relation with the testing of an implementation:

1) the selection of the test cases, and
2) trace analysis, that is, the determination whether the trace of input and output interactions observed during a test conforms to the specification.

The first concern is important since the applied test input determines to a large extent what kind of malfunctions can be detected.

The second concern is important since it will provide the verdict as to whether a faulty behavior was found in the IUT (oracle function).

The two concerns are not independent of one another. On the one hand, the selection of test cases should take into account the detection of the possible errors that are foreseen in the underlying fault model through the observation of the IUT's output. On the other hand, the correct reaction of an IUT to a given test input can not always be predicted, either due to a nondeterministic test environment, or because the specification admits several different behaviors for the IUT. The later parts of the test input may depend on outputs received during the initial part of the test(s).

We assume in the following that the trace analysis is performed in a manner independent of the specific test input applied. While the validity of an output usually depends on previous test input, we assume that a trace analysis module is available which is based on the IUT specification, independent of the test case to be executed. (We note, however, as discussed below, that the error detection power may often be improved if knowledge about the applied test input can be taken into account.)

We take the view that the selection of test cases should be separated from the problem of deciding whether the IUT behaves according to the specification for a particular test case. As described in this paper, the second aspect can be automated if a formal specification of the IUT is available. In the case that such a specification is not available, which is the case in most protocol and other software development projects, the test cases usually include

Manuscript received March 31, 1987; revised June 30, 1988 and June 2, 1989. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

G.v. Bochmann is with the Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal P.Q. H3C 3J7, Canada.

R. Dssouli was with the Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, P.Q. H3C 3J7, Canada. She is now with Mohamed 1er University, Oujda, Morocco.

J. R. Zhao was visiting the Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, P.Q. H3C 3J7, Canada. She is with Tsinghua University, Beijing, People's Republic of China.

IEEE Log Number 8930503.

explicitly the allowed output expected from the IUT during test execution. This is a duplication of the information given in the specification. It is therefore desirable to validate the test cases in respect to the system specification. However, this is difficult to do if the system specification is written in an informal language. In the case that a formal system specification is available, test cases can be validated against it using an approach similar to the trace analysis described in this paper.

While the above considerations apply not only to communicate protocols, but to the problem of software development in general, protocol testing has certain particularities which relate to the distributed nature of the implementations. Various distributed test architectures [21] are reviewed in Section II. In most of these architectures, the test system is partitioned into several components, each accessing a local test control and observation point (TCOP) which allows only partial control and observation of the IUT.

The possibilities of trace analysis in such distributed test architectures, and the limitation of the error detection power is discussed in Sections III and IV. The problem of deriving the appropriate trace analysis module for a given test architecture and protocol specification is addressed in Section V. The experience with an example of this approach is presented in Section VI, which deals with trace analysis for the arbitration testing between two OSI Transport protocol implementations. Section VII contains the conclusions and a discussion of the relevance of trace analysis to OSI conformance testing.

II. Test Architectures

As shown in the layered protocol architecture of Fig. 1, a protocol entity has essentially two points through which it interacts with its environment, namely the upper and lower local service interfaces, sometimes called "service access points" (called USAP and LSAP in Fig. 1). For a protocol entity of layer N, these are the service access points for layer N and $(N-1)$, respectively. In a local test environment, as shown in Fig. 2, the IUT (which is assumed to represent a protocol entity of layer N) is directly controlled and observed by the test system through these two access points. They are the test control and observation points (TCOP's), which in general are the interaction points through which the test system (or parts of it) interact(s), directly or indirectly, with the IUT.

In addition to this local test architecture, which corresponds to traditional software testing, the OSI standardization community has identified [21] a number of so-called external test architectures for which at least part of the test system resides outside the system containing the IUT, which is called "system under test" (SUT). They are included in the following architectures considered in this paper:

1) Distributed Test Architecture: As shown in Fig. 3, the test system is divided into so-called upper and lower testers which access the upper and lower service interfaces of the IUT, respectively. The lower interface is accessed over distance and indirectly through the underlying communication service.

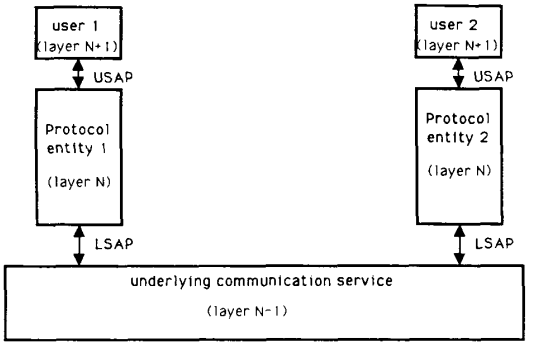


Fig. 1. Layered protocol architecture.

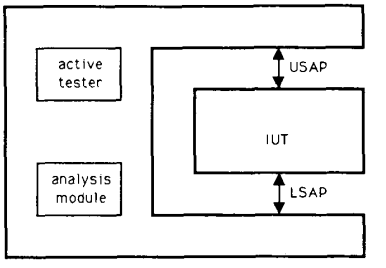


Fig. 2. Local test architecture.

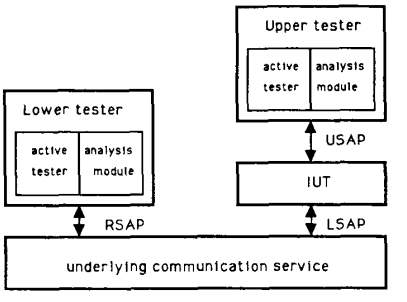


Fig. 3. Distributed test architecture.

2) Remote Test Architecture: This corresponds to the distributed test architecture where only the lower tester is used. Instead of the upper tester, the system under test may include a stack of several protocol layers above the layer being tested.

