A Service-Component Testing Method and a Suitable CORBA Architecture

Ana Cavalli, Bruno Defude, Christian Rinderknecht, Fatiha Zaïdi
Institut National des Télécommunications
9 rue Charles Fourier
F-91011 Évry Cedex
{Ana.Cavalli, Bruno.Defude, Christian.Rinderknecht, Fatiha.Zaidi}@int-evry.fr

Abstract

This paper presents a method for service-component testing and a suitable CORBA test architecture. This test environment allows the service validation from its components and is a close step towards the execution of the obtained tests on a CORBA environment. Two aspects are relevant: the test of components and the test architecture. The testing method for components is new: it is based on the generation of partial graphs and avoids the combinatorial explosion of number of states of the global system. The test architecture on a CORBA platform is also new, since there are few works on testing based on these environments. As an application we present a case study on a real conference call service.

Keywords *Components testing, test generation, test architecture, CORBA*

1 Introduction

The telecommunication operators are facing the dramatic growth of new services that utilize the telephonic network but also mix voice transmission with image and sound transmission. The inherent difficulty of this kind of services lies on their fast and reliable integration, in order to satisfy the users' requirements.

The Intelligent Network (IN) has been the first attempt to solve this sort of problems and, in the meanwhile, some others architectures have been proposed, like TINA, in order to cope efficiently with the design and the implementation of new services. These latter must meet the users' expectations and be definitely reliable. With this aim, after the design phase, the services should be validated and tested in order to guarantee that the proposed implementations comply with the requirements. Moreover, because of modularity and reusability constraints, these services are to be defined from base components which may further be combined and reused.

In this paper we present a method for service-component testing and a Corba test architecture. To illustrate the application of the method we present a case study: a conference call service. Our input is an SDL model of the service we want to test. Starting from this model, we generate a test sequence by using the component test algorithm, Hit-or-Jump. This sequence is built from the SDL model and is thus expressed in terms of SDL entities. The values associated to the exchanged signals have types represented within SDL. In order to step forward the test execution on a CORBA environment, this sequence must be transformed to be defined in terms of CORBA objects, and hence mapped to the IDL language. Last, an execution environment on a CORBA architecture is proposed.

The contribution of this work lies on the methods definition for the service-component testing and the test execution architecture on a CORBA environment. At the CFIP99 [7], we have presented yet a method for the embedded testing, but this method was based upon the generation of the whole accessibility graph of the system under test. The method presented here is new, it is based on the production of partial accessibility graphs and in this way avoids the combinatorial explosion of the state number of the global system.

We also present in this work the test execution architecture on a distributed CORBA environment. We give the principles allowing the test of a CORBA implementation by combining interactive and purely observational techniques (also called *passive testing*). This approach is innovative since, to our knowledge, there exists very few works on validation and test execution for CORBA environments.

This work was achieved in the CASTOR RNRT French project [2]. The specification of the conference call service was provided by France Telecom R&D.

This article is organized as follows. In section 2 we give a short overview of the algorithm used for the derivation of the service-components tests. Section 3 presents the case study, the conference call and its components. Moreover, in this section we present the results of the experiments we have made. Then section 4 presents a description of

the CORBA environment devoted to test execution, and explains the principles which guided its design. Last, section 5 gives the conclusions of this work.

2 A service-component testing method

2.1 Preliminaries

In this section we present briefly the method used for the generation of component tests. A more complete presentation can be found in [4]. A comparison with the existing methods [1, 12] is presented in [3]. Some tools devoted to the embedded testing (or *testing in context*) are presented in [11, 5].

The aim of the method is to test components in their context, i.e. the complement of the component in the system, because usually there is no direct access to the component. The system is defined in terms of *Communicating Extended Finite-State Machines* (CEFSMs), which are based upon *Extended Finite-State Machines* (EFSM) defined as follows.

