Application of Two Test Generation Tools to an Industrial Case Study

Ana Cavalli[1], Stéphane Maag[1], Wissam Mallouli[1], Mikael Marche[2], and Yves-Marie Quemener[2]

[1] Institut National des Télécommunications GET-INT
Evry, France
{wissam.mallouli,ana.cavalli,stephane.maag}@int-evry.fr
[2] France Télécom R&D Division
Lannion, France
{mikael.marche,yvesmarie.quemener}@francetelecom.com

Abstract. Many tools for test generation already exist and are used in industry; others are under development or improvement to allow faster generation and more effective tests. Comparing testing tools permits to acquire in-depth knowledge of the characteristics of each tool and to discover its strong points and limitations. Thus, the analysis of different automatic test generation tools provides a precise idea on the appropriate tool to be used to attain the expected results. This paper describes the application of two test generation tools to an industrial case study: a reverse directory telephone service similar to deployed services of this category developed by France Telecom. The tools used, for the automatic test generation, are a commercial tool *TestComposer* and *SIRIUS*, a tool developed by INT team. France Telecom R&D division provided the test campaign designed manually by a France Telecom service expert used to define the test objectives. The goal of this paper is to present the experimental results of tools application, to compare their performances and analyze some issues related to test execution.

Keywords: *Case study, telephonic service, extended finite state machine, conformance testing, service testing, automatic test generation, formal specification, test generation tools.*

1 Introduction

In the telecommunication field, the complexity and the variety of the implemented systems, as well as the high degree of reliability required for their global functioning, justify the care provided to the design of the best possible tests. Moreover, it is significant to automate these steps with an aim of reducing the time and the development cost and especially of increasing the reliability of the offered products. Manual tests are expensive in terms of time, and are less reliable. Thus methods of automatic test generation are proposed. The tools for test generation are varied and closely related to the language (formal or not) in which the system, protocol or service specification is written. Nevertheless, even if automatic test generation [5,12] is seen as one of the most profitable application of formal methods, there are still very few commercial tools capable of automatically generating tests from a formal description and to execute them on the real system. The main reason for this is the difficulty of incorporating algorithms, to the methods, sufficiently powerful to make them scalable, i.e., applicable to real systems.

Our contribution: In this paper we present an industrial experience that consists of the application of two test generation tools to an industrial case study, a reverse directory service proposed by France Telecom, to perform test experiments and to compare their performances. One of the tools is a commercial one, *TestComposer*, while the other, *SIRIUS*, is a prototype developed in an academic environment. Both tools automatically generate test sequences from test objectives. These latter have been selected taking into account the testing campaign designed by a service expert of France Telecom. The comparison between these tools can help us to take a faster choice when modeling a system to validate it. It is indeed important for the validation phase to know as soon as possible which tool will be used. This comparison can be based on several criteria according to the technical resources we have. Among these criteria we can quote: test generation time, length of test sequences, complexity of the objectives, etc. Other aspects are also treated to clarify the strong points and the limits of each tool.

Different types of tests exist to ensure the reliability of a tested product. The tests presented in this paper are conformance tests [4,2], which consists of testing that an implementation conforms its specification. It must be noted that at the beginning standardization activities related to conformance concerned the strict field of communication protocols. Later many researchers proposed [8] the extension of the applicability of conformance testing methods and its techniques to cover all the fields where it is possible to specify a system interface or an operation in a formalism close to that used for protocols (automata, process algebra, etc.). As this was the case for services specification, it was proposed to apply these methods to services, in particular, to those provided to a network user. However, this extension has several consequences on the test generation: it is necessary to redefine the test methodology according to the characteristics of the services to be tested; it is necessary to extend the capacity of expression of the test description languages; it is necessary to define strategies for test selection adapted to the various types of systems. The test of a transport protocol OSI is not performed in the same way as a telephone service (i.e. we do not seek the same errors). In this paper, we propose to apply and compare methods used for protocol conformance testing to the test of services.