3) Coordinated Test Architecture: This is a distributed test architecture where some form of coordination between the lower and upper testers is established through the exchange of messages according to a so-called test coordination protocol through a (possibly separate) communication channel between the upper and lower testers. A particular case is the so-called ferry architecture shown in Fig. 4 [23], [30] where all interactions at the upper interface of the IUT are exchanged with the remote test system which therefore controls and observes both interfaces, however indirectly.

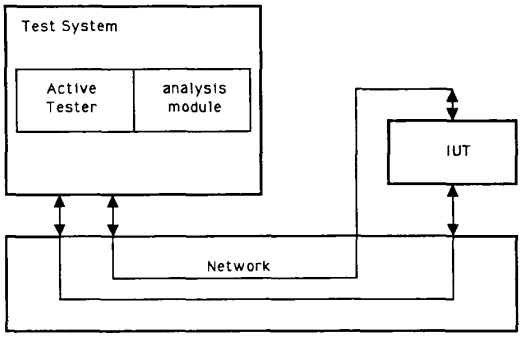


Fig. 4. "Ferry" or "astride" test architecture.

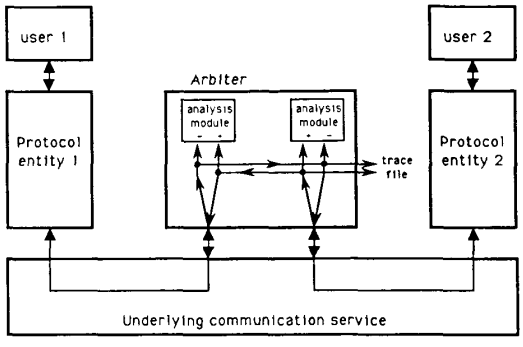


Fig. 5. Test architecture with arbiter.

4) Architecture for Arbitration Testing: The architecture of Fig. 1 can be used for interoperability testing between two different implementations of the same protocol. If it is not known which implementation exhibits a deviation from the protocol specification, it is possible to introduce an observer, sometimes called "arbiter," as shown in Fig. 5, which observes the exchange of protocol data units (PDU's) between the two implementations in both directions and should detect any deviations from the protocol specifications. As only the exchanged PDU's are observed (and not the interactions at the upper interfaces of the implementations), the error detection power of the observer is equal to the case of the remote testing architecture, although the application of specific test cases is more difficult than in the latter case.

In each of the testers involved in the above architectures, a distinction between aspects of test case selection and trace analysis can be made, as indicated in the figures. It is noted that an arbiter, as shown in Fig. 5, only realizes the trace analysis function, while the aspect of test case selection is realized within the interworking systems by the users of the communication protocol. The arbiter should have a minimal impact on the communication between the interworking systems (compare Figs. 5 and 1). This is satisfied, for example, in the context of a local area network where an observer in one station may observe the traffic between the other stations [2], [20], if a suitable modification to the network access module is made. The case of arbitration for higher layer OSI protocols is discussed in Section VI.

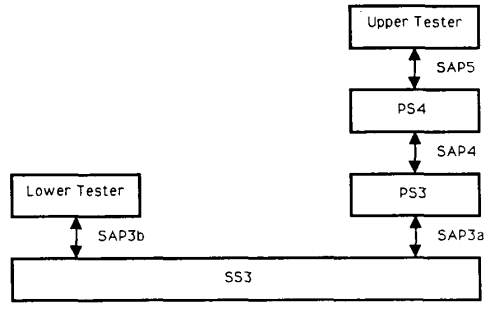


Fig. 6. Example of multilayer test architecture.

Except for the remote test architecture, all test architectures above assume that the upper interface of the protocol IUT is accessible for testing purposes. If this is not the case, either the remote test architecture or a so-called "multilayer" test architecture must be used. Fig. 6 shows a particular example.

III. Local Observers and Their Reference Specification

Ideally, the trace of observed interactions performed by the IUT during a test is verified by global and direct observation at the interfaces of the IUT, as possible for the local test architecture shown in Fig. 2. An error is detected if the observed trace t is not a valid trace according to the specification S of the IUT, which we write "t does not conform to S." All cases of erroneous behavior generated by the IUT can in principle be detected in this manner, as long as appropriate test inputs are chosen that lead the IUT to exhibit these behaviors. As discussed in the Introduction, this paper does not address the issues related to the selection of such test cases. In order to verify real-time properties of the IUT, such as related to timer specifications, it is necessary to either analyze the interactions of the IUT in real time, or record, within the trace file, the real time of each interaction in the form of a time stamp. Possible deadlocks of the IUT can be detected by noticing that certain output interactions, expected according to the specification, are missing.

An observer watching only the interactions at one of the interfaces is called a local observer. We call "Oi" the local observer at the interface i. It observes only a partial trace which is obtained from the global trace by deleting all interactions not taking place at the interface i. We write "Pi(t)" for this trace, where "Pi" stands for "projection onto interface i".

The observer compares the observed local trace with a specification which we call the reference specification of the observer. In the case of the test architecture of Fig. 2, the reference specification for a local observer at one of the interfaces can be obtained from the specification of the IUT by projection on the interface in question; in the case of the local observer at the remote service access

point (RSAP) of Fig. 3, its reference specification can be obtained through the composition of the specifications of the IUT and the underlying communication service and subsequent projection onto the interface RSAP (see for instance [19] or [8] for a discussion of the concept of projection). In general, the reference specification of an observer is a specification which determines which traces (observed at the interface in question) conform to the system specification. It can be obtained from the protocol specification of the IUT and the test architecture [11], [13], as explained below.

