Validation of Distributed Algorithms and Protocols

Qiang Gao[1] Roland Groz[2] Gregor v. Bochmann[3] Joumana Dargham[3] E. Houssain Htite[4]

1. International Validation & Testing Corp.
P.O. Box, 32115, 1386 Richmond Road, Ottawa, Ontario, Canada, K2B 8B0

2. CNET LAA/EIA/EVP, Bat WB, Technopole Anticipa, 2 avenue Pierre Marzin F-22307 Lannion cedex France.

3. Université de Montréal, Dept. I.R.O, C.P. 6128, Succ Centre Ville, Montreal, Quebec, Canada, H3C, 3J7

4. Hewlett-Parckard Protocol Test Centre, 3333 Cavendish Place Suit 501, St-Laurent, Quebec, Canada, H4M 2X6

Abstract: *The use of formal description techniques allows the partial automation of the design, the validation, and the implementation of communication protocols and distributed algorithms. In this paper, we present a methodology for validation of distributed algorithms and protocols, and our experiences of using the Estelle [7] language, and a simulation and validation tool, called Veda [12], to simulate and validate complex distributed algorithms for the distributed implementation of multi-rendezvous. Some design errors in published distributed rendezvous algorithms were found. We obtain from these experiences heuristic guidelines for trouble shooting of distributed algorithms.*

Key Words: *Protocol validation, distributed algorithms, rendezvous*

1. Introduction

In distributed systems, processes proceed with different speeds and communicate with each other by message passing with unknown bound on message transmission delay. This asynchronous nature, together with concurrency and overlapping of different processing activities, makes coordination between processes difficult, and complicates the design and validation of distributed algorithms and protocols. The validation methods can be classified into logical proof, exhaustive reachability analysis, and simulation methods.

The logical proof method proceeds by proving assertions about the values of program variables. However, it is not possible to derive and prove the assertions in an algorithmic manner from the specification. This method relies on the human intuition to formulate critical assertions, and it is very difficult to apply this method to complex distributed algorithms and protocols.

The exhaustive methods consider all possible situations that may occur during the execution of distributed algorithms and protocols which are modeled by several interconnected processes, each can be modeled by a simple or extended finite state machine (FSM). The global state is determined by the states of each of the individual processes and the "messages" in transit between them. The method is aimed at deriving a reachability graph of all the global states that are reachable from the initial global state. The reachability graph is analyzed for deadlock, livelock, and unspecified receptions. This method is called reachability analysis. Another similar method is based on Petri net analysis. These two methods tend to lead to state space explosion when applied to complex distributed algorithms. To apply the above proof techniques, we have to simplify the description of the algorithm or the protocol. For instance, we could consider only a simple "phase" of the algorithm or protocol (and we may miss the problems related to inter-phase relations) or we could consider a reduced architecture (two or three stations).

The simulation method proceeds by executing the specification in a centralized

0-8186-7216-1/95 $04.00

way. It is aimed at inspecting as many reachable system states as possible by randomly-walking through the state space [23]. Simulation can also be guided with some heuristic guidelines. The real distributed environment is modeled and embedded in the simulation processes. It avoids the limitations of the above verification methods, at the expense of possibly missing some errors.

We present in this paper a methodology for validation of distributed algorithms and protocols, and our experiences and results of using the formal specification language Estelle [7], and a simulation tool called Veda [12], to simulate and validate some complex distributed rendezvous algorithms. The first algorithm we simulated was a virtual ring rendezvous algorithm [8]. It was designed in the context of the distributed implementation of LOTOS specifications [5]. Before we implemented the algorithm on a real network, we first performed its validation. In this process we found some design problems. Then we tried other algorithms [13, 20], and found similar problems. We summarize these experiences in heuristic guidelines for trouble shooting of distributed algorithms.

The rest of this paper is organized as follows. In Section 2, we present a validation methodology for distributed algorithms and protocols, tools needed to support it. Following this methodology, we have validated several distributed rendezvous algorithms as presented in Section 3. We summarize the experiences obtained from this validation process in heuristic guidelines for trouble shooting of distributed algorithms in Section 4. The paper ends with a conclusion.

