16

Design for testability of protocols based on formal specifications

Myungchul Kim, Samuel T. Chanson and Sangjo Yoo*
Korea Telecom Research Laboratories
Sochogu Umyundong 17, Seoul, Korea 137-792
*E-mail:*mckim@sava.kotel.co.kr
* *University of British Columbia*
*Department of Computer Science, University of British Columbia, Vancouver, B .C. Canada V6T 1Z2, E-mail:*chanson@cs.ubc.ca

Abstract

In this paper, we propose a generic scheme which instruments a formal protocol specification automatically to enhance the testability of the implementation. This approach is a special case of design for testability. It is cost-effective considering the entire cycle of protocol development. The advantage of automatic instrumentation is that the user need not pay special attention to testing problems in the design phase. Unlike most other techniques, our scheme also works with existing protocol specifications since it does not affect the original design. Our models address the problems of controllability and observability, and can handle both sequential and concurrent formal protocol specifications.

Keywords

Design for testability, formal specifications, protocols

1 INTRODUCTION

The world's market for computer products is becoming very competitive. In order to win market share, some companies are changing the entire production cycle from a sequential one (called waterfall model) to a parallel model. The so-called concurrent engineering practice [IEEE 91] has emerged in recent years with the aim of making the entire production cycle as parallel and integrated as possible based on many considerations such as manufacturability, testability, performance, quality, installability, reliability, safety, and serviceability. The concept is based on the premise that if these aspects are taken into consideration in the design phase, the total cost and time of the production cycle will be significantly reduced. So far, most of the work on concurrent engineering has been focussed

*currently on leave at the Hong Kong University of Science and Technology.

A. Cavalli et al. (eds.), *Protocol Test Systems VIII*

on hardware products. This paper addresses the design for testability of communication protocols based on formal specifications.

Design for testability on hardware, especially chip testing, is quite mature [Fujiw 85]. In fact, some techniques have been standardized. Chips that are built following the standards can be tested more easily, precisely and in a cost-effective manner using standard techniques through Test Access Ports [Parke 89]. The approach can be classified as gray-box testing, which is in between white-box and black-box testing, as it allows partial access in a controlled fashion to the internals of the system.

The ultimate objectives of both hardware and software testability are similar: avoid or minimize the state space explosion problem in order to reduce the time and cost of testing. However, there are major differences between hardware and software with respect to the testing process [Hoffm 89]. The differences are mainly due to the fact that in hardware testing, we would like to determine whether an implementation (such as a chip) is an accurate copy of the circuit design since the correctness of the circuit design has been established in an earlier verification process. This is because implementations from the proven valid design may still contain serious errors introduced in the manufacturing process. On the other hand, for software, once an implementation is proven correct by testing, copies of the same implementation are always correct. In addition, hardware consisting of distributed circuits or boards often has a single physical global clock which is difficult to provide for software running in a distributed system.

Communication protocols are inherently distributed and concurrent. Understanding and analyzing concurrent programs (or specifications) are much harder than those for sequential ones because the execution order of the sequential programs is fixed (or totally ordered) for a given set of inputs. Since Lamport's pioneering work [Lampo 78], there has been considerable research on the study of concurrency, logical clocks and global states in distributed systems. The logical clock [Matte 89, Fidge 91] is a common technique used to determine whether or not two events are concurrent.

The International Organization for Standardization (ISO) and the International Telecommunication Union - Telecommunication (ITU-T) have developed formal description techniques (FDTs), viz., Estelle, LOTOS, and SDL [ISOb, ISOc, ITU], for specification of communication protocols and services in order to avoid the ambiguity of standard documents written in English. The FDTs can be used in the design phase for providing precise description of the products and for rapid prototyping. In this paper, we assume that the implementations are done according to the design described by an FDT, including the modular structure. In the case of automatic implementation of a protocol from a formal specification, the assumption is usually valid. Even when manually implementing a protocol from a design described by an FDT, the implementation process is easier and more straightforward if the modular structure of the formal specification is followed.