The rest of the paper is organized as follows. Section 2 gives an outline of the test methodology and test assumptions. In Section 3 the tools applied to the case study are presented. Section 4 presents the reverse directory service specification, the test objectives to be checked on the service and provides an outline of the test generation methods used by each tool. Section 5 presents the results and compares the performance of each tool. Finally, Section 6 concludes the paper.

2 Basics

2.1 Service definition

Before introducing our test methodology for services, it is necessary to provide the service definition, the concept of service in telecommunication being indeed very important. However, there is no unique definition of a service. In this paper, we used the definition that considers a service as a product offered by an operator to a customer in order to satisfy his needs.

2.2 Testing methodology

The service conformance test methodology is based on the one presented in the standard ISO9646 [2] and it is divided into four aspects:

1. The definition of test architecture: in general, services are hosted on servers where direct access is difficult for the tester. The service access is done by users and consequently, the points of control and observation (PCO) are placed on the user side in order to initialize the transactions, to inject the events (valid, inappropriate or invalid) and to recover the results. Points of observation (PO) can be attached to certain strategic points to observe that data are well transcribed during the transfer between the various entities (customer, server, Proxy, gateway...). Vocal services such as the reverse directory studied here add a difficulty to test execution: the interaction is made over the phone network, either by DTMF[3] (phone keys) or by voice which are difficult to handle automatically (see Section 5.6).
2. The description of service behavior using a formal specification language. The description shows the behavior of the service by taking in consideration the test architecture and includes the actions of the various entities that intervene in the correct operation of the service. The formal specification languages used for the description of the reverse directory service are SDL [6] and IF[4] .
3. The characterization of the tests to be carried out and the test generation. It consists of a selection of tests according to certain preset criteria and the tests generation following a given procedure. As the size of the specifications for the services is rather considerable, the use of traditional methods that produce a global reachability graph from which tests are generated is not possible. Therefore, it is necessary to use methods which are based on a partial generation of the accessibility graph by applying different algorithms. These methods are based on the definition of test objectives to guide the generation of the tests. In this work, test selection has been based on test objectives that represent relevant aspects of the reverse directory service behavior. Different test generation algorithms have been used by the tools *TestComposer* and *SIRIUS*. Many of these algorithms can be embedded in

[3] Dual Tone Multi-Frequency

[4] http://www-verimag.imag.fr/ãsync/IF/

several graph searching strategies like depth-first search (DFS), breath-first search (BFS) and BDFS which is a random combination of the two methods. In Section 5.2, we will explain the weight of these strategies in automatic test generation.

4. Test execution. The generated tests are executed according to the defined test architecture. Verdicts are established according to a conformance relation on each PO and PCO. All verdicts are directed to a central tester that deduces the final verdict. For this case study, test execution has been performed by France Telecom internal tools, which enable to send DTMF or vocal requests and to give a verdict from the vocal answers of the service.

After the description of the test methodology for services, we present in the following section the two tools used for the automatic test generation which are the commercial tool *TestComposer* and *SIRIUS*, a tool developed by INT team.

3 Tools presentation

3.1 TestComposer

TestComposer [11] is a test generator tool commercialized by Telelogic. This tool corresponds to a major revision of the ObjectGeode TTCgeN tool which is a test generator produced by Verilog. *TestComposer* is based on two complementary prototypes: Tveda and TGV.

Tveda is a tool developed by the research laboratory of France Telecom CNET (currently France Telecom R&D division). The main features of Tveda are the automatic generation of test purposes, its heuristic approach to the use of reachability analysis, and the wealth of pragmatic customizations that have been included to cater to the needs of many different applications within CNET. The first versions of Tveda only accepted the Estelle language as an input language. However, an extension that allows the use of SDL language as an input language was introduced in the version 3, which is integrated with TGV in ObjectGeode. And a new semantic approach based on the analysis of the reachability graph produced by the Véda simulator was also applied in this version. The Tveda tool allows the generation of test sequences by using the heuristics related to exploration of the reachability graph. Its main advantage is that it calculates automatically the test objectives, which is not the case with other existing approaches.