2. Methodology and Tool Support

2.1 Methodology

Formal description techniques have been proposed for protocol engineering to support the different phases of the life cycle of protocol development. For the validation of distributed algorithms and protocols within an FDT-based environment [9, 10, 12, 16], we propose the following steps which should be performed in sequence.

(a) **Defining the Requirements:** This phase consists of designing a formal model of the service to be provided and of the properties to be satisfied by protocols and algorithms. In the case of protocols, this is called the service description. This task is difficult in general, because the assumptions of correctness are almost never explicitly stated in the informal design description. However, we do not need to write a full service description. We can restrict ourselves to the verification of selected properties of particular interest. The description of the service may be linked to the verification technique used (different techniques have different abilities of checking properties). For instance, Veda 2.0 [1] uses an observer language to describe the properties. This is powerful, but limited to safety properties, including bounded liveness.

(b) **Modeling**: Given an algorithm or a protocol described in natural language or in a loose pseudo-code fashion, we should first make a formal description of it in Estelle [7], or SDL [22]. Efforts are needed for the modeling of the architecture aspects of a distributed algorithm because the informal description of an algorithm usually makes very rough and naive assumptions in this area. This is very important because crucial choices in this area will influence greatly the ability to detect certain kinds of errors. Therefore, some information about the implementation environment may have to be added at this stage.

(c) **Debugging and Conformance check of the model:** This phase is debugging the FDT code. It consists of compile-time checks (syntax, cross-references etc.), and some preliminary simulations just to check that the formal model can be executed with sensible results, and it is a faithful representation of an algorithm or a protocol. At this point, we are not looking for a formal proof, which anyway cannot exist. But we can perform some basic tests by trying to reproduce, with the interactive facilities of the simulator. It is useful to know how much of the formal

specification has been covered in the process of running these few typical examples. Ideally, this should be 100%, however, this level may not always be easy to reach.

After steps (a), (b) and (c) have been performed, we are reasonably confident that the simulation will tell us something about the original informal description of design. We can now proceed to the real validation phases. Two levels of validation can be distinguished:

- A naive level consists of going on with simulation scenarios and checking the results (messages exchanged, states reached by stations) until it is very rare to find any error during the analysis of simulation runs. This is a simple prolongation of phase (c).

- A higher-level validation, as described under points (d) and (e) consists of performing an automated intensive verification based on a formal model captured in phase (a). This formal model will serve as an input to the automatic verification tool in order to replace the human analysis of traces and configurations by a much faster verification done by a program.

(d) **Random Simulation:** In this phase, verification proceeds through long random simulation runs. The ability to detect errors may be influenced by the ingenuity of the (human) validator to use varied simulation parameters (such as transmission delays, error rates, rates of requests, depending on the model for the environment).

(e) **Verification:** Verification is done by going through exhaustive analysis, reachability analysis or model-checking, for instance.

2.2 Tool Support

We will describe in this section the tools and their features needed to support each of the above phases. Although the methodology is independent of any choice of tool, the discussion is based on the experience we acquired with a series of tools for Estelle [7]: Veda 2.0 [1], Xesar [21], EWS [2]. Veda 2.0 has been used for most of these steps.

Phase (a) may depend on the choice of validation techniques used for phases (d) and (e). Different tools would accept different forms of requirement specifications: e.g., temporal logic formulas, FSM or EFSM specification for a service, behavior trees. In our case, things were made simpler by the fact that Veda 2.0 implements both a model-checking technique (e) and a random simulator (d), using a common description for the service in both cases. Service properties or requirements are described in the observer language, a modified syntax taken from Estelle [7]. The observer comes in during the course of execution to check the correctness during behavior explorations.

Phase (b) goes from informal to formal. Tool support may consist of syntactical help (graphic or syntax-directed editors, e.g., Veda 2.0 offers a graphical editor), and automated generation of systematic parts of a distributed model. For instance, most distributed algorithms make assumptions about the underlying communication networks: topology (ring structure, or various graphs), reliability (loss or corruption of messages), transmission parameters (order preserved, transmission delays), etc. A model of such a network may be built from standard building blocks. This idea has been implemented in e.g., the Oscar tool [18]. And also, for many tools, a closed environment is assumed. Unspecified environment modules can be derived automatically by using a tool like the Universal Test Drivers Generator [14]. We have not used any such generation tool for the experiment reported in this paper.

