Revision May 1990

PROTOCOL ENGINEERING

Contribution to the
Concise Encyclopedia of Software Engineering

Gregor v. Bochmann
Departement d'informatique et de recherche operationnelle
Universite de Montreal

January 1990

Introduction

Communication protocols are the rules that govern the communication between the different components within a distributed computer system. In order to organize the complexity of these rules, they are usually partitioned into a hierarchical structure of protocol layers, as examplified by the seven layers of the standardized OSI Reference Model [Suns 89, Larm 88].

As they develop, protocols must be described for many purposes. Early descriptions provide a reference for cooperation among designers of different parts of a protocol system. The design must be checked for logical correctness. Then the protocol must be implemented, and if the protocol is in wide use, many different implementations may have to be checked for compliance with a standard. Although narrative descriptions and informal walk-throughs are invaluable elements of this process, painful experience has shown that by themselves they are inadequate.

The informal techniques traditionally used to design and implement communication protocols have been largely successful, but have also yielded a disturbing number of errors or unexpected and undesirable behavior in most protocols. The use of specification written in natural language gives the illusion of being easily understood, but leads to lengthy and informal specifications which often contain ambiguities and are difficult to check for completeness and correctness. The arguments for the use of formal specification methods in the general context of software engineering [Somm 89] apply also to protocols.

The following activities can be identified within the protocol engineering process. These activities can be partially automated if a formal protocol specification is used [Boch 87c].

(a) Protocol design: The protocol specification is developed based on the communication service to be provided by the protocol. The protocol also depends on the underlying (existing) communication service; e.g. the protocol may have to recover from transmission errors or lost messages if the underlying service is unreliable. The design process is largely based on intuition.

(b) Protocol design validation: The protocol specification must be checked (1) for logical consistency, (2) to provide the requested communication service, and (3) to provide it with acceptable efficiency.

(c) Implementation development: The protocol implementation must satisfy the rules of the protocol specification; the implementation environment and the user requirements provide additional constraints to be satisfied by the implementation. The implementation may be realized in hardware or software.

(d) Conformance testing and implementation assessment: The purpose of conformance testing is to check that a protocol implementation conforms to the protocol specification, that is, that it satisfies all rules defined by the specification. This activity is especially important for interworking between independently developed implementations, as in the case of OSI standards. The testing of an implementation involves three sub-activities: (1) the selection of appropriate test cases, (2) the execution of the test cases on the implementation under test, and (3) the analysis of the results obtained during test execution. The sub-activities (1) and (3) use the protocol specification as a reference.

1. Protocol specification

Figure 1(a) shows a communication system from the point of view of two users. The users interact with the communication service through interactions, called service primitives, exchanged at so-called service access points. The definition of the behavior of the box which extends between the two users is called the service specification. It defines the local rules of interactions at a given service access point, as well as the so-called end-to-end properties which relate the interactions at the different access points and represent the communication properties, such as end-to-end connection establishment or reliable data transfer.

Figure 1(b) shows a more detailed view of the service box showing two so-called protocol entities which communicate through an underlying, more simple communication service. The definition of the required behavior for a protocol entity is called the protocol definition, and involves the interactions at the upper and lower service access points. In addition, the protocol specification usually identifies different types of so-called protocol data units (PDU, or messages) which are coded and exchanged between the protocol entities through the underlying medium.

As an example, figure 2 shows a diagram of a finite state machine representing part of a connection estabilishment protocol. The entity is initially in the IDLE state. When it receives a CONrequest interaction from its user through the upper service access point, it makes a transition into an intermediate state and produces as output a CR-PDU (connect request) which is coded and sent to the peer protocol entity in the form of a parameter of a data transmission interaction passed to the underlying communication service. The second transition of the diagram corresponds to the reception of data from the peer entity which corresponds to a (encoded) CC-PDU (connect confirm), which results in the

CONconfirmation interaction with the user and leads the protocol entity into the OPEN state. It is important to note that this specification is very partial, since it ignores all rules concerning various parameters which are associated with the interactions at the service access points and with the PDUs.

Whereas the PDUs and their encoding must be precisely defined for assuring compatibility between different protocol entities, the service primitives at the service access point need not be defined in detail, but only in terms of their abstract meaning which is the basis for the service specification. The abstract definition of the service primitives and the local rules of the service specification represent the rules that must be satisfied by an interface between the protocol and its user. Each implementation of the protocol has to decide on the mapping of these abstract rules to the real interface that is used in the implementation project.

The protocol specification should assure that any two implementations that satisfy this specification are compatible and provide the corresponding communication service. It is therefore the basis for any protocol implementation. The corresponding service specification is used as reference in the validation of the protocol specification, and is also the basis on which the design of the communication behavior of the user processes can be based. In addition, the service specification provides a platform for the design of gateways between systems using incompatible protocols [Boch 90b].