In this paper we use the notation of LOTOS [18] for describing the composition of system modules and projections. A system module is represented as a Lotos process of the form

process-name [list of interfaces]

and the composition of two modules M1 [I1, I2] and M2 [I2, I3] interacting over the common interface I2 is written as

hide I2 in (M1 [I1, I2] |[I2]| M2 [I2, I3])

which implies that I2 is not accessible to any other modules in the system. Similarly, the projection of the module specification M1 onto the interface I1 is written as a Lotos expression of the form

hide I2 in M1 [I1, I2]

which means that the interactions at interface I1 remain visible and must conform to the specification of M1, while at I2 any interaction may occur at any time (but not visible to the projected specification).

Using this notation, the reference specification RSUT for the observer at the upper interface of the IUT (sometimes called "upper tester," UT, see Fig. 3) can be written as

RSUT [USAP] := hide LSAP in SPECIUT [LSAP, USAP]

where SPECIUT is the protocol specification for the IUT and if one assumes that the underlying communication service does not restrict the possible interactions at the lower interface of the IUT. Similarly, the reference specification RSLT for the lower tester of Fig. 3 can be written as

RSLT [RSAP] := hide LSAP, USAP in
(SPECIUT [LSAP, USAP] |[LSAP]|
LS [RSAP, LSAP])

where LS is the specification of the underlying communication service. Another example is the multilayer test architecture of Fig. 6. Here, the reference specifications for the local observers at the interfaces SAP5 and SAP3b observing indirectly the implementation of the protocol PS3 can be written as

RSUT34 [SAP5]:=hide SAP4, SAP3a, SAP3b in
(SS3 [SAP3b, SAP3a]
|[SAP3a]| PS3 [SAP3a, SAP4]
|[SAP4]| PS4 [SAP4, SAP5])

and

RSLT34 [SAP3b] := hide SAP3a, SAP4, SAP5 in
(SS3 [SAP3b, SAP3a]
|[SAP3a]| PS3 [SAP3a, SAP4]
|[SAP4]| PS4 [SAP4, SAP5])

IV. Error Detection Power of Local Observers

A. Limited Error Detection Power

In general, the error detection power of a local observer is less than perfect. It is conceivable that the IUT exhibits a faulty behavior, say a trace t which does not conform to S, the specification of the IUT, but a local observer does not detect an error since the projection of t on the observed interface results in a local trace which is allowed according to the reference specification of the observer.

As an example, we consider the simplified OSI Transport protocol class 2 [22]; the state transition diagram of Fig. 7 defines the allowed execution orders of service primitives, and sending and receiving of protocol data units (PDU's). An allowed execution trace is, for instance, the following

t1 = < TCONreq, s-CR, r-CC, TCONconf,
TDATAreq, s-DT, . . . >

where the service primitives TCONreq, TCONconf, and TDATAreq are executed at the "upper" interface USAP (see Fig. 2), and the sending and receiving of the PDU's CR, CC, and DT are executed at the "lower" interface LSAP. The notation "s-CR" (or "r-CR") means the sending (or reception) of the connect request (CR) PDU. The local observer O_{USAP} at the upper interface USAP would therefore observe the trace

< TCONreq, TCONconf, TDATAreq, . . . >

while a local observer O_{LSAP} at the lower interface would observe the trace

< s-CR, r-CC, s-DT, . . . >.

Let us consider a fault where the IUT does not send the required CR PDU. The resulting global trace would be

t2 = < TCONreq, TCONconf, TDATAreq,
s-DT, . . . >.

This fault would not be detected by the local observer O_{USAP}, since it would observe a valid local trace, in fact $P_{USAP}(t2) = P_{USAP}(t1)$. However, this fault would be detected by a local observer O_{LSAP}, since it would observe the trace

< s-DT, . . . >

which is not allowed according to $P_{LSAP}(S)$.

It is important to note that local observers alone do not detect all errors, at least in most cases. While the above error would be detected, the following error would not be detected by O_{USAP} nor O_{LSAP}. We assume that the IUT

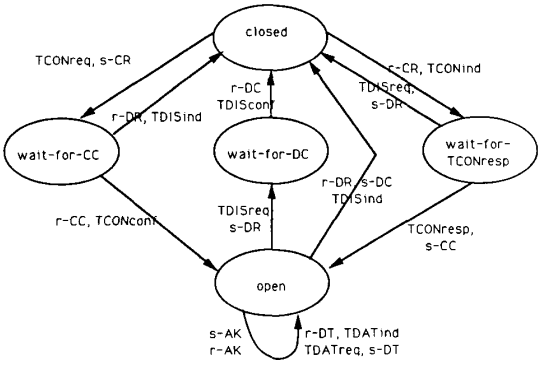


Fig. 7. State diagram of a transport protocol specification.

sends the CR PDU, but invokes the TCONconf before, thus leading possibly to the trace

t3 = < TCONreq, TCONconf, s-CR, r-CC, TDATAreq, s-DT, . . . >

This global trace has the same local traces as the valid trace t1. Therefore no error is detected.

While the above examples relate to the order of execution of interaction primitives, other errors may relate to the interaction parameters and the data flow between them. The analysis of possible data flows within the specification can also be used for deriving appropriate test input for covering data flow related faults [27]. An important property of communication protocols related to data flow is the reliable transfer of user data. For the Transport example considered above, this means that the user data parameter of the TDATAreq primitive, in traces such as t1, must be equal to the user data parameter in the corresponding data (DT) PDU. An example where this is not true is the faulty trace (we assume x1 not equal to x2)

t4 = < TCONreq, s-CR, r-CC, TCONconf, TDATAreq(x1), s-DT(x2), . . . >.

Most data flow relations, such as the one above, are not local properties, that is, their violation cannot be detected by local observers.