Phase (c) and (d) require a compiler and animation facilities. Apart from usual traces, Veda 2.0 offers "watch windows" that can be opened on instances of modules to trace their changes of states or the contents of their input queues. Other tools, like Grope [19] offer much more: it is possible to provide the user with graphic representation of the actual behavior including motion of messages along channel links between modules, and the change of states of the

FSM modules. Graphic facilities are very helpful to get an understanding of the system behavior "at a glance."

When a problem is identified, the tool should be able to record the scenario leading to this problem. This scenario can be readily analyzed by replaying it with more traces added.

When no new error is found, we come to the limits of the verification technique, and we can state that up to the limits of the verification tool and technique used, the system that we described formally is correct.

3. Simulation and Validation of Distributed Rendezvous Algorithms

3.1. The Distributed Rendezvous Problem and Algorithms

A distributed system is a system where information is distributed, there is no centralized controller to store the information and make decisions. Such a system consists of a set of communicating processes. Processes do not share variables. Processes communicate with each other by means of message passing. We assume that (1) processes are reliable; (2) channels connecting a pair of processes are reliable and FIFO, every message transmitted is eventually received; (3) each process has a distinct identifier. These assumptions will be considered when modeling the simulation architecture.

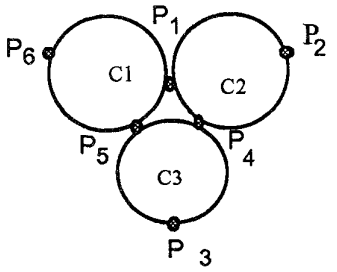
Rendezvous, also called an interaction, is an abstract mechanism of communication and synchronization among processes in a distributed system [11, 17, 20]. Multiple rendezvous is a natural extension of two-way rendezvous, where more than two processes are involved in a rendezvous. A rendezvous can only happen when all the processes involved in the rendezvous are ready, i.e., there is synchronization among all the processes belonging to the same rendezvous. A process can only participate in one rendezvous at a time, i.e., there is mutual exclusion between any two rendezvous that share common processes. Multiple rendezvous is sometimes referred to as the committee coordination problem [6].

The problem of distributed implementation of multiple rendezvous captures two fundamental issues in distributed computing: mutual exclusion and synchronization. Several algorithms have been designed so far [3, 4, 6, 8, 13, 20]. We are interested in the distributed implementation of these algorithms. Before doing the implementation, we tried to validate some of these algorithms, i.e., the virtual ring algorithm and its simplified verson [8], Kumar's algorithm [13], and Ramesh's algorithm [20].

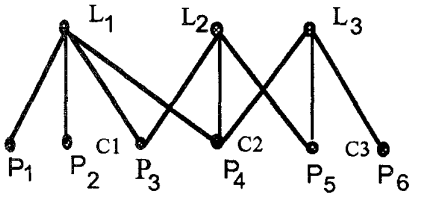
3.2. Simulation Architecture

The assumption made in Section 3 about a distributed system is considered in the Estelle specification. All the processes in the system are fully connected by Estelle FIFO queues, and addressed by their identifiers. The membership information of interactions is coded in the initialization part of the Estelle specification.

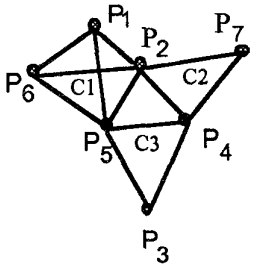
As mentioned in the Section 2.1, it is important to model the architecture of a distributed algorithm. We have considered three types of membership configurations: the virtual ring configuration, the umbrella configuration, and the lattice configuration, as shown in Fig.1. In the virtual ring configuration, all the processes belonging to the same interaction are connected on a ring; in the umbrella configuration, they are connected to their leader; in the lattice configuration, they are connected to each other. The virtual ring configuration is used for the simulation of the virtual ring algorithm [8] and Kumar's algorithm [13]; the umbrella configuration for the simplified algorithm of the virtual ring algorithm; the lattice configuration for Ramesh's algorithm [20]. For each type of configurations, there are many different actual configurations, for example, for the virtual ring configuration, there could be many rings with different numbers of processes.