Various languages and formalism can be used for writing formal protocol and service specifications, including finite state machines (FSM's) and standarized formal description techniques (FDT's) [Budk 87, Bolo 87, Beli 89] which are intended for the development of formal specifications of OSI protocols and services. A more detailed discussion of these issues may be found in [Boch 89d].

2. Protocol design validation

Once a specification has been created in its initial form, it must be validated. This is usually a difficult task. In the case of formal specifications the following methods and tools can be used for this purpose.

Static analysis: Based on the text of the specification, static analysis is quite useful to find clerical errors. Tools, normally related to a particular specification language, exist for the checking of context-free syntax, scope rules, type conformance, and other semantic conditions. The analysis corresponds to what compilers do for programming languages.

Dynamic analysis: In constrast to static analysis, the dynamic analysis of a specification considers some kind of "execution" of the specified system. Because of the large number of possible situations that may occur during an execution of the system, dynamic analysis is usually much more difficult to do than static analysis. However, it can detect errors which are not detectable by static methods. A more detailed tutorial of this topic can be found in [Pehr 89].

The dynamic methods can be classified into exhaustive and simulation methods. The exhaustive methods consider all possible situations that may occur during the execution of the specified system. In most cases, however, there are too many cases to be considered. Therefore these methods are usually applied to a simplified description of the system. The best-known methods are related to the exhaustive reachability analysis for systems specified as a collection of finite state machines. The verification of programs and assertions, and other methods involving theorem proving, also belong to this class. They can be applied to the complete specifications; they are, however, difficult to apply to specifications of the size that are found in most practical applications.

The simulation methods validate only certain selected paths among all the possible executions. However, they can be applied to real-size specifications, provided that the specification language allows some form of simulated execution. In order to reduce the large state space to be explored by exhaustive reachability analysis, certain authors have proposed random and probability-based exploration procedures [Maxe 87].

Two goals can be distinguished for the protocol validation process: (1) checking that the specification satisfies so-called general properties, and (2) checking that the specification satisfies the properties defined by the communication service which the protocol is supposed to provide. Typical general properties to be satisfied by any specification include the checking for deadlocks (system blocking), verifying that the specification defines appropirate behavior for any message that might be received, or (for certain applications) that the number of messages in transit remains below a given bound.

Validation by reachability analysis proceeds by exhaustively exploring all the possible interactions of two (or more) entities within a protocol layer. A composite or global state of the system is defined as a combination of the states of the cooperating protocol entities and the lower layer connecting them. From a given initial state, all possible transitions (user commands, time-outs, message arrivals) are generated, leading to a number of new global states. This process is repeated for each of the newly generated states until no new states are generated (some transitions lead back to already generated states). For a given initial state and set of assumptions about the underlying layer (the type of service it offers), this type of analysis determines all of the possible outcomes that the protocol may achieve.

Reachability analysis is well suited to checking the general correctness properties described above because these properties are a direct consequence of the structure of the reachability graph. For instance, global states with no exits are either deadlocks or desired termination states. It can also be used for comparison with the service specification if the latter is given in the form of a transition system. The major difficulty of reachability analysis is the so-called "state space explosion" because the size of the global state space may grow rapidly with the number and complexity of protocol entities involved and the underlying layer's services. Techniques for dealing with this problem are discussed in chapter 17 of [Suns 89].

3. Implementation and simulation tools

The requirements to be satisfied by a protocol implementation include the protocol specification and usually additional constraints which are particular to the implementation project. These additional constraints may define such questions as "how does the implementation react to unexpected (invalid) user interactions?", "how many simultaneous connections should be supported?", or "what should be the performance of the implementation?". Based on these requirements, the implementation is usually developed in several steps of refinement using the standard software or hardward design and implementation methods.

In the case that a formal protocol specification is available, the following types of tools can be used to partially automate the implementation process:

(a) Generation of program skeletons from FSM-oriented specifications. Such skeletons must be completed with the code related to the handling of interaction parameters and the updating of internal program variables.

(b) Generation of program source code from FDT specifications. As explained in [Boch 87h], the abstract formal protocol specification must usually be refined before the program generation tool can be applied. Large parts of the implementation code can be automatically generated from detailed formal specifications.

(c) Generation of encoding and decoding routines from ASN.1 specifications of PDU's. ASN.1 is a notation used for defining the data structures of PDU parameters for OSI Application layer protocols. Because of the regular coding scheme used with ASN.1, the (de-)coding function can be automated. Existing tools either interpret the given ASN.1 description of the protocol dynamically, or generate, in source code, the specialized encoding and decoding routines for the given protocol.