The main contributions of this paper are:

- Automatic instrumentation of formal specifications with respect to design for testability,
- Provision of a practical view of controllability and observability of an implementation under test (IUT) following a formal specification,
- Ability to handle both sequential and concurrent specifications, and
- Provision of a mechanism for error location.

The rest of the paper is organized as follows. Section 2 surveys some related work on design for testability. The generic instrumentation schemes for testability in terms of controllability and observability on sequential specifications, and controllability on concurrent specifications are presented in Section 3. In order to provide observability on concurrent specifications, a formal model is proposed in Section 4. Finally, Section 5 concludes the paper.

2 RELATED WORK

Existing work does not offer a standard mechanism which provides testability for formal specifications. The following is a brief survey of some papers on design for testability of software, and in particular, protocols.

Dssouli and Fournier [Dssou 90] suggested a modification in the software development process to accommodate the notion of testability between the implementation phase and the design phase of the software development cycle. By comparison, our model provides a generic approach to supporting design for testability on formal specification by automatic instrumentation.

After investigating the meaning of software testability formally, Freedman [Freed 91] defined domain testability by applying the concepts of observability and controllability which are used in assessing the testability of hardware components to software. A domain testable program is observable and controllable in that it does not exhibit any test input-output inconsistencies. This concept applies to sequential programs (or specifications) only.

Ellsberger and Kristoffersen [Ellsb 92] identified some aspects of SDL specifications that are difficult to test, i.e., asynchronous communication, the time model, and nondeterminism. They also presented suggestions for improving testability in these aspects of SDL which are difficult to be generalized to other specification techniques.

Petrenko and Dssouli [Petre 93] proposed a testability metric based on Finite State Machines (FSM) under the complete fault coverage assumption. The metric can be applied to multiple FSMs.

In [Vuong 93], a framework was presented with respect to the factors that affect testing and testability of communication protocols in the context of analysis, design, implementation, and testing phases of the protocol engineering cycle. While the framework provides a good basis to reason about protocol design for testability, no detailed solution is given that is directly usable on specifications given in FDTs.

ITU-T and ISO have created a task force on "Formal Methods in Conformance Testing"[FMCT]. Specification styles for testability are proposed. The style is a guideline for increasing the testability of specifications given in FDTs. There is no easy way to automate the recommendations.

3 INSTRUMENTATION SCHEMES TO ENHANCE TESTABILITY

In the ISO conformance testing methodology, an IUT is viewed as a black-box for testing purposes [ISOa]. The protocol specification may consist of a single module (i.e., sequential)

or multiple modules (i.e., concurrent). Again, we assume that an FDT is used in the design phase and the modular structure described by the FDT is preserved in the implementation.

The objective of design for testability for protocols is to provide precise and efficient ways of testing the protocol implementation (i.e., the IUT). We believe that testing will be made much easier if some internal behavior of the IUT are exposed to the testers, i.e., treating the IUT as a gray-box for testing purposes. Even though we consider the IUT as a gray-box, there are inherent uncertainties due to nondeterminism. The sources of nondeterminisms are summarized below:

1. Nondeterminism in the specification: In order to make the specification concise and provide flexibility to the implementors, FDTs allow the specification of nondeterministic actions. These intentional nondeterministic actions can be translated into deterministic ones during the implementation phase in a standard way (like the translation from Nondeterministic Finite State Automata to Deterministic Finite State Automata).

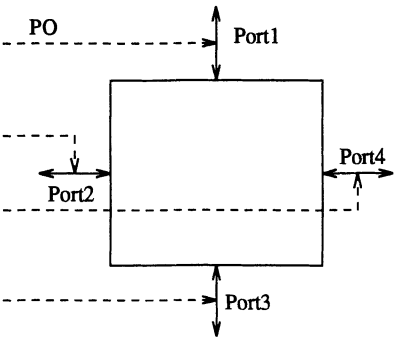

Figure 1 Points of observation.