The virtual ring configuration



The umbrella configuration



The lattice configuration

Fig.1. Different configurations

A given algorithm has to be able to work in all possible actual configurations of one of the above three types. However, the designer may consider only a few situations. After going through the simulation and validation without finding errors with the configuration shown in Fig.1, we have written a program to generate randomly the membership configuration for each of the above three types as follows. Recall that there is a set of "n" processes in the system. For each interaction, we choose at random an integer "k" ($0 < k <= n$) to be the number of processes involved in the interaction, and we choose at random "k" times from the set of "n" processes to select the members of the interaction.

The results described in the next session indicate that careful design of the configurations helps to detect errors.

For the validation of protocols and distributed algorithms, it is good to have a random delay box in modeling the communication channel since lot of design errors are due to race condition and relative delay.

3.3 Verification and Results

After going through long simulations without finding any error, we would like to perform an automatic intensive verification. The important property that a distributed rendezvous algorithm should have, is to satisfy mutual exclusion and synchronization.

We wrote a program in the Veda observer to check automatically that processes obey these conditions in the execution. The fairness property can be checked by looking at the traces. If rendezvous always happens at certain interactions, and never happens on some other interactions, we would suspect that the algorithm is unfair. Further analysis is necessary to come to a conclusion, as discussed later together with the example shown in Fig. 3.

Veda 2.0 provides reachability analysis. The state limit depends on the memory of the machine used, and is of the order of several millions.

Many errors have been found during simulation and validation activities. They fall into two large categories:

(1) Errors in the Estelle specification

The specification is an unfaithful representation of the design. Specification errors are most likely detected in the simulation through modeling, debugging and conformance checking. These are errors in Estelle coding, such as the following:
- Value out of range;
- Variables are not initialized, not updated properly, or not re-initialized after each session;

- The guard of a transition is not specified correctly to cover all the cases considered in the design.

(2) Design Errors

Design errors are much more serious. In most cases, they could be detected by running the simulation and analyzing simulation traces. They could be many types, such as the following:

- Internal logical consistency is not satisfied after some design modifications;
- Incomplete designs, unspecified receptions;
- Non-progress cycles;
- System deadlocks (i.e., circular waiting);
- Deadlock due to the delay in the FIFO queue;
- Errors due to collision or relative delay.

Examples of Errors Detected

Many errors were found in the validation process. Due to the space limitation, we can not list them all. Here we only give two examples. A design problem was found in the Ramesh's algorithm [20]. There are three processes, and two rendezvous between P1, P2 and P2, P3 as shown in Fig. 2. P3 sends *Req(P3, P2)* to capture P2 for rendezvous. P2 sends *Req(P2, P1)* to capture P1. However, P2 could not capture itself without capturing P1 first. So when P2 receives *Req(P3, P2)* from P3, P2 has to send YES to P3, and P2 will receive *Success* for rendezvous from P3. Then P2 goes to the initial state. The *YES* message sent by P1 to P2 in response to *Req(P2, P1)* will not be processed, therefore P1 will wait forever.

A possible way to fix this problem is to send a special *Cancel* message from P2 to P1, and P2 has to wait for this *Cancel* message to come back. So if there is a message *(YES)* sent out from P1 to P2, this special *Cancel* message will carry this information to P2, and P2 will wait until it receives this *(YES)* message before it goes to the initial state.