Instead of generating executable source code from the formal protocol specification, many specification tools allow for an interpretive execution of the formal specification. The main drawback of this approach is reduced efficiency. However, both of these appoaches, executable source code and interpretive execution, can be used to perform simulations of the protocol system, which are useful for the dynamic validation and analysis of the protocol specifications.

4. Protocol implementation testing

The validation of a new implementation usually includes some testing activities. In the case of protocol implementations, two validation concerns are distinguished:

(a) Protocol conformance testing is concerned with checking that all rules defined by the protocol specification (requirements for compatibility with other systems) are satisfied by the implementation [Rayn 87].

(b) Implementation assessment, in a more general context, is concerned with also verifying other properties of the implementation (see Section 3), possibly including performance parameters.

4.1. System architectures for protocol testing

Several system architectures for conformance testing have been identified in the context of OSI standardization [Rayn 87]. These architectures can also be used for implementation assessment. Besides the local architecture, where the implementation under test (IUT) and the tester reside within the same computer, several distributed architectures have been defined. Figure 3 shows the so-called "Distributed" architecture, where the IUT and a test user, called "upper tester", reside in a computer that is connected through a network to a remote test system computer. The latter contains a module called "lower tester" which communicates through the network with the IUT.

It is important to note that complete testing of a protocol implementation implies the observation of the interactions at the upper and lower interfaces. Nevertheless, in many cases the so-called "Remote" test architecture is used which does not have an upper tester and therefore does not check the communication service provided to the user.

The Remote and Distributed test architectures have the advantage that the test system resides in a separate computer and can be accessed over distance from a variety of systems under test. In the context of OSI, certain test centers provide public conformance testing services which are accessed over public data networks.

The Distributed test architecture presents some difficulty concerning the synchronization between the upper and lower testers, since they reside in different computers and communicate only indirectly through the IUT. A separate communication channel is usually introduced, sometimes in the form of a terminal connection to the remote test system, which allows the operator at the system under test to coordinate the activities of the test system with the operation of the system under test. Various test coordination protocols, sometimes using a separate channel, have also been developed for automatically coordinating the actions of the upper and lower testers.

4.2. Development of test cases

Methods for the development of test cases have received much attention recently in relation with conformance testing of communication protocols [Sari 89c]. The purpose of a test selection method is to come up with a set of test cases, usually called "test suite", which has the following conflicting properties:

(a) The test suite should be relatively short, that is, the number of test cases should be small, and each test case should be fast and easily executable in relation with the implementation under test (IUT).

(b) The test suite should cover, as much as possible, all faults which some implementation may contain.

Existing test selection methods differ in kind of compromise which is reached between these two conflicting objectives, and in the amount of formalism which is used to define the method. In the case that a formal specification of the protocol is available, the test selection and fault analysis can be based on this specification [Sari 89c, Boch 89m]. It is important to note that these issues also arise in the more general context of software and hardware testing, and many methods developed in those areas can be adapted to protocol testing.

Many test selection methods have been developed for the case that the specification of the system to be tested is given in the form of a finite state machine (FSM) [Sari 84]. These methods can be evaluated in relation with a fault model which is based on the FSM formalism. Two kinds of faults are considered: (1) output errors, where the implementation produces a wrong output for a given transition, and (2) transfer errors, where the implementation goes into a wrong state for a given transition. Most methods provide test suites that detect all output errors, but not necessarily all transfer errors; if nothing is known about the number of states of the implementation, no guarantee can be made for the detection of transfer errors.

The test case selection methods developed for software (see for instance [Howd 78]) can be adapted to the area of protocol testing. However, in contrast to software testing, where the program code is often taken as the basis for the selection of test cases, the test cases for protocol conformance testing are based on the protocol specification, and the protocol implementation is considered a black box where only its interactions at the upper and lower interfaces are visible. In this context, it is also possible to combine the FSM test methods with the testing of the data flow functions which are defined by the formal protocol specification in terms of the parameters of input and output interactions and their relation with internal state variables [Sari 87].

Special precautions must be taken in view of the problem of synchronization between the upper and lower testers in the distributed test architectures [Sari 84]. In the area of OSI conformance testing, standardized suites of test cases have been developed for several protocols by the standardization committees. There is a tendancy of specifying so-called generic test cases which are formulated independently of the testing architecture. They must later be adapted for execution in a particular architecture.

Many OSI protocols allow for a large number of implementation options. Therefore the tests executed during OSI conformance testing must be adapted to the options realized by the implementation. The (standardized) suite of test cases for a given protocol usually contains separate tests for each of the possible options. For the testing of each protocol implementation, the selection, from the test suite, of test cases to be executed is based on the so-called "protocol implementation conformance statement" (PICS) which states the options supported. For certain protocols this selection process, called "test selection" in the OSI context, is very complex and justifies its automation.