TGV (Test Generation using Verification Technology) is an automatic prototype for the generation of conformance tests developed by IRISA/INRIA and Verimag within the framework of the PAMPA[5] project. It allows generating test cases using an on-the-fly exploration of the state space based on test objectives. The on-the-fly exploration permits to carry out the reachability graph creation and the checking of a property at the same time. The advantage of that method is that it just keeps the required part of the graph necessary for checking the

[5] http://www.irisa.fr/pampa/

properties. Moreover, this technique can give a partial result even if it stops with a memory fault. Using on-the-fly algorithms makes the execution of the principal algorithm activates the execution of each intermediate stage. TGV needs the specification and the test objective as input and generates the test case in an Aldebaran automaton format [9]. This format can be then translated into the TTCN language [13,10].

3.2 SIRIUS

SIRIUS belongs to a set of powerful test tools (TestGen-SDL [3,7]) developed by the INT team. The major advantage of this tool is that it avoids the exhaustive generation of the reachability graph of a system specified by building only partial graphs, allowing solving combinative explosion constraints. *SIRIUS* is based on one of the flagship algorithms of TestGen-SDL, namely Hit-or-Jump [7]. In particular it allows the automatic generation of test sequences for the test in context. This is very significant mainly for the features integration in an operating platform. In addition to the generation of test sequences from test objectives, *SIRIUS* also offers many other functionalities. Indeed, it also allows:

- The automata minimization according to the bisimulation.
- The detection of sink nodes.
- The transformation of a possibly partial reachability graph in an Aldebaran automaton.
- The parsing and lexing in the purpose of debugging all the output files of ObjectGEODE.

4 The reverse directory case study

4.1 Vocal services presentation

The development of vocal services at France Telecom R&D division uses many distinct competencies. There are specialists in the techniques of voice recognition and text-to-speech systems which provide reusable components. The use and the adaptation of these components are themselves rather technical: it is necessary to build models of the sentences being able to be pronounced by the users and to check the good pronunciation by the text-to-speech system of precise information.

However, the main activity in the development of a vocal service consists in the conception of the dialogue between the human being and the service. An automaton should be defined, which, reacting to the input pronounced by the user as determined by the voice recognition, will provide output information via the text-to-speech system. This dialogue is often specified by a document in natural language.

In order to formalize the process of the development and the validation of vocal services, France Telecom R&D division developed a design tool based on

TauG2[6]. This tool makes it possible to design the dialogue of the service in the form of automata, and to produce a formal specification of it in IF. This specification can be used for doing simulations and automatic test generation.

At first approximation, the dialogue to be specified can be seen like a reactive system composed of a single automaton communicating by synchronous messages. This approximation corresponds to a rather simple problem looking to the state of the art. However, the following points complicate this first approximation.

– The vocal services often need to have access to data provided by the information system. That makes it necessary to model the vocal service in several distinct components.
– The assumption of synchronism and atomicity of the vocal messages is not entirely correct. It is necessary to be able to take into account phenomena such as the duration of pronunciation of the statements, the fact that the user can interrupt them or that this interruption can be blocked by the vocal service (barge-in can be allowed or not).

As a consequence, each dialogue is formalized by a distinct automaton, and the execution of the service corresponds to the parallel execution of each one, activated or not in a given state.