Definition. An EFSM is a 5-tuple $M = (I, O, S, \vec{x}, T)$ where I, O, S, $\vec{x}$, and T are respectively finite sets of input symbols, output symbols, transitions. Each transition t in the set T is a 6-tuple $t = (s_t, q_t, a_t, o_t, P_t, A_t)$ where s_t, q_t, a_t, and o_t are respectively the start (current) state, the next state, one input and one output. $P_t(\vec{x})$ is a predicate on the current values of the variables and $A_t(\vec{x})$ defines an action on the values of the variables. Initially the machine is in a state $s^{(0)} \in S$ with the values of variables $\vec{x}^{(0)}$. Let the machine be in state s_t with the current values of the variables $\vec{x}$. When the input a_t occurs, if $\vec{x}$ is valid for P_t, i.e. $P_t(\vec{x}) = \mathsf{true}$, then the machine achieves the transition t, outputs o_t, changes the current values of the variables by means of action $\vec{x} := A_t(\vec{x})$, and finally reaches the state q_t.

The case study we present in this paper (conference call service) is specified with the SDL language [9]. The specification of behaviours in SDL is based upon CEFSMs. The actions associated to a transition may include the running of tasks, procedure calls, dynamic creation of processes (SDL has the concepts of *types* and *type instance*) and also the enabling and disabling of timers.

Until now the research on automatic test generation for components was based on the *exhaustive simulation* of these SDL specifications, i.e. the exploration of the whole state-space (or *accessibility graph*). The first drawback is that, when testing real-world service components, the number of states is huge; the second one is that redundant parts may be explored. To avoid these problems, we have designed and implemented a new algorithm, called *Hit-or-jump*, based on the generation of partial accessibility graphs.

2.2 Outline of the Hit-or-Jump algorithm

The algorithm presented here allows to cover all the interactions of the component in its context. The essence of our approach is as follows. At any moment we conduct a local search from the current state in a neighborhood of the accessibility graph. If an untested part of the component is found (a Hit), we keep it for the final test sequence, and then continue the search process from there. Otherwise, we move randomly to the frontier of the neighborhood searched (Jump), and resume the process from there. This procedure avoids the building of a whole system accessibility graph. Accordingly, the space required is determined by the user, e.g. a depth limit or a maximum number of states, and it is independent of the system under consideration. On the other hand, a random walk may get "trapped" at certain part of the component under test [12]. Our algorithm is designed to "jump" out of the "trap" and pursue the exploration further. to build at each step a partial accessibility graph to avoid the state-number explosion problem mentioned before. The algorithm finally produces a test sequence as a transition tour of the component in its context.

Initial condition. The environment machine C is in an initial state $s_C^{(0)}$, the component machine under test A is in an initial state $s_A^{(0)}$, and the system variables have initial values $\vec{x}^{(0)}$.
Termination. The algorithm terminates when all the transitions of A have been marked off.
Execution.

1. **HIT** From the current node $(s_C^{(k)}, s_A^{(k)}, \vec{x}^{(k)})$ conduct a search in $C \times A$ until (a) or (b) occurs:
   (a) Reach an edge which is associated with unmarked transitions of the component machine A: a Hit. Then:
      i. Include the path from the current node to the edge (inclusive) in the test sequence under construction;
      ii. Mark off the newly exercised transitions of A;
      iii. Arrive at a node $(s_C^{(k+1)}, s_A^{(k+1)}, \vec{x}^{(k+1)})$;
      iv. Erase the searched graph;
      v. Repeat from 1.
   (b) Reach a search depth or space limit without hitting any unmarked transition of A. Then move to 2.
2. **JUMP**
   (a) We have constructed a search tree, rooted at $(s_C^{(k)}, s_A^{(k)}, \vec{x}^{(k)})$.

(b) Examine all the leaf nodes of the tree, and select one uniformly at random.

(c) Include the path from the root to the selected leaf node in the test sequence.

(d) We arrive at the selected leaf node $(s_C^{(k+1)}, s_A^{(k+1)}, \vec{x}^{(k+1)})$: a Jump.

(e) Repeat from 1.

This algorithm avoids looping on the same state, as far as we choose uniformly and randomly in the spanning tree. Notice that Hit-or-Jump assumes that the specification is correct in the sense that it does not imply run-time deadlocks, non accessible states, etc.

3 A case study: A Conference Call Service

The conference call service is specified by means of the SDL language, as we mentioned in the previous section. This service allows a given set of users to establish a mutual communication. A subscriber, initiator of the communication, plays a special role: the moderator. He has in charge the conference-bridge reservation for a given hour and a given day. The conference cannot start before the arriving of the moderator.