B. Improving the Error Detection Power of Local Observers

Error detection power of local observers can be increased by the following two approaches (see also [11]–[13]).

1) Communication Between Observers and Global, Indirect Observation: In most test systems, there is some global instance which determines the test input to be applied, and provides for the detection of errors. We assume that this global error detection function is provided by a so-called global analyzer which bases its error detection diagnostic solely on information received from the local observers. Several level of information transfer from the local observers to the global analyzer can be considered:

a) Diagnostic Information Transfer: Each local observer *Oi* indicates whether it detected an error based on its reference specification. This corresponds to the discussion of the example above, where an error would be found in the trace t2, but not in the traces t3 nor t4.

b) Transfer of Local Trace: Each local observer *Oi* provides the global analyzer with the locally observed trace *ti*, that is the sequence of observed interactions and their parameter values. Such an approach can be realized through a test architecture as shown in Fig. 4. With this approach, many data flow related faults can be detected, such as the error in trace t4.

c) Transfer of Local Trace with Timing Information: In addition to the information provided under point b), it is assumed that the local observers have synchronized clocks which are used to indicate for each observed interaction the real time of its occurrence. If the clocks are well enough synchronized, this may allow the global observer to detect the error in trace t3.

Information transfer at level c) provides the global analyzer with complete information about the observed interactions. Therefore all errors can be detected with this methods, provided the timing information is precise enough. However, the realization of synchronized clocks in a distributed environment is not easy (see for instance [16]). Therefore information transfer at level b) is used more frequently.

Level b) communication allows the detection of many sequencing errors, namely when the sequences observed at the two sides are inconsistent. An example is the trace

t5 = < TCONreq, s-CR, r-DR, TCONconf, . . . >,

where disconnection occurs at the "lower" interface, while a connection is established at the "upper" side. Even sequencing errors where no such inconsistency exists, such as in trace t3, may be detected if data flow relations are taken into account. If, for instance, the TCONconf contains a parameter value which depends on a parameter value received in the CC PDU, the sequencing error of t3 should usually be accompanied with an error in the TCONconf parameter, since the IUT evoking the TCONconf too early has not yet received the CC PDU for correctly determining the TCONconf parameter.

2) Test Case Specific Trace Analysis: The second approach to improving the error detection power is to provide each local observer with information about the test input supplied at the other interfaces. For instance, if the local observer O_{LSAP} knows in advance the user data to be applied at interface USAP in the TDATAreq interaction, it will be able to detect the error in trace t4. This approach is the basis for many standardized OSI test cases [21] used in the distributed test architecture of Fig. 3. However, this approach can not be used for random testing where the test input is (to a certain extent) randomly chosen, since in this case the test input is not known in advance.

Formally, the additional knowledge of the local observer can be expressed by its reference specification. This specification can take into account the known behavior of

the test modules providing input to the other interfaces. For example, if UTT1 is the specification of the upper tester for a given test case using the architecture of Fig. 3, the reference specification for the lower tester at the interface RSAP can be written as

```
RSLTT1 [RSAP] := hide LSAP, USAP in
(UTT1 [USAP]
   |[USAP]| SPECIUT [LSAP, USAP]
   |[LSAP]| LS [RSAP, LSAP])
```

which in general accepts less traces than the reference specification RSLT given in Section III.

The semiadaptive testing often proposed for protocol conformance testing [10], [3] also takes advantage of this approach. In this case, a complete test suite consists of a number of predetermined test cases. For each test case, the observer function at each of the interfaces knows the test input to be applied at the other interface, and this is taken into account for the detection of errors. The next test case to be executed is determined by the test system based on the global results of test cases previously executed. Communication between the different parts of the test system are therefore only required to communicate the local test diagnostics and the identity of the next test case to be executed, but not the details of its definition. Various methods for such test coordination procedures have been described [24], [3], [21].

C. Influence of the Underlying Communication Service and Multilayer Protocol Testing

In the case of most testing architectures, such as those shown in Figs. 3, 4, 5, and 6, the error detection power of the observers is further decreased due to the fact that the "lower" interface of the IUT are not observed directly, but only through an underlying communication service. The latter usually implies delays and possible queuing of messages in transit. Even if the communication service has the FIFO property (in a given direction, messages are received in the same order as they were sent), the order of interactions observed may be different from the order they occurred at the IUT interface due to message cross-over within the communication service.

In the case of multilayer protocol testing as shown in Fig. 6, similar cross-overs of interactions may occur for the indirect observation of the upper service primitives of the PS3 protocol entity observed at the SAP5 interface, and the lower service primitives of the PS4 protocol entity observed at the remote SAB3b interface. However, the situation is more difficult to analyze since the behavior of the protocol entity through which these interactions are observed is usually much more complicated than the underlying communication service. In addition, it can usually not be assumed that the implementation of this entity is without faults. Therefore, in practice, the different protocol layers are usually tested in conjunction, using the reference specifications RSUT34 and RSLT34 defined in Section III.

V. Deriving a Trace Analysis Module from the Reference Specification

A. General Considerations

In the section above, various configurations of local observers and their error detection power were discussed. Each observer compares the locally observed (partial) trace with its reference specification in order to determine whether an error can be detected. The discussion in this section deals with the construction of a trace analysis module which actually does this comparison. The trace analysis module should be derived from the reference specification, and should be in such a form that it reads the observed trace, one interaction after the other, and reports, after each interaction considered, whether an error occurred up to this point.