4.3. Test result analysis

During or after the execution of the test cases, the output produced by the IUT must be analysed in order to determine whether the produced output conforms to the specification. In most cases, the expected (correct) output is already defined by the test case. In the case of (standardized) OSI conformance test cases, usually several different possible outputs are foreseen by the test case description, including for each possibility a verdict which could be "pass" (positive test outcome), "fail" (error detected), or "inconclusive" (allowed behavior, but the behavior to be tested could not be observed).

In the case that a formal specification is available, the trace of observed input and output interactions of the IUT can be automatically analysed in order to determine whether it conforms to the specification [Boch 89m]. Such an automatic test result analysis can be useful in the following situations:

(a) In the case that the IUT is subjected to ad hoc or random tests, for instance during debugging, or for complementing the standard conformance tests.

(b) For arbitration testing. This involves two or more systems that have already been tested individually, and which nevertheless turn out to have problems interworking. Figure 4 shows a testing architecture where the tester passively observes the PDU's exchanged between the different systems. The tester includes a trace analysis module which checks the observed trace in respect to the specifications of all the systems and will notify any detected error.

(c) For validating the defined test cases. A suite of test cases for a given protocol can be very voluminous. Since most test cases are developed by informal methods, they may contain errors, that is, wrong verdicts. Automatic trace analysis can be used to check the verdicts of test cases with the formal specification of the protocol.

It is to be noted that a distributed test architecture somehow limits the power of error detection, since neither the upper, nor the lower tester have a global view of all the interactions in which the IUT is involved. Each tester only observes a local interface, and the relation between interactions taking place at different interfaces is difficult to be taken into account [Boch 89m].

REFERENCES

[Beli 89] F. Belina and D. Hogrefe, *The CCITT-Specification and Description Language SDL*, Computer Networks and ISDN Systems, Vol. 16, pp.311-341, 1989.

[Boch 87c] G. v. Bochmann, *Usage of protocol development tools: the results of a survey*, (invited paper), 7-th IFIP Symposium on Protocol Specification, Testing and Verification, Zurich, May 1987, pp.139-161.

[Boch 89d] G. v. Bochmann, *Protocol specification for OSI*, to be published in Computer Networks and ISDN Systems.

[Boch 89m] G. v. Bochmann, R. Dssouli and J. R. Zhao, *Trace analysis for conformance and arbitration testing*, IEEE Tr. on Softw. Eng., Nov. 1989, pp.1347-1356.

[Boch 90b] G. v. Bochmann and P. Mondain-Monval, *Design Principles for communica - tion gateways*, to be published in IEEE Tr. on Selected Areas in Communications, 1990.

[Bolo 87] T. Bolognesi and E. Brinksma, *Introduction to the ISO Specification Language Lotos*, Computer Networks and ISDN Systems, vol. 14, no. 1, pp.25-59, 1987.

[Budk 87] S. Budkowski and P. Dembinski, *An introduction to Estelle: a specification language for distributed systems*, Computer Networks and ISDN Systems, vol. 14, no. 1, pp.3-23, 1987.

[Howd 78] W. E. Howden, *A survey of dynamic analysis methods*, in Software Testing and Validation Techniques, E. Miller and W.E. Howden eds., IEEE EHD 138-8, 1978.

[Larm 88] J. Larmouth, K.G. Knightson, T. Knowles, *Standards for Open Systems Interconnection*, McGraw-Hill, 1988.

[Maxe 87] N. F. Maxemchuk and K. Sabnani, *Probabilistic verification of communication protocols*, in Proc. IFIP Symp. on Protocol Specification, Testing and Verification VII, North Holland Publ., 1987, pp.307-320.

[Pehr 90] B. Pehrson, *Protocol verification*, to be published in Computer Networks and ISDN systems.

[Rayn 87] D. Rayner, *OSI Conformance testing*, Computer Network and ISDN Systems, 14 (1987), pp. 79-98.

[Sari 89c] B. Sarikaya, *Conformance Testing: Architectures and Test Sequences*, Computer Networks and ISDN Systems 17 (1989), pp. 111-126.

[Sari 84] B. Sarikaya and G. v. Bochmann, *Synchronization and specification issues in protocol testing*, IEEE Trans. on Comm., COM-32, No.4 (April 1984), pp. 389-395.

[Sari 87] B. Sarikaya, G. v. Bochmann and E. Cerny, *A test design methodology for protocol testing*, IEEE Trans. on SE, (May 1987), pp. 518-531.

[Somm 89] I. Sommerville, *Software Engineering*, 3-rd ed., Addison-Wesley Publ. 1989.

[Suns 89] C. Sunshine, *Computer Network Architectures and Protocols*, (2-nd ed., C.A.Sunshine (ed.)), Plenum Press, 1989.