This specification lies upon an Intelligent Network (IN) architecture and on the concept of reusable components, the *Service Independent Building Blocks* (SIBs). The Q.1201 recommendation defines the conceptual model of the IN. This model is divided in four plans, each plan being an abstract view of the system. The specification we considered stresses more the global functional plan, which is responsible for the service creation, and put the accent also on the service plan itself. We are here particularly interested in the user's view of the network and the services. This plan models the network as a unique entity, which executes the service.

In the system architecture described in figure 1 we find three main blocks: (1) the *Users* block, which consists of three processes: *users*, the *conference* and *term_manager* (2) the SSP block (3) the SCF block. The *users* process (the sole process that interacts with the environment) models the user's behaviour (to off-hook, to on-hook, to dial, to hear the busy-tone and the ring-back, to speak) which is finely defined. The *term_manager* process manages the SDL data of the block, and copes with other process creations. The SSP block is responsible for the switching function; it is divided into four processes: the *obcsm* process for handling the originating half calls, the *tbcsm* process for the terminating half calls, the CCF process and lastly the SSF process. The CCF process establishes, handles and releases the calls. The SSF makes the connection, by association with CCF, between a user and the Service Control Function (SCF).

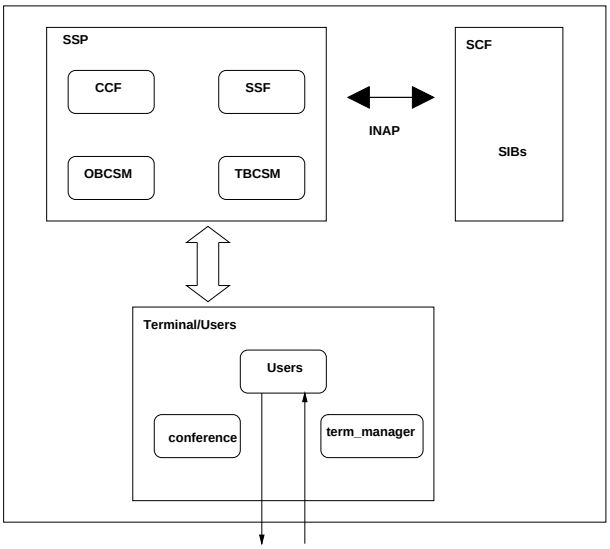

Figure 1. Architecture of the specification

The network communications, linked to the INAP protocol, are shortly considered here (only a few primitives are specified). In this specification the accent is put on the modular aspect of the service design, and on its SIB decomposition. A SIB is a piece of logic (or building block) which comprehends instantiation parameters as input, and returns a result.

3.1 Test generation for the service components

A SIB is a monolithic, reusable and standard component, used for building services. The SIBs take as input some static parameters called SSD (*Service Support Data*) and CID (*Call Instance Data*); it can be activated at an activation point POI (*Point of Initiation*) and may have one or more terminations POR (*Point of Return*).

The SIBs of the intelligent network are designed in an interprocedural way. Each component is mapped to a SDL procedure. Hence, in order to supply the service, a process is used, which has in charge the chaining of the building blocks (this chaining is sequential). This process plays the role of glue between the components.

The results obtained

The conference call service is supplied by a specific chaining of building blocks. The different components are generic, each of them models a function. With the aim of building up an executable service logic, they have to be instantiated by some parameters. These latter allow the definition of different behaviour of the same component, which are the following:

- DCL supplies the call triggering;
- EDA allows the call establishing;
- MVl_ann issues the play announcements to the users;
- MVL_exp has in charge the network play announcements;
- RSC_incr, RSC_decr and RSC_obs manage respectively the *conference bridge* resource incrementation, its decrementation and the monitoring.

Figure 2 presents some metrics on the service specification under study: number of lines of code, of blocks, of processes, etc. Figure 3 presents our results: the first column points the coverage rate of each component and the second one gives the length of the derived sequence. Note that the coverages are not always 100% because the components are generic and have been instanciated partially (for this application). The maximum execution time on a Sun Ultra 5 running SunOS 5.5.1 is 120 seconds.