If the reference specification is given in an executable form, it may be used for trace analysis by having it executed in a simulated manner. However, the following points must be taken into account:

a) Any output interaction specified by the reference specification corresponds to an input to the trace analysis module received from the IUT. The trace analysis module verifies that the received parameter values are equal to the parameter values specified for the output in the reference specification.

b) The reference specification may be nondeterministic, in the sense that a given trace may lead to more than one internal "state" of the specification. The allowed subsequent interactions may be different for these different states. Therefore the reference specification must be simulated in a nondeterministic manner, considering all possible branches in parallel. Only if for the next interaction of the observed trace, there is no branch for which this interaction is possible, then an error is detected. In addition, any branch for which the next interaction is not possible may be deleted for further consideration in the nondeterministic execution of the reference specification. The interested reader is referred to [15] and [28] for further discussion and examples.

The following subsections discuss the derivation of an analysis module for different kinds of reference specifications.

B. Finite State Machine Specifications

In the case that protocol specifications are given as finite state machines, the construction of trace analysis modules is relatively straightforward. The projection operation Pi is performed by deleting from the state diagram all input/output labels which do not occur at the interface i. For example, the Transport protocol specification of Fig. 7 leads to a reference specification for the "lower" observer O_{LSAP} shown in Fig. 8. In general, the specification may be simplified by reducing the state machine to an equivalent form [1], thus avoiding spontaneous transitions without output (i.e., transitions with no input or output occurring at interface i). The composition opera-

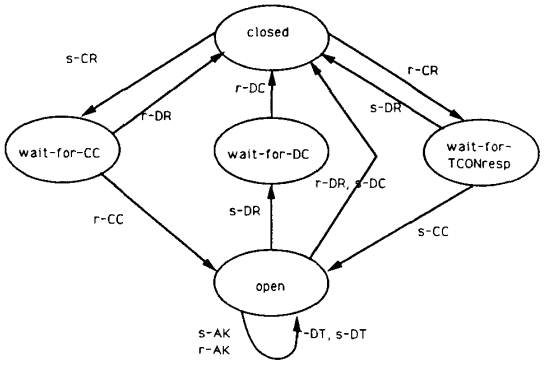


Fig. 8. State diagram of a reference specification for lower observer.

tion is also relatively simple [19]; it may, however, lead to a certain state space explosion.

In order to transform outputs to inputs (see point a) above and obtain a state diagram where all transitions have a single input interaction, transitions with both input and output (such as the transition labeled "r-DR, s-DC" in Fig. 8) are split into two transitions and an intermediate state. Then all output interactions are converted to input. If the resulting state machine is deterministic, error detection transitions can be added as follows. For each state of the machine and each input for which the machine does not include a transition, an error detection transition should be included leading to a "detected error" state.

If the resulting state machine is not deterministic, it may either be converted into an equivalent deterministic machine [1], or it may be executed in a nondeterministic manner, as described under point b) above.

C. Full Specifications Using a Formal Description Technique

Finite state machines have a limited modeling power. So-called formal description techniques (FDT) have been developed for writing full specifications of OSI communication services and protocols [29]. It is an advantage of formal protocol specifications written in an FDT, that trace analysis modules for local observers can be derived from such specifications using similar transformations as those described for finite state machines above.

The techniques Estelle [14] and SDL [25] are based on an extended finite state machine model which allows the specification of a system of interconnected extended state machines. The different machines communicate through the exchange of messages. For the derivation of trace analysis modules, the transformations described in Section V-B can be applied with proper consideration of interaction parameters. This means that during the transformation from output to input interactions, as described above, a condition must be associated with the new input transition which verifies that the received interaction parameters are equal to the values specified for the output in the reference specification. The transformation from a nondeterministic specification into a deterministic one is not easily feasible for FDT specifications; therefore the trace analysis module has to execute the resulting specification in a nondeterministic manner, as explained under point b) in Section V-A.

In the case of the LOTOS language [18], the situation is similar. However, the rendezvous interactions are not distinguished as input and output; instead individual parameters of these interactions may be input or output, as indicated by the notation "?" or "!", respectively. Therefore the transformation from output to input must be applied to each output parameter of an interaction. This means that, for each occurrence of "!⟨ value expression ⟩" within the parameter list of a Lotos *action prefix* representing an interaction, the occurrence should be replaced by the text "?*v*:*type_id*" where *v* is a new parameter identifier and *type_id* is the identifier of the data type (Lotos *sort*) of the ⟨ value expression ⟩. In addition, the *action prefix* should be associated with the guard "[v = ⟨ value expression ⟩]", possibly to be combined through a logical *and* with other guards already present. The obtained Lotos specification should be executed in a simulated nondeterministic manner, as explained in Section V-A. An existing Lotos interpreter [17] has been adapted for this purpose [9].

VI. A Practical Example: Transport Arbiter

A. Arbitration Testing

In the case of arbitration testing, as shown in Fig. 5, an arbiter observes the PDU's exchanged between two protocol implementations with the purpose of detecting any cases where one of the implementations (IUT's) does not conform to the protocol specification. Since both implementations must be checked, we suggest an internal structure for the arbiter which contains two trace analysis modules, one for each observed IUT, as shown in the figure. The observed trace of PDU's, received from the different IUT's, is recorded in a trace file for eventual manual inspection or later processing. In addition, the oracle function is performed on the observed trace separately for each IUT. The two trace analysis modules check whether the observed trace is in contradiction to the specification of the corresponding IUT.

For each of the two trace analysis modules, the reference specification is equal to the one for a lower tester (RSLT) in the distributed test architecture (see Fig. 3) as defined in Section III. This takes care of the possible queuing of PDU's within the underlying communication service. Therefore only the properties visible at the "lower" interface of the IUT's can be checked by the arbiter. Many protocol properties, such as correct transfer of user data for instance (see also trace t4 in Section IV-A) cannot be checked. Many of these properties could, however, be verified by checking the correct operation of the next higher protocol layer which relies on these properties for its correct operation.