2. Nondeterminism arising from the concurrent environment: Concurrent behavior in distributed systems is difficult to specify and analyze. In a distributed system, it is generally not possible to determine the total order of events in the system because of the lack of a global clock. Only the partial order of events in the system can be computed, and for concurrent events there is no general way of telling which one has occurred first [Kim 93]. This unintentional nondeterminism is inherent in distributed systems.
3. Nondeterminism due to the black-box approach: The order of messages observed from the Points of Observation (PO) and/or the Points of Control and Observation (PCO) between modules as shown in Figure 1 does not always correspond to the order of event occurrences caused by these messages inside the IUT.

For example, consider two input messages where the input message at port 1 arrives at the IUT earlier than the input message at port 2. However, the messages may be executed in the reverse order of arrival. In order to determine the right order, we may need to enumerate every possible combination of message inputs and outputs beforehand and compare them with the observed behavior. This unintentional nondeterminism may cause state space explosion in trace analysis.

Controllability and observability can be applied at three levels: data, transition and module. Controllability (observability) on data means the ability to assign (print) values to (of) internal variables or parameters of the interaction primitives. However, in the context of conformance testing, it is not valid to modify the internal data. Controllability at the transition/module level is the ability to select specific transitions/modules for testing. Observability at the transition/module level refers to the ability to observe the execution order of the transitions/modules. In this paper, we propose techniques for controllability and observability at the transition and the module levels only. To simplify the discussion, we assume that each module is described in Normal Form Specification (NFS) [Sarik 87]. The NFS is written in Estelle [ISOb] which consists of a single module without procedures, functions, and *while* statements.

3.1 Overall framework

Formal specifications with enhanced testability can be produced by an automatic instrumentation process on the original formal specifications as shown in Figure 2. The instrumentation schemes proposed are generic in the sense that they work for all specifications that satisfy the assumptions given in the previous section. Therefore, the process relieves the protocol designer from having to be concerned with testability issues in the design phase. The instrumented specification can be used to generate the implementation with enhanced controllability and observability automatically (or semi-automatically) using existing tools, or manually.

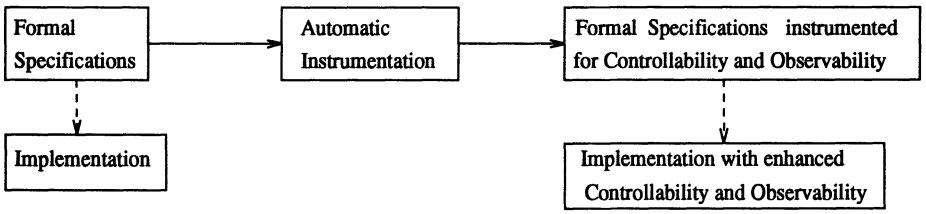

Figure 2 Overall framework.

3.2 Controllability on sequential specification

The issue here is to be able to select the proper transition to be tested. For controllability on a sequential specification, we propose that each transition be instrumented by appending a particular condition just after the PROVIDED clause (if exists). The condition

is used to select a particular transition out of possibly more than one candidate transitions due to nondeterminism. The instrumented specification will also work for existing testers not designed to take advantage of the proposed scheme. However, in that case, the selection of nondeterministic transitions cannot be controlled.

The scheme is very simple and is given below. Operations 1, 2 and 3 below are just to specify if the controllability feature should be activated. Operation 4 says that if the existing condition for firing a condition is satisfied and the controllability option is selected, then the transition is fired only if it is the transition specified. Note that the instrumentation does not add illegal behaviour not permitted by the original specification. Its only function is to select a transition out of a set of possible transitions (due to nondeterminism) for testing.