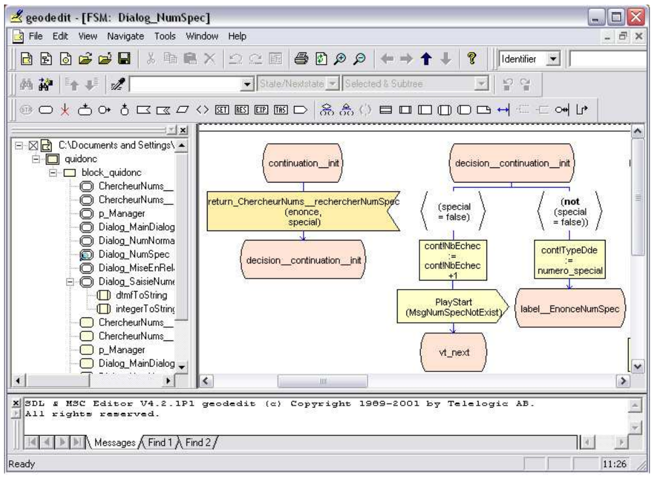
Each automaton composing the service can react to inputs of types DTMF or speech of the user. However, it is not important to model the voice recognition and its possible errors in the behavior of the vocal service. It is enough to consider an abstraction of this one starting from events in input of the service, and to consider an event corresponding to a failure of the voice recognition.

4.2 A reverse directory specification

France Telecom team has written a specification of a reverse directory vocal service similar to real services of this category deployed over the network, using the IF language. It is an intermediate language developed by VERIMAG[7] to describe timed communicating systems. Many tools were also developed allowing the translation from languages like SDL, LOTOS, UML to IF and the generation of code or labeled transition systems in order to formally verify some properties.

INT team has used the following tools: *TestComposer* from ObjectGeode and *SIRIUS* developed at the INT. These two tools are based on a well adapted language for the formal specification of interactive systems: SDL (Specification Language Description). As the original specification was written in IF, we carried out a translation from IF towards SDL. For that aim, we used the respective relations between the semantics and syntax of SDL and IF. Moreover, ObjectGEODE tool gives the possibility to draw the specification using a graphical mode which facilitates the comprehension of the service. Figure 1 produced by ObjectGEODE tool presents two states of Dialog_NumSpec process of the reverse directory specification.

[6] http://www.telelogic.com/
[7] http://www-verimag.imag.fr/



```
SDL & MSC Editor V4.2.1P1 geodedit (c) Copyright 1989-2001 by Telelogic AB.
All rights reserved.
```

Fig. 1. Reverse directory specification using ObjectGEODE tool

After the redaction of the specification, the ObjectGeode tool makes syntactic and semantic verifications of the specification. The syntactic analysis ensures that the specification complies with the syntactic rules of SDL in order to have a correct specification, whereas the semantic verification ensures the consistency of the specification. This step is carried out not only by this static analysis but also by an automatic exhaustive exploration of the specification. This is performed by testing all possible ways of system execution, with a certain number of rules and the cases of violations such as deadlocks, loops etc. During the verification, the main analyzed properties are:

- Safety (absence of deadlock, unspecified reception, blocking cycles, etc). Deadlock takes place when a state of the system, reachable from the initial state can not trigger a transition anymore.
- Promptness (livelock), indeed, a state is known as alive if it can be reached starting from all the states of the global system.

4.3 Determination of test objectives

France Telecom provided the test campaign designed manually by a France Telecom service expert. This document was the first reference we used to define our test objectives. In this document, the validation of the reverse directory service was described and organized in several phases related on one hand to the service conformance, and on the other hand to interoperability with the components used for voice synthesis and recognition. 130 test cases that were informally described cover the whole interactions of the service with the user, which includes: - Unitary checking of each service's functionality; - Checking of system behaviors related to eventual user inactivity; - And checking of behaviors activated at the time of voice recognition failures. The unit checking makes it possible to validate that all the service functionalities conform to the specification. The other verifications allow the validation of the service ergonomics in the case of defective use, related either to the user or to the service.