Lines of SDL code	9.873
Blocks	3
Processes	13
Procedures	49
States	307
Signals	67
Macro definitions	29
Timers	1

Figure 2. The specification metrics

DCL	100%	32
EDA	65%	119
MVL_ann	50%	55
MVL_exp	80%	197
RSC_incr	100%	36
RSC_decr	100%	131
RSC_obs	80%	140

Figure 3. Test components results

The *conference bridge* module

We have also applied the algorithm to the *conference bridge* module. It seemed to us that it would be wiser to show the sequence computed for this module, as far as it allows a better understanding of a conference bridge — the sequence for the components being less clear. We got a sequence of length 247, expressed in the MSC [10] format and in the TTCN [8] notation.

Figure 4 is the simplified MSC for the test sequence obtained for the module representing the bridge-conference terminal. This sequence brings to light all the interactions of this module (it indeed covers all the transitions). We only show here the signal exchanges between the two peers, i.e. the switch (SSF) and the user, with which the bridge interacts. The environment here is schematized by the frame of the figure. The scenarios that can be found in the signal exchanges of the figure 4 are the following:

The user 2 asks for a connection to the conference call by dialing number 08.36.00.00.01. At the conference level, this is translated into the signal input setupreq. The conference module sends an offhook signal to the switch. This latter, after a chaining of exchanges with the PCS and a series of component activations, sends a setupresp signal which means that the communication can be established. First, the user receives a waiting message and, as soon as the communication is established, a welcome message. Once this latter is received, the user decides to on-hook: this sends a releaseind signal to the switch; this latter relays it to the conference module by means of the releasereq. This latter module acknowledges the disconnection request by the releaseind signal; then it receives a line_free message, which indicates that the line is free again.

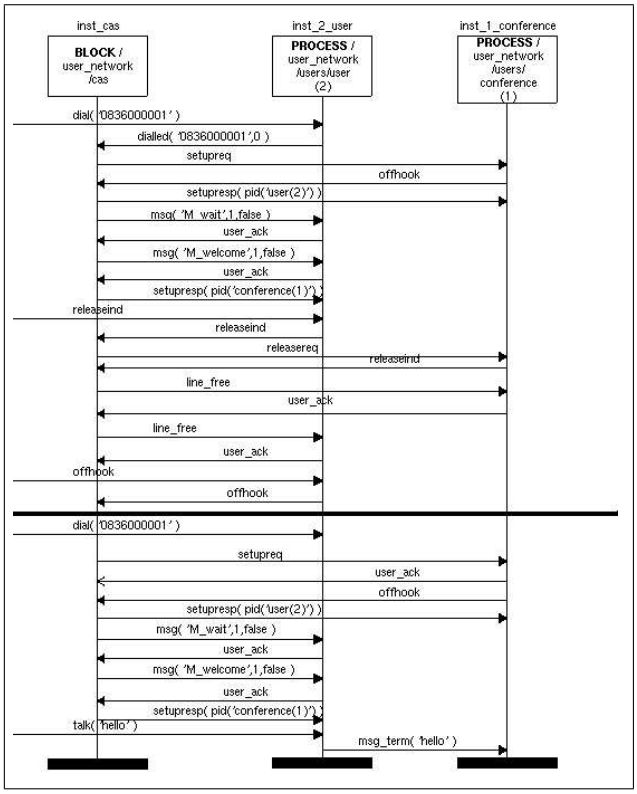

Figure 4. Simplified test sequence of the conference-bridge module

The second part of the figure brings to the fore the following scenario: the user 1 calls the bridge, he receives the waiting signal and then the conference is opened; the waiting subscribers receive the welcome message; consequently the communication is established, they can talk to each other: this exchange is revealed by the reception of a msg_term signal.

4 A suitable CORBA test architecture

CORBA is a standard promoted by the OMG consortium (*Object Management Group*) which is widely accepted in

the telecommunication industry, and the main actors of this sector have made large investments in this technology.

The CORBA standard defines a general model for distributed programming addressing heterogeneity at different levels (hardware, operating systems, networks and protocols, programming languages). CORBA can be viewed as a software infrastructure allowing the interconnection of heterogeneous pieces of software. CORBA is based on an object-oriented approach and on the client/server model.

The core level of a CORBA implementation is called an ORB (*Object Request Broker*). CORBA objects are described in a pivot language called IDL CORBA. This language allows communication between different programming languages (for instance C, C++, COBOL, Java, ST80, ...). An IDL CORBA definition describes the interface of a software and not a complete implementation. The IDL definitions are stored in an interface repository. This repository is mainly used to dynamically construct a method invocation (dynamic invocation interface).