Rules for instrumentation:

1. Interaction primitives "TEST_control" and "TEST_uncontrol" are included in the channel interaction primitives of a sequential specification.
2. The global variable "TEST_var_control = 0" and state set "stateset all = [list_of_all_states]" are defined.
3. The following transitions are added. If the sequential specification or the IUT receives "TEST_control" / "TEST_uncontrol" at whatever state when we need to control / not to control the specification or IUT, set the global variable "TEST_var_control = 1" / "TEST_var_control = 0", respectively.

```
from all                          from all
when TEST_control                 when TEST_uncontrol
begin                             begin
   TEST_var_control = 1;             TEST_var_control = 0;
end                               end
```

4. A particular condition is appended to the existing condition in the PROVIDED clause as follows:

```
PROVIDED (existing_condition) AND ((!TEST_var_control) OR
     (Input_Primitive.Data = id_of_tr))
```

In order to run an IUT with/without the controllability feature, interaction primitives
"TEST_control" / "TEST_uncontrol" are provided. For example, if the feature is to be activated, the interaction primitive "TEST_control" is sent by the tester to the IUT. In that case, the global variable "TEST_var_control" is set to '1' which is used in Operation 4.

By using transition identifier "id_of_tr" in the data field of the input interaction primitive "Input_Primitive.Data", a particular transition can be selected to be fired. Note that no additional PCO is used. If we use an additional PCO for controllability, we will encounter unintentional nondeterminism arising from the concurrent environment as discussed in Section 3. For this reason, the information for controllability is stored in the

data field of the interaction primitives. This approach is different from testability in hardware. Hardware testing supports additional PCOs since a global physical clock is usually available.

3.3 Observability on sequential specification

In order to deal with the problems of unintentional nondeterminism due to the black-box approach described in Section 3, an additional output statement with a message identifying the transition that has just been executed is inserted at the end of every transition in the Estelle (NFS) specification. For each module, this output statement is directed to a particular port (PO) created for the purpose of observation where a log of the trace will be recorded. The trace provides the precise execution order of transitions within the module independent of the number of interaction ports in the module. Note that for debugging as well as for testing purposes, it is desirable to have a trace of the execution events even when the controllability feature is used.

The scheme for observability on sequential specification is given below. The instrumentation does not change the semantics of the original specification as only output statements are added. Again, the scheme allows the observability feature to be 'turned off' so that the instrumented specification can work with conventional testers.

1. Interaction primitives "TEST_observe" and "TEST_unobserve" are included in the channel interaction primitives of a sequential specification.
2. Global variable "TEST_var_observe = 0" and state set "stateset all = [list_of_all_states]" are defined.
3. The following transitions are added in the specification. If the sequential specification or the IUT receives "TEST_observe" / "TEST_unobserve" on whatever states, set the global variable "TEST_var_observe = 1" / "TEST_var_observe = 0", respectively.

```
from all
when TEST_observe
begin
   TEST_var_observe = 1;
end
```

```
from all
when TEST_unobserve
begin
   TEST_var_observe = 0;
end
```

4. At the end of each transition, the following statement is inserted. The statement outputs to a particular PO ("PO_one" say) an interaction primitive "OBSERVE" whose parameter is the identifier of the transition"id_of_tr".

```
if(TEST_var_observe)   OUTPUT PO_one.OBSERVE(id_of_tr);
```

Interaction primitives "TEST_observe"/ "TEST_unobserve" are provided in order to select/unselect the observability feature. If the feature is needed, the tester sends the interaction primitive "TEST_observe" to the IUT. In that case, the global variable "TEST_var_observe" is set to 1 which is used in Operation 4. If the performance of the IUT is critical or we do not need to observe the IUT, the interaction primitive "TEST_unobserve" is sent instead.

The proposed mechanisms also allow us to control and observe spontaneous transitions as well as transitions containing timers which are difficult problems in conformance testing. Debugging and error location are also made easier since a complete trace of the transition path is maintained.

3.4 Controllability for concurrent specification

Since an NFS specification consists of a single module, we view a concurrent specification as consisting of multiple NFSs. According to our assumption, an IUT derived from a concurrent specification retains the modular structure of the concurrent specification. Also we assume that Estelle, which is the description technique of NFS, supports parameters in the configuration statements (such as init, connect, attach, release, disconnect, and detach) in order to simplify the discussion. Controllability in the concurrent environment means the ability to select a set of specific modules to be tested. The scheme for controllability of concurrent specification is presented below.