It is assumed in the following that the observed PDU's are exchanged through a reliable communication service.

This assumption simplifies the analysis of protocol conformance. If errors may occur within the underlying communication service, it is difficult to decide whether an error observed in a given PDU is due to a transmission error in the lower service or to a faulty behavior of the IUT from where the PDU originated.

B. Constructing an Arbiter for the OSI Transport Protocol

We have applied the principles discussed in the sections above to the implementation of an arbiter for the OSI Transport protocol classes 0 and 2. The reference specification for trace analysis was essentially derived from a formal specification of the protocol (written in an Estelle dialect) which was previously used for deriving an implementation [5]. The transformations performed for this purpose are similar to those discussed in Section V, and take also into account the interaction parameters and additional state variables. These transformations were performed by hand; no automated tool was available for this purpose. It turned out that the resulting specification of the trace analysis module was deterministic, which simplified its execution.

The arbiter was implemented using a semiautomated approach [5]. The arbiter was first specified in Estelle. After this specification was sufficiently completed, it was automatically translated into Pascal and combined with run-time support routines, including in particular an interface to the underlying Network service (provided by an X.25 network) and functions for operator control and copying the trace onto a file for possible later inspection.

The Estelle module structure of the arbiter is shown in Fig. 9(a). The **Network-medium** provides a Network service access point. The **Manager** module manages the two Network connections to the respective computers containing the two IUT's. It also includes a PDU decoding function and writes the trace file. The decoded PDU's are passed to each of the two **Side** modules which perform the trace analysis for the two respective IUT's. The internal structure of the **Side** modules is shown in Fig. 9(b) and is closely related to the structure of the original formal protocol specification. Each **AP** module checks a single Transport connection. Since the class 2 protocol allows for multiplexing, several such connection may be established over the same Network connection.

The **Mapping** module looks after multiplexing and performs a number of checks on the observed PDU's. A procedure **Sending_check** is derived from the original protocol specification. It checks conditions for the PDU's sent by the corresponding IUT. A **Receiving_check** procedure does the same tests for the PDU's to be received by the corresponding IUT. This allows the arbiter to know whether the corresponding IUT is expected to send an ERR PDU in response to an erroneous PDU received.

Each **AP** module is an extended finite state machine derived from the **AP** module of the original protocol specification. Its state diagram is shown in Fig. 10. Note that the transitions leading to the "detected error" state (see Section V-B) are not shown on the diagram; the states **error, wfCC_er,** and **wftresp_er** in the diagram are caused by valid transitions of the protocol machine which

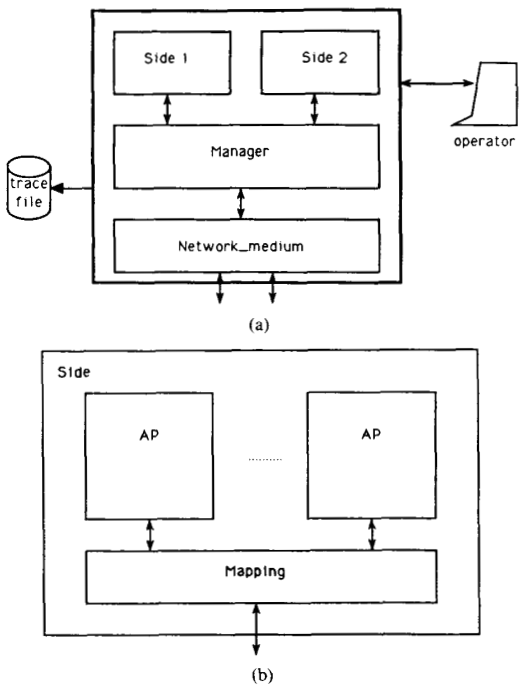


Fig. 9. (a) Estelle structure of arbiter. (b) Estelle structure for side module of arbiter.

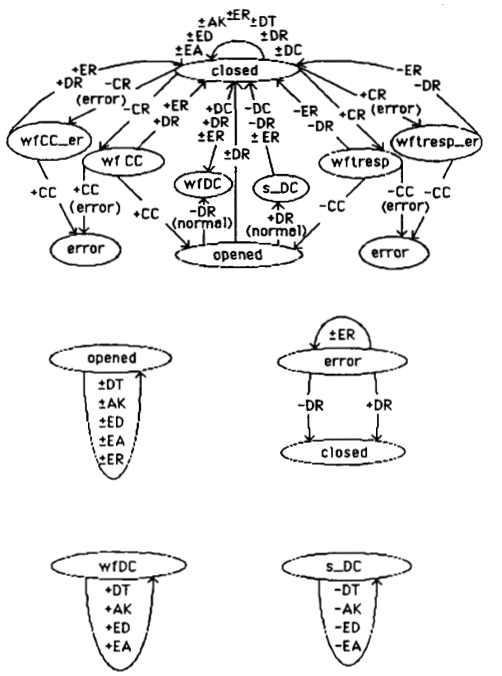


Fig. 10. State transition diagram for analysis module of transport arbiter.

can be reached when the IUT receives certain invalid PDU's.

In addition to using the formal protocol specification for deriving the trace analysis rules, we also studied the OSI standard [22] in order to validate these rules. It turned out that there was a small number of inconsistencies between the formal specification and the standard. It is not easy to make a comparison of a formal specification with a standard written in natural language. It would therefore be useful if recognized and validated formal specifications of protocol standards would be available.