Using this test campaign, we have chosen to test the most relevant service reactions and functionalities. Therefore, 17 test objectives were obtained which represent more than 95% of the specification states and transitions. These test objectives can be classified in four categories: - Errors when dialing a number: we suppose that the user makes mistakes while trying to dial the telephone number of whom he wants to know the name. - Functionalities offered after dialing a normal number: Three functionalities are offered to the user. Only some reactions of the system are described. - Functionalities offered after dialing a special number: Some reactions of the system are described. - Errors due to user inactivity: we suppose that the user stops typing on the keys for a significant period. Messages for assistance will try to guide him for a good use of the service.

The three first categories describe functional tests of the service. It represents the details of what have been already called in France Telecom test campaign "the unitary checking if each service's functionality". The last category allows exactly, like in the France Telecom document, the system behaviors verification related to eventual user inactivity. We did not take into account in our test objectives the system behaviors related to voice recognition failures because this service feature was not specified.

Following are the test objectives we have defined in this case study:

N	Objectives to be tested: To test the correct reaction of the system, following this case:
1	The user does not press on any key after dialing the 3288. She then receives messages for assistance before receiving three messages to ask her to hang up followed by the stop of the service and the disconnection of the user.
2	After dialing the 3288, the user presses on the star key as it is required by the welcome message. Then, if she does not press on any key, an assistance message is transmitted to her explaining the functionalities of the service.
3	Error when dialing a number: the user presses on # key without having entered before a number. Sharp is a key which indicates the end of dialing.
4	Error when dialing a number: the user gives a telephone number that doesn't start with 0.
5	Error when dialing a number: the user gives a phone number which contains less than 10 digits.
6	Error when dialing a number: the user gives a telephone number which contains more than 10 digits.
7	Error when dialing a number: After having begun dialing, the user cancels his input number and dials * to give another number.
8	The user gives a good normal number (a normal number is a number starting with 0 and containing 10 digits). This number does not exist in the telephone directory of the reverse directory, an informative message is then transmitted to the user.

N	Objectives to be tested: To test the correct reaction of the system, following this case:
9	The user gives a normal number, receives the name of the corresponding person and remains inactive; this stops the service.
10	The user gives a normal number, listens to the name of the corresponding person and then dials 1 to listen to the spelling of this name.
11	The user gives a normal number, receives the name of the corresponding person and presses # that stops the service.
12	The user gives a normal number, listens to the name of the corresponding person and then presses 3 to contact this person. The telephone of the called rings without answer.
13	The user gives a normal number, listens to the name of the corresponding person and presses 3 to contact this person. The connection succeeds.
14	The user gives a normal number, listens to the name of the corresponding person and then dials 2 to listen to the address of the correspondent.
15	The user gives a good special number. This number does not exist in the telephone directory of the reverse directory; an informative message is then transmitted to the user.
16	The user gives a good special number that exists in the telephone directory. Information concerning this special number is transmitted to the user.
17	The user gives a good special number that exists in the telephone directory. Information concerning this special number is transmitted to the user. Some variables' values are transmitted to the system to allow generation of statistics.

Fig. 2. Test Objectives for the case study

4.4 Generation with TestComposer

According to [11], *TestComposer* is based on two complementary prototypes Tveda, which makes it possible to calculate automatically the test objectives and TGV, which allows, starting from calculated objectives or specific objectives, to calculate the corresponding tests. In our case, we give specific test objectives using a "test condition" formalism. The defined or calculated test objectives may be applied on a global SDL specification representing the system under test (the case of the reverse directory). We are also able to apply them on a part of this specification to test a component in a context. From the test objectives, test cases are produced and stored in a database. Then, the test sequence built from the obtained test cases is written in a TTCN file. Experimental results are presented in the Section 5.

4.5 Generation with SIRIUS

As mentioned above, *SIRIUS* is based on Hit-or-Jump, an algorithm especially used for components testing to perform test sequences generation through the specification. This research is guided by objectives which are illustrated by predicates on transitions. The research in the partial reachability graphs is performed in depth, width or both at the same time, and is restricted by a limited depth. In order to initialize the generation of test sequences, several parameters are necessary. Four main files must be developed. The first is the specification of the service (component to be tested), the second allows to initialize certain variables if necessary, the third one mentions the stop conditions (i.e. test objectives) and finally the last one allows the expert to guide the system at the beginning of the simulation, this file is called preamble. This latter is very important; it allows reducing in a consequent way the length of the sequence and the duration of its generation. As for *TestComposer*, we present our analysis and experimental results from the *SIRIUS* use in the following section.