1. For those interaction ports which are not exposed to the outside environment, mirrored interaction ports (MIPs) with the same interaction primitives are created in the outermost module as shown in the example in Figure 3.

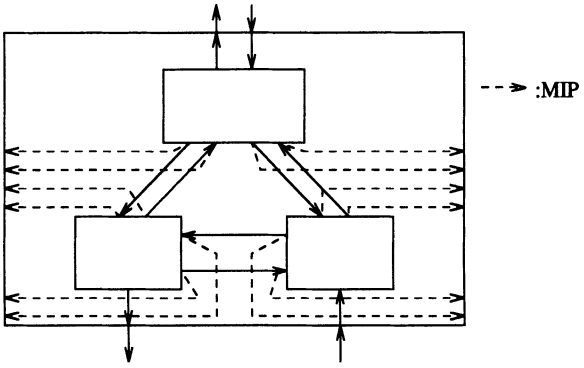

Figure 3 Mirrored interaction ports.

2. Depending on which modules are to be tested, we need to configure the modular structure of the IUT accordingly. Thus a capability for dynamic configuration of the module structure is needed.

A way to do this is to provide the following instrumentation:

```
stateset all = [list_of_all_states];
DATA = ...;
```

```
STATEMENTTYPE = (INIT, CONNECT, ATTACH, RELEASE, DISCONNECT, DETACH)
channel example(user, provider)
by user
    .
    CONFIGURE(ACTION:STATEMENTTYPE; DATA_A;DATA, DATA_B:DATA);
    .


from all
when X.CONFIGURE
provided ACTION = INIT
    begin
       init DATA_A with DATA_B;
     end

from all
when X.CONFIGURE
provided ACTION = CONNECT
    begin
       connect DATA_A to DATA_B;
     end
 .
 .
 .
```

Since all configuration statements of Estelle are provided through the interaction primitive "CONFIGURE" and its parameters "ACTION", "DATA_A" and "DATA_B", it is possible to configure the IUT dynamically at any time. The interaction port "X" can either be a port in the original formal specification or a MIP in the formal specification that has been transformed for testability. For example, if we want to initialize a module in an IUT, we send the interaction primitive "CONFIGURE" to the module with ACTION "INIT", module variables in "DATA_A" and module type in "DATA_B".

Alternately, we may enumerate every possible combination of modules, and then make a set of transitions providing a configuration capability for each combination. This scheme may cause transition explosion if the number of modules is large, but will not need the assumption that parameters are supported in the configuration statements of Estelle.

A formal model is proposed in the following section to deal with the problem of observability based on concurrent specification.

4 FORMAL MODEL

In this section, we shall use the term module to mean an implementation executing as a single process. Let us assume that a concurrent module M_{conc} consists of a set of sequential modules M_1, ..., M_i, ..., M_n; a concurrent formal specification S_{conc} consists of a set of sequential formal specifications S_1, ..., S_i, ..., S_n; and, a concurrent trace T_{conc} from M_{conc}

consists of a set of sequential traces T_1, ..., T_i, ..., T_n from individual modules M_1, ..., M_i, ..., M_n (See Figure 4). The concurrent formal specification S_{conc} is assumed to be error-free and conform to the standards.

Definition 1 A sequential trace T_i of module M_i consists of a chronologically ordered sequence of event vectors t_i^k, k = 1, 2, ... If M_i has p interaction ports, then the event vector $t_i^k = <(in_{i,1}^k, out_{i,1}^k), ..., (in_{i,p}^k, out_{i,p}^k)>$ where $in_{i,j}^k$ and $out_{i,j}^k$ are the input and output messages respectively at interaction port j associated with the k-th event vector in trace T_i. $in_{i,j}^k$ and $out_{i,j}^k$ may be null.

Notice that we have a trace structure equivalent to the one presented in our earlier paper [Kim 91] which dealt with trace analysis based on single modules. An event vector consists of zero or one input, and x output messages where $0 < x < p$.

Definition 2 A sequential module M_i is pointwise conformant to the formal specification S_i on an event vector t_i^k if the behavior of the event vector is permissible (i.e., contained) in the specification. This is denoted as pconf(M_i, S_i, t_i^k).