The choice of CORBA as a test environment is motivated by the following reasons: (1) The telecommunication companies are moving to CORBA; (2) CORBA is a distributed environment garantying the transparency of resource location and communication, despite heterogeneity; (3) CORBA is well fit for a modular architecture like the SIB approach of the IN (see section 3).

4.1 Test architecture

Following classical architectures proposed for telecommunication protocols, such as those described in the ISO 9646 normalization, we propose a CORBA test architecture based on two types of components (testers and components under test) [13]. These components can be distributed on different workstations but we have just one tester.

The Hit-or-jump algorithm generates test sequences that are composed by active and passive segments. For the tester this implies the ability to stimulate the components under test and to compare the output to the attended result (the active part) but also the ability to observe the messages exchanged between the components under test (the passive part). Each message is composed by a name, a target object and some arguments values.

The tester has two parts (see figure 5). The first one processes CORBA messages corresponding to the active part of the test and the second one is an observer for the passive part of the test. From a CORBA point of view, the first part is a generic CORBA client object able to invoke any operation on any CORBA object. The CORBA naming service is used to resolve names used in the test sequences on CORBA object references (called IOR). The implementation of this first part uses intensively the dynamic invocation interface which allows to dynamically construct invocations using the interface repository. Testers must have a timer to detect livelocks and locks.

The second part of our tester is more difficult to address because we have to observe the invocations exchanged between the tested objects without modifying their source code. Moreover, the programming interface of ORBs do not offer any mechanism to observe the invocations. We have studied different solutions to this problem:

- *Extend an ORB with such capability.* This solution is more powerful because we can add any functionality we want but is also more difficult to realize. [6] is an example of this approach.
- *Use interceptors.* Interceptors are a new functionality in the CORBA 2.3 standard. Interceptors allow to add some code during an invocation between a client and a server. Interceptors seem to be a good solution for our problem, unfortunately modification of the source code is necessary;
- *Observe the invocations at the transport protocol layer.* CORBA 2 has standardized a communication protocol called GIOP (*Generic Inter Orb Protocol*) to allow different ORBs to cooperate in order to transport an invocation message. GIOP allows interoperability between different ORBs. GIOP can have different incarnations depending on the transport layer used. IIOP (Internet Inter Orb Protocol) is the main incarnation of GIOP. IIOP uses TCP as a transport layer. This solution has several advantages for us. First it is independent of the ORB and second it does not imply any modification of the source code. The main drawback is that IIOP is a transport layer that ignores the semantic of the transported data. Another drawback is that IIOP is used for remote invocations but not for local invocations (many ORBs do some optimizations in the case of local invocations).

We have chosen the third solution but we have to force the distribution of the components under test to ensure that all the invocations will be remote ones. The software architecture of our tester is described in figure 5.

With the distribution constraint, we have to define a two-level architecture for our observer. At the first stage we have one local observer per workstation and at the second stage a global observer which can be located at any site. Local observers and the global observer has to cooperate in order to construct the global history of invocations (this collaboration process is simplified because all the invocations are synchronous). We have chosen CORBA as the distributed infrastructure among the observers. The global observer interacts with the tester using a CORBA IDL interface (so they may be located on different sites). Interactions between

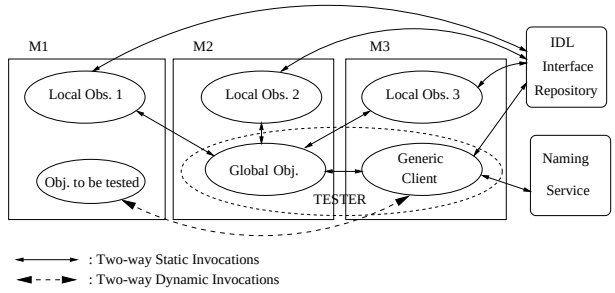

Figure 5. Tester architecture

the global observer and local observers also use a CORBA IDL interface (in both directions).