5 Experimental results

This section presents the results of the experimentation performed by both tools. Different aspects related to test coverage and execution of the tests are also discussed.

5.1 The proposed tools are scalable

The first objective of this paper is the application of two test generation tools to an industrial case study. The reverse directory service is an example of significant size as illustrated by the following table that provides some metrics of the specification, both in IF and SDL. These numbers are representative of real examples that have been treated at INT and France Telecom (MAP-GSM protocol, TCP/IP, SSCOP). It must be noted that the specification in SDL of the service has 4603 lines and the specification in IF has 1213 lines. The difference is due to the fact that SDL specification contains many lines of comments describing graphical aspects of the service specification (cf Figure 1). The SDL specification has been used by both tools to generate the tests.

	Blocks	Processes	States	Transitions	Signals	Channels	Number of lines
IF	—	8	77	160	31	15	1213
SDL	1	8	77	160	31	15	4603

Fig. 3. Metrics of IF and SDL reverse directory service specification

5.2 Test objectives generation

The results of test generation for the seventeen objectives are recapitulated in the following table. The tests are performed in the order in Section 4.3. For each test, the length of the sequence (number of transitions), the test generation time and the length of the preamble to be provided to guide the simulator to lead the test objectives are given.

	TestComposer			SIRIUS		
	Nb of transitions	Generation duration	Preamble length	Nb of transitions	Generation duration	Preamble lenght
Test 1	8	0mn 0s	0	7	0mn 0s	0
Test 2	9	0mn 0s	0	5	0mn 0s	0
Test 3	22	0mn 2s	0	7	0mn 12s	0
Test 4	64	0mn 33s	35	15	3mn 36s	35
Test 5	11	0mn 0s	0	5	0mn 0s	0
Test 6	13	0mn 0s	0	5	0mn 1s	0
Test 7	13	0mn 0s	0	5	0mn 0s	0
Test 8	87	0mn 18s	50	28	1mn 40s	50
Test 9	83	0mn 34s	45	24	3mn 47s	45
Test 10	84	3mn 47s	45	25	2mn 33s	50
Test 11	82	0mn 24s	45	25	1mn 52s	45
Test 12	85	0mn 9s	50	24	0mn 36s	50
Test 13	89	0mn 10s	50	25	0mn 51s	50
Test 14	83	0mn 38s	45	24	3mn 59s	45
Test 15	47	0mn 32s	8	14	3mn 34s	8
Test 16	48	4mn 34s	8	14	3mn 36s	13
Test 17	49	0mn 12s	14	16	1mn 02s	14

Fig. 4. Some automatic test generation results

The results established in this table are obtained after a BFS (breath-first search) exploration of the reachability graph. This choice is due to the specificity of the service which has to take into account after each transition all the possible inputs injected by the user, to analyze them and generate the right output. The test objectives we defined are reachable via quite short sequences (almost 50 transitions) and do not need a DFS or BDFS exploration that tries to search in depth of the reachability graph.

5.3 Performance analysis and discussions

According to the Figure 4, we can elaborate on performancewise comparison between *TestComposer* and *SIRIUS*. First, we can easily notice that the test generation duration using *SIRIUS* is larger than using *TestComposer*. This fact constitutes a positive point for *TestComposer*. But if we refer to another criterion which is the length of test sequences, *SIRIUS* becomes more efficient. Actually, the length of tests sequences constitutes a very significant comparative data since test execution duration depends on it, and mainly when this execution is manual. This is often the case for vocal services where the test automatization is difficult because of the peculiarities of DTMF or vocal interaction (see Section 5.6). This length is on average three times shorter for *SIRIUS*. Indeed, *SIRIUS* has an advantage compared to *TestComposer* since it allows the automatic elimination of silent transitions. With this operation, the length of the test sequence becomes shorter and more comprehensible for the person carrying out the test. This fact constitutes one of the strongest points of *SIRIUS*. The Figure 5 recapitulate the length of the test sequences for the seventeen predefined objectives.