Several different Transport protocol implementations were tested using the configuration of Fig. 5 [6]. It was found that the arbiter was useful for detecting a large part of the errors in the implementations (some of the errors were not visible at the lower service interfaces). It seems that an arbiter can be especially useful in the following cases:

1) capturing the errors which occur occasionally or cannot be easily reproduced,

2) capturing errors which occur only after interactions have taken place repeatedly for a long time,

3) recording unexpected events which come from the lower layer and could result in a misbehavior of the IUT,

4) monitoring the execution of two implementations which have been tested separately and have just been put into use, and

5) being used as a first filter for debugging before more costly tests are undertaken.

VII. Concluding Discussion

There are two aspects to testing: 1) the selection of appropriate test inputs and 2) the analysis of the observed interactions of the implementation under test (IUT) including input *and* output in order to determine whether the observed trace is conform to the IUT's specification. While standardized test suites for OSI communication protocols combine these two aspects into a single test case description, this paper explores the advantages of keeping the two aspects separate.

This paper explores the automated analysis of the observed interaction trace in respect to the IUT's specification. It is clear that such an analysis can only be automated if the specification is given in a formal, machine-processable form. This is one of the reasons why the development of formal descriptions of OSI protocols and services in a formal description technique (FDT) is advocated. They cannot only be used for automating the implementation testing process, as described here, but can also play a major role during the validation of the protocol design, and can be used for partly automating the implementation process [4].

It is important to note that most test architectures used for protocol implementation testing allow only for partial control and observation of the IUT interactions [13]. In most situations, one or several local observers record and analyze the partial interaction trace visible at the interface which they observe. Many important error types cannot be detected by such local analyzers unless some global information is introduced. There are essentially two possible approaches:

1) Static Knowledge: Each local analyzer has some *a priori* knowledge about the applied test input at the other interfaces; or

2) Dynamic Knowledge: Some global analyzer obtains a copy (possibly with real-time information) of the partial traces observed at the different interfaces.

Both approaches lead to satisfactory error detection power.

For established OSI protocol standards, such as X.25, FTAM, or MHS (X.400), a number of standard test cases are defined by interested groups and/or standardization bodies. These test cases not only include the inputs to be applied to the IUT, but also describe the possible outputs observed, and for each possible output whether its occurrence means a successful test, the detection of an error, or an inconclusive test outcome. The latter information is also provided by the trace analysis discussed in this paper. Nevertheless, the automated trace analysis based on the specification of the IUT is useful in this area for the following purposes.

First, it can be used to validate the predefined test cases. This is an important point, since it is difficult in general to check that the error detection diagnostic given by a predefined test case, such as those used for OSI, is conform to the protocol specification. In fact, the automated trace analysis described in this paper could be used to validate a predefined test case by comparing the error detection diagnostic given by the test with the diagnostic of the automated trace analysis, and this for all the traces foreseen by the test case [9].

Secondly, automated trace analysis can be used in situations where predefined test cases cannot be used, for example:

1) During conformance testing of an implementation, it is often desirable to execute tests which have not been foreseen by the implementor. New test cases may be selected for this purpose. A similar situation occurs in hardware testing where test cases are sometimes selected randomly.

2) While standard OSI conformance test cases are defined for verifying conformance of an implementation in respect to the protocol specification, each implementation usually has to satisfy additional requirements which are system-specific. Specific test cases for verifying these requirements must therefore be designed and executed.

3) Another case where the standard OSI conformance test cases cannot be used is arbitration testing which is performed when two implementations, already well tested, do not interwork properly. A test architecture as shown in Fig. 5 is normally used, and the applied test input is not standardized.

4) Finally, the standard test cases are usually not used for the initial debugging of a new implementation. In this early implementation phase, it is often desirable to be able to execute ad hoc test cases and verify the behavior of the

implementation in these situations. Similarly, specific tests may be selected during conformance resolution testing [21] to check particular conformance requirements.

Before closing, it is important to note that the automated trace analysis discussed here is only useful in detecting a fault in an IUT if the applied test input is able to lead the IUT into a situation where it exhibits an error due to that fault. Therefore the selection of appropriate test input is of prime importance. Although test input can also be selected based on the system specification (for application to communication protocols, see for instance [26], [27]), it seems that it is very difficult to fully automate the test selection process. This paper tries to show that the trace analysis aspect of testing is "orthogonal" to the test input selection issues, and to explore its automation based on a formal system specifications. These principles are not new in the area of hardware design, however, the point of this paper is the application of these principles to communication protocols. They may also be applicable in other software areas.

REFERENCES

[1] A. V. Aho and J. D. Ullman, *Principles of Compiler Design*. Reading, MA: Addison-Wesley, 1977.

[2] J. M. Ayache, P. Azema, and M. Diaz, "Observer: A concept for on-line detection for control errors in concurrent systems," in *Proc. 9th Int. Symp. FTC*, Madison, WI, June 1979.

[3] G.v. Bochmann, E. Cerny, M. Maksud, and B. Sarikaya, "Testing of transport protocol implementations," in *Proc. CIPS Conf.*, Ottawa, 1983, pp. 123-129.

[4] G.v. Bochmann, "Usage of protocol development tools: the results of a survey" (invited paper), presented at the 7th IFIP Symp. Protocol Specification, Testing and Verification, Zurich, May 1987.

[5] G.v. Bochmann, G. Gerber, and J. M. Serre, "Semiautomatic implementation of communication protocols," IEEE *Trans. Software Eng.*, vol. SE-13, no. 9, pp. 989-1000, Sept. 1987; reprinted in *Automatic Implementation and Conformance Testing of OSI Protocols*. D. P. Sidhu, Ed. New York: IEEE, 1989.

[6] G.v. Bochmann, C. He, D. Ouimet, and R. Zhao, "Protocol testing using automatic trace analysis," in *Proc. IEEE Canadian Conf. Electrical and Computer Engineering*, Sept. 1989.