Note that even if more than one path (or transition) of the specification satisfies the event vector, the pconf relation still holds based on the above definition.

Definition 3 A trace T_i conforms to the sequential module M_i, written as $conf_{seq}(M_i, S_i, T_i)$ iff pconf(M_i, S_i, t_i^k) for all t_i^k in T_i.

The sequential trace T_i from Definition 1 consists of the external behavior of a module M_i with respect to its environment (i.e., the other modules). If $conf_{seq}(M_i, S_i, T_i)$ is valid, the trace T_i conforms to the specification of M_i. In addition, the input/output messages to/from the other modules in T_i are also valid since we assume that S_{conc} is error-free.

Definition 4 A concurrent trace T_{conc} of a concurrent module M_{conc} conforms to the concurrent specification S_{conc} of M_{conc}, written as $conf_{conc}(M_{conc}, S_{conc}, T_{conc})$ iff $conf_{seq}(M_1, S_1, T_1) \wedge \ldots \wedge conf_{seq}(M_i, S_i, T_i) \wedge \ldots \wedge conf_{seq}(M_n, S_n, T_n)$, i.e., $\wedge_{i=1}^{n} conf_{seq}(M_i, S_i, T_i)$.

Note that the trace analysis proposed does not depend on the communication scheme (synchronous or asynchronous). The trace analysis of concurrent modules with respect to concurrent formal specifications provides verdicts based on a set of trace analyses of single modules with sequential formal specifications.

Since there are two types of problems that could arise in a concurrent (or distributed) environment but not in a sequential environment, namely deadlocks and data races, we need to show how they can be detected. A concurrency model was proposed for this purpose based on time-event sequence diagrams [Kim 93]. To detect data races, we note that a set of $conf_{seq}(M_i, S_i, T_i)$, i.e., trace analyses with respect to the single modules in the system, enable us to construct the time-event sequence diagram used by the concurrency model. The concurrency model provides a way to identify the sets of concurrent events which can be checked for data races by determining if there are read/write or write/write conflicts to shared variables. An example in Ada is given in [Kim 93]. The external behavior (i.e., a trace T) of a module M which does not conform to its specification S may cause deadlocks. Deadlock detection depends on the communication scheme used. In the

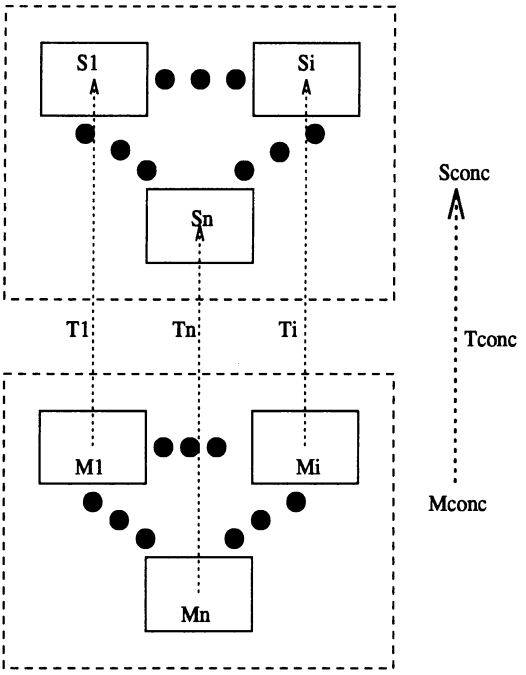

Figure 4 Trace analysis based on concurrent specifications.

synchronous scheme, non-conformance in terms of input or output message may lead to deadlocks. However, asynchronous schemes will not cause deadlocks even though input or output messages from a module M do not conform to the specification S. This is because asynchronous messages do not block and wait, thus removing a necessary condition for deadlock to occur. So far, most of the work in protocol testing has been concerned with the sequential aspects of protocols. The ideas presented in this section is a first step in dealing with testing of concurrent modules which is an extension of our previous work on trace analysis of sequential module [Kim 91]. More research is needed to work out the theory and to build the tools.