When we simulated Kumar's algorithm [13], we designed a combination of the virtual ring configuration as shown in the Fig.3, which permits us to

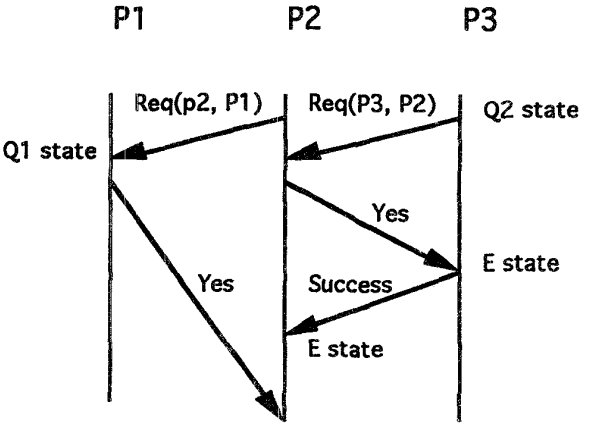


Fig. 2. Unprocessed message YES left in the channel

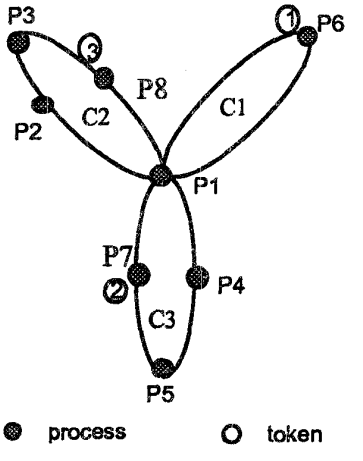


Fig. 3. A scenario of possible unfair rendezvous implementation

observe the fact that rendezvous always happens at interaction C1 in the simulation, and shows that the algorithm is unfair. In Kumar's algorithm, a token has to be circulated in the order of decreasing process identifier. The implementation could be such that Token 1 always arrives at process P1 earlier than other tokens, and captures P1 first. This is why rendezvous may always happen at interaction 1, and may never happen at the other two interactions. We conclude that this algorithm is unfair.

4. Hints for trouble shooting of distributed algorithms

The asynchronous nature of distributed systems makes the design and verification of distributed algorithms difficult. The errors detected by simulation and exhaustive validation are related to this nature. Based on our experiences with the validation of rendezvous algorithms, we present in the following several points that may be useful to detect errors in distributed algorithms in general.

(a) If an algorithm has to be able to work continuously, overlapping of different rounds (sessions) is likely to lead to problems related to variables, contents of queues, or token reallocations. These problems may cause total or partial system blocking.

(b) Some distributed algorithms use FIFO queues. In the specification, the size of the FIFO queues of these algorithms is infinite, but in an implementation it is finite. This may lead to message losses due to queue overflow.

(c) Random selection has been used in distributed algorithms for fair conflict resolutions [8], [15]. A practical problem may arise with the random number generator. When the random number generated is not very "random," it may take many random selections before a successful selection can be made, or it may even lead to livelock in extreme cases.

(d) Relative delay of messages could cause problems. One can always ask the question what will happen if a certain message is late. The sequence of messages is an important aspect to examine, the execution behavior can depend on it.

(e) In order to detect errors more effectively, simulation with different randomly generated architecture (different combinations of certain type of configurations) is recommended. Different architecture may have different aspects that are not covered in the original design.

5. Conclusion

In this paper, we present a methodology for the validation of distributed algorithms and protocols, and our experiences of using the Estelle language, and a simulation and validation tool, called Veda, to simulate and validate complex distributed algorithms for the distributed implementation of multi-rendezvous. Some design errors in published distributed rendezvous algorithms were found. We obtain from these experiences heuristic guidelines for trouble shooting of distributed algorithms. Although the experiences come directly from validation of distributed algorithms, it is applicable to protocols which can be considered as special cases of distributed algorithms.

The effectiveness of the random simulation technique is discussed in [23]. West claimed that a random exploration of the reachable-state is as effective as an attempt to perform an exhaustive state exploration. Our experiments also support this claim. We found that simulation is very effective to detect errors especial at the early stage of validation process. One major disadvantages of simulation is that there is no clear termination of the simulation process. Therefore, there is no way to determine when all the errors have found. In practice, we can terminate simulation after several days or a week without finding any errors. The application of the methodology proposed here gives us a high level of confidence in the quality of the formal design.

Acknowledgment: This work was partly supported by the Natural Sciences and Engineering Research Council of Canada, the Ministry of Education of Quebec and the Hewlett-Packard-NSERC-CITI Industrial Research Chair on Communication Protocols.