[7] G.v. Bochmann, "Protocol specification for OSI," *Comput. Networks and ISDN Syst.*, to be published.

[8] S. S. Lam and A. U. Shankar, "Protocol verification via projections," *IEEE Trans. Software Eng.*, vol. SE-10, no. 4, pp. 325-342, July 1984.

[9] G.v. Bochmann and O. Bellal, "Test result analysis in respect to formal specifications," submitted for publication.

[10] I. C. Davidson, "The NCC protocol testing service," in *Proc. Workshop Introduction of High Level Protocol Standards for OSI (BNI/AFNOR)*, Paris, June 1983, pp. 273-279.

[11] R. Dssouli and G.v. Bochmann, "Error detection with multiple observers," in *Proc. IFIP Workshop Protocol Specification, Testing, and Validation*, Toulouse, France, June 1985.

[12] ——, "Conformance testing with multiple observers," in *Proc. IFIP Workshop Protocol Specification, Testing, and Validation*, 1986, pp. 217-229.

[13] R. Dssouli, "Etude des methodes de test pour les implantations de protocoles de communication basees sur les specifications formelles," Ph.D. dissertation, Univ. Montreal, 1986.

[14] ISO IS9074, "Estelle: A formal description technique based on an extended state transition model," 1989.

[15] C. Jard and G.v. Bochmann, "An approach to testing specifications," *J. Syst. Software*, vol. 3, no. 4, pp. 315-323, Dec. 1983.

[16] L. Lamport, "Time, clocks and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558-565, July 1978.

[17] L. Logrippo *et al.*, "An interpreter for LOTOS: A specification language for distributed systems," *Software Practice and Experience*, vol. 18, no. 4, pp. 365-385, Apr. 1988.

[18] ISO IS8807, "LOTOS: A formal description technique," 1989.

[19] P. Merlin and G.v. Bochmann, "On the construction of submodule specifications and communication protocols," *ACM Trans. Program. Lang. Syst.*, vol. 5, no. 1, pp. 1-25, Jan. 1983.

[20] R. Molva, M. Diaz, and J. M. Ayache, "Observer: A run-time checking tool for local area networks," *5th IFIP Workshop Protocol Specification, Testing and Verification*, Toulouse, 1985.

[21] ISO TC97/SC21, DIS 9646/1, and DIS 9646/2, "OSI Conformance testing methodology and framework, Part 1: General Concepts, Part 2: Abstract Test Suite Specification," 1989.

[22] ISO TC97/SC6, IS 8073, "OSI—Connection oriented transport protocol specification."

[23] D. Rafiq, R. Castanet, C. Chraibi, J. P. Goursaud, J. Haddad, and X. Perdu, "Towards and environment for testing OSI protocols," in *Proc. IFIP Workshop Protocol Specification, Verification and Testing*, Toulouse, 1985.

[24] D. Rayner, "A system for testing protocol implementations," *Comput. Networks*, vol. 6, no. 6, Dec. 1982.

[25] CCITT SG XI, Recommendation Z.100, 1987.

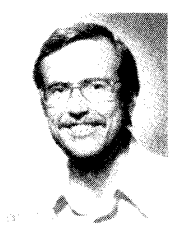
[26] B. Sarikaya and G.v. Bochmann, "Synchronization and specification issues in protocol testing," *IEEE Trans. Commun.*, vol. COM-32, no. 4, pp. 389-395, Apr. 1984.

[27] B. Sarikaya, G.v. Bochmann, and E. Cerny, "A test design methodology for protocol testing," *IEEE Trans. Software Eng.*, vol. SE-13, pp. 518-531, Apr. 1987.

[28] H. Ural and R. L. Probert, "Automated testing of protocol specifications and their implementations," in *Proc. ACM SIGCOMM Symp.*, 1984.

[29] C. Vissers, "Formal description techniques for OSI," in *Proc. IFIP Congress, Information Processing '86*. Amsterdam, The Netherlands: North-Holland, 1986.

[30] H. X. Zeng and D. Rayner, "The impact of the ferry concept on protocol testing," in *Protocol Specification, Testing and Verification (IFIP Workshop)*, M. Diaz, Ed. Amsterdam, The Netherlands: North-Holland, 1986, pp. 533-544.



Gregor v. Bochmann (M'82-SM'84) received the Diploma degree in physics from the University of Munich, Munich, West Germany, in 1968 and the Ph.D. degree from McGill University, Montreal, P.Q., Canada, in 1971.

He has worked in the areas of programming languages, compiler design, communication protocols, and software engineering and has published many papers in these areas. He is currently a Professor in the Département d'Informatique et de Recherche Opérationelle, Université de Montréal, Montréal. His present work is aimed at design models for communication protocols and distributed systems. He has been actively involved in the standardization of formal description techniques for OSI. From 1977 to 1978 he was a Visiting Professor at the Ecole Polytechnique Federale, Lausanne, Switzerland. From 1979 to 1980 he was a Visiting Professor in the Computer Systems Laboratory, Stanford University, Stanford, CA. From 1986 to 1987 he was a Visiting Researcher at Siemens, Munich.

Rachida Dssouli received the "Doctorat d'Université" from Université Paul Sabatier, Toulouse, France, in 1981, and the Ph.D. degree from Université de Montreal, Montreal, P.Q., Canada, in 1986.

She is currently a full Professor at Université Mohamed 1er, Oujda, Morocco.

J. R. Zhao graduated from the Department of Applied Mathematics, Tsinghua University, Beijing, People's Republic of China, in 1965.

She is an Associate Professor of Computer Center at Tsinghua University. Her current areas of interest include formal specification and validation of communication protocols.