5 CONCLUSIONS

The ISO conformance testing methodology treats an IUT as a black-box which may consist of one or more modules. This makes testing difficult since nondeterminism may arise from the tester's point of view because internal actions are not observable. In this paper, we have proposed a generic scheme to provide precise and efficient means of instrumenting

the formal specification to enhance controllability and observability in testing protocol implementations consisting of a single module. A framework for testing multiple concurrent modules is also presented. The proposed schemes can work with existing specifications and testers also. To our knowledge, this is the first work that has adopted this approach. Future work includes techniques for data level design for testability (so that we can pinpoint the cause of an error), and the theory and tools to test concurrent modules.

REFERENCES

[Dssou 90] Dssouli R. and Fournier R. (1990) Communication Software Testability. *The 3rd Int'l Workshop on Protocol Test Systems*, McLean, Virginia.

[Ellsb 92] Ellsberger J. and Kristoffersen F. (1992) Testability in the context of SDL. *IFIP Symposium on Protocol Specification, Testing, and Verification, XII*, Lake Buena Vista, Florida.

[Fidge 91] Fidge C. (1991) Logical Time in Distributed Computing Systems. *IEEE Computer*, vol. **24**, no. **8**, 28 – 33.

[FMCT] ISO SC21 P.54 and ITU-T SG10 Q.8 (1994) Formal Methods in Conformance Testing.

[Freed 91] Freedman R. S. (1991) Testability of Software Components. *IEEE Tr. on Software Engineering*, vol. **17**, no. **6**, 553 – 564.

[Fujiw 85] Fujiwara H. (1985) Logic Testing and Design for Testability. MIT Press.

[Hoffm 89] Hoffman D. (1989) Hardware Testing and Software ICs. *The Pacific NW Software Quality Conference*, 234 – 244.

[IEEE 91] IEEE (1991) Special Report on Concurrent Engineering. *IEEE Spectrum* , vol. **28**, no. **7**.

[ISOa] ISO IS 9646 (1991) OSI Conformance Testing Methodology and Framework.

[ISOb] ISO 9074 (1989) ESTELLE - A Formal Description Technique Based on an Extended State Transition Model.

[ISOc] ISO 8807 (1989) LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior.

[ITU] ITU-T (1988) Specification and Description Language (SDL) Recommendations Z.100. *ITU-T Blue Book.*

[Kim 91] Kim M., Chanson S. T. and Vuong S. T. (1991) Protocol Trace Analysis based on Formal Specifications. *Fourth Int'l Conference on Formal Description Techniques*, Sydney, Australia, 399 – 414.

[Kim 93] Kim M., Chanson S. T. and Vuong S. T. (1993) Concurrency Model and Its Application to Formal Protocol Specifications. *IEEE INFOCOM*, San Francisco, California, 766 – 773.

[Lampo 78] Lamport L. (1978) Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. ACM*, vol. **21**, no. **7**, 558 – 565.

[Matte 89] Mattern F. (1989) Virtual Time and Global States of Distributed Systems. *Parallel and Distributed Algorithms*, North-Holland, 215–226.

[Parke 89] Parker K. P. (1989) The Impact of Boundary Scan on Board Test. *IEEE Design & Test of Computers*, vol. **6**, no.**4**, 18 – 30.

[Petre 93] Petrenko A., Dssouli R. and et al. (1993) On Evaluation of Testability of Protocol Structures. *The 6th Int'l Workshop on Protocol Testing Systems*, Pau, France, 115 –

127.

[Sarik 87] Sarikaya B., Bochmann G. and et al (1987) A Test Design Methodology for Protocol Testing. *IEEE Tr. on Software Engineering*, vol. **13**, no. **5**, 518 – 531.

[Vuong 93] Vuong S. T., Loureiro A. A. F. and Chanson S. T. (1993) Toward a Framework for the Design for Testability of Communication Protocols. *The 6th Int'l Workshop on Protocol Testing Systems*, Pau, France, 91 – 111.