References:

[1] B. Algayres, et al., "VESAR: A Pragmatic Approach to Formal Specification and Verification," Computer Networks and ISDN Systems, Special Issue on Tools for FDTs, vol. 25, N0. 7, Feb. 1993.

[2] J. M. Ayache, et al , "EWS; An Integrated Workstation for Design and the Automatic

Generation of Distributed Software," FORTE' 88, The 2nd International Conference on Formal Description Techniques, pp. 85-89, K. J. Turner, editor, North-Holland, Amsterdam, 1988.

[3] R. Bagrodia, "A Distributed Algorithm to Implement N-Party Rendezvous," LNCS 287.

[4] R. Bagrodia, "Process Synchronization: Design and Performance Evaluation of Distributed Algorithms," IEEE Trans. on Software Engineering, vol. 15, No. 9, Sept. 1989.

[5] G. v. Bochmann, Q. Gao, C. Wu, "On the Distributed Implementation of LOTOS," The Second International Conference on Formal Description Techniques (FDTs) for Distributed Systems and Communications Protocols, December 5-8, 1989 Vancouver B. C., Canada.

[6] K. M. Chandy, J. Misra, A Fundamental of Parallel Program Design, Addison-Weseley, 1988.

[7] ISO/ IEC ISO 9074 (1989), "Information Processing Systems - Open System Interaction-Estelle - A Formal Description Technique Based on an Extended State Transition Model."

[8] Qiang. Gao, "On the Design, Validation and Implementation of Distributed Rendezvous Algorithms," Ph. D. Thesis, University of Montreal, Dept. IRO, July, 1995.

[9] R. Groz, C. Jard, C. Lassudrie, "Attacking a Complex Distributed Algorithm from Different Sides: an Experience with Complementary Validation Tools," Computer Network and ISDN Systems, No. 10, 1985, pp 245-257.

[10] G. J. Holtzmann, Design and Validation of Computer Protocols, Prentice Hall, 1991.

[11] C. A. R. Hoare, "Communication Sequential Processes," Communications of the ACM, August 1978, Vol. 21, No. 8.

[12] C. Jard, R. Groz, J. F. Monin, "Development of Veda: a prototyping tool for distributed algorithms," IEEE Trans. on Software Engineering, March 1988. pp. 339-352.

[13] D. Kumar, "An Implementation of N-Party Synchronization Using Tokens," Proceedings of the 10th International Conference on Distributed Computing Systems, Paris, France, 1990.

[14] E. Lallet, Ch. A. Lebrun, J. F. Martin, "Un outil de generation automatique de l'environnement d'execution de specifications Estelle," CFIP'91, Pau 17-19 Sept. 1991, Hermes publised.

[15] D. Lehmann, M. O. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem," ACM Conference Record of the Eighth Annual ACM Symposium on the Principles of Programming Language, Williamsburg, Virginia, Jan. 26-28, 1981.

[16] A. A. F. Loureiro et al, "FDT Tools for Protocol Development," In the Tutorial of FORTE'92, The Fifth International Conference on Formal Description Techniques, Lannion, France, Oct. 1992.

[17] ISO IS 8807: Information Processing Systems, Open Systems Interconnection. LOTOS: A Formal Description Technique for the Temporal ordering Observational Behavior, 1989.

[18] P. Maviel "Definition de la classe d'environnements reseaux pour la simulatioon d'algorithmes distribues," These de doctorat de l'universite Paris 6, Fev. 1987.

[19] D. New, "Protocol Visualization, " Ph.D. thesis, Univ. of Delaware, 1991.

[20] S. Ramesh, "A New and Efficient Implementation of Multiprocesses Synchronization," PARLE conf. Eindhoven, June 1987.

[21] J. K. Richier et al, Xesar: A Tool for Protocol Validation -User Manual. Laboratoire de Génie Informatique, Grenoble, France, 1.2 edition, September 1987.

[22] CCITT Specification and Description language SDL. Recommendation Z.100. CCITT Blue Book, 1988.

[23] Colin H. West "Protocol Validation by Random State Exploration," Protocol Specification, Testing and Verification, VI , B. Sarikaya and G. v. Bochmann (editors), Elsevier Science Publishers B. V. (North-Holland) IFIP, 1987.