A local observer observes the local IIOP traffic and extracts useful information from the segments (name of the invoked operation, name of the target CORBA object and value of the arguments, if any). The problem here is that IIOP is a transport protocol and that we have to interpret the content of the segment in order to extract the information we need. This interpretation process uses the CORBA naming service to extract the information about the target object and the interface repository to extract the information about the invoked operation.

5 Conclusion

In this paper we have presented a global test architecture for distributed services including the generation of test sequences for service components. Our approach was validated by a case study: a France Telecom conference call service.

The full service was described using the SDL language, and the running of the Hit-or-Jump algorithm showed no deadlocks and produced the test sequences for all the components of the studied service.

For sake of simplicity, we have selected a component of the conference call service that is at the very heart of the service, the conference bridge, and which coordinates the other components and illustrates clearly what one imagines a conference call service is. The other components were generic components that can be present in other kind of telecommunication services, and for which we also generated the corresponding tests. We have produced the tests for the bridge component in its context, and we translated them in the TTCN and MSC formats.

We have also defined an architecture for the tester, which combines an active part (based on a stimulation of the implementation) and a passive one (based on the observation of the exchanges between the CORBA objects).

The results we got show that the use of formal methods considerably eases the task of the service designers and developers, and that they are usable for real services. Since the design phase to the implementation and test phases, we used formal description techniques (SDL, TTCN, MSC) and a formal test methodology. Moreover, we showed it is possible to test the service components in the context of the others (and not artificially in isolation). We think this is a notable step towards the validation and the design of reusable service-components.

References

[1] C. Bourhfir, R. Dssouli, E. Aboulhamid, and N. Rico. A guided incremental test case generation procedure for conformance testing for CEFSM specified protocols. In *IWTCS'98*, Tomsk, Russia, Aug. 1998.

[2] CASTOR. Conception d'un Atelier de création de Services pour les plateformes Tina, CORBA et RI. Partenaires Alcatel, France-Telecom-CNET, Verilog, INT. Projet précompétitif RNRT, `www-inf.int-evry.fr/~castor`.

[3] A. Cavalli, B. Defude, C. Rinderknecht, and F. Zaïdi. Test de composants de service et exécution de tests sur une plateforme CORBA. In P. S. Jean-Pierre Courtiat, Michel Diaz, editor, *CFIP'2000 Ingénierie des protocoles*, pages 363–378, Toulouse, France, Oct. 2000. Hermes.

[4] A. Cavalli, D. Lee, C. Rinderknecht, and F. Zaïdi. Hit-or-Jump: An Algorithm for Embedded Testing with Applications to IN Services. In *Proceedings of FORTE/PSTV'99*, Beijing, China, Oct. 1999.

[5] M. Clatin, R. Groz, M. Phalippou, and R. Thummel. Two approaches linking test generation with verification techniques. In A. Cavalli and S. Budkowski, editors, *Protocol Test Systems VII*. Chapman & Hall, 1996.

[6] C. Gransart, P. Merle, and J. Geib. Goodewatch: Supervision of CORBA applications. In *Proceedings of ECOOP'99 Workshop on Object-Orientation and Operating Systems*, Lisbonne, Portugal, June 1999.

[7] M. Ionescu and A. Cavalli. Test imbriqué du protocole MAP-GSM. In *Proceedings of CFIP'99*, Nancy, Apr. 1999.

[8] ISO. *Information Technology, Open Systems Interconnection, Conformance Testing Methodology and Framework, International Standard IS-9646*, 1991.

[9] ITU. *Recommendation Z.100 : CCITT Specification and Description Language (SDL)*, 1992.

[10] ITU-T, Geneva. *Recommendation Z. 120 Message Sequence Charts, (MSC)*, 1996.

[11] A. Kerbrat, T. Jeron, and R. Groz. Automated test generation from SDL specifications. In R. Dssouli, G. Bochman, and Y. Lahav, editors, *SDL'99*. Elsiever Science, 1999.

[12] D. Lee, K. Sabnani, D. Kristol, and S. Paul. Conformance Testing of Protocols Specified as Communicating Finite State Machines - A Guided Random Walk Based Approach. In *IEEE Transactions on Communications*, volume 44, No.5, May 1996.

[13] L. P. Lima and A. Cavalli. Exécution de tests de services sur une plate-forme distribuée. In *Proceedings NOTERE'97*, Pau, France, Nov. 1997.