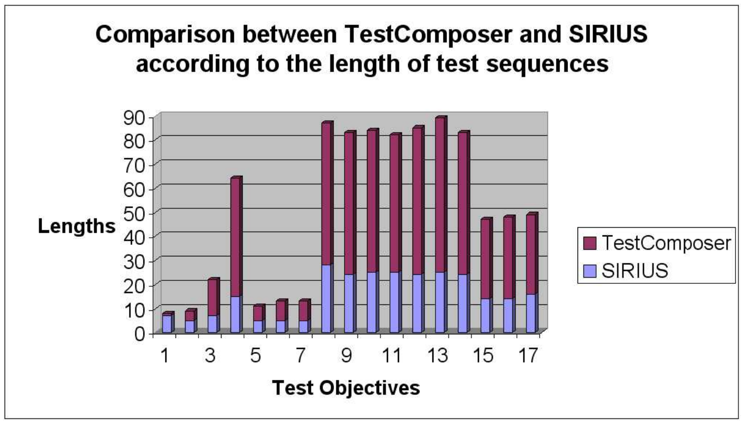


Fig. 5. Graphics comparison according to the length of test sequences

The automatic test generation is not always possible especially if the system to be tested presents a complex reachability graph. It is sometimes necessary to guide the simulator by providing a preamble. This preamble comprises the first transitions to be followed in order to begin the research of the test objective. The shortness of the preamble constitutes a strong point of the used tool. In our application, *TestComposer* shows superiority compared to *SIRIUS*. Indeed, we may in certain cases (test 10 for example) automatically generate tests using *TestComposer* giving a 45 transition length preamble, whereas a generation with *SIRIUS* requires 5 more transitions.

Another significant comparison criterion between various tools of simulation and test generation is the memory use. With ObjectGEODE, the memory consumption for the reverse directory telephone service is about 35.0 MB; this consumption is 10 times lower with *SIRIUS* (3.1 MB). This is easy to understand, first because *SIRIUS* only builds a part of the reachability graph for the test generation and second because ObjectGEODE has a graphic interface and several other functionalities which is not the case of *SIRIUS*.

	TestComposer	SIRIUS
Generation duration	+ Fast generation	-
Length of sequence	-	+ Sequence 3 times shorter
Memory used	-	+ Space 10 times smaller
Length of Preamble	+ Shorter preamble	-

Fig. 6. Summary table

According to this summary table, the user can make his choice based on his personal criteria and the characteristics of his application.

5.4 Test coverage can be evaluated and is reasonable

The advantage of automatic test generation compared to the manual procedures is that test selection is made by an algorithm and, therefore, it is possible to have a precise evaluation of the coverage achieved. From this point of view, both tools *TestComposer* and *SIRIUS* produce similar results that correspond to the notion of coverage normally accepted. Both tools have been based on the know-how of experts: test objectives were based in the test campaign proposed by the service expert of France Telecom. By placing this know-how in an algorithmic form we realize that their strategy for selecting tests correspond to branch coverage of a subset carefully chosen in the specification. If the test passes successfully, the implementation conforms the specification; assuming a uniformity hypothesis (passing once through each branch is representative of the whole protocol).

Furthermore, it should be noted that this test objective study is appropriate, since it corresponds to a reasonable number of tests. 17 test objectives have been selected, which cover more than 95% of the specification states and transitions.

5.5 Tests are really usable

The produced tests are really usable, since they have the same format as (and are of comparable size to) test suites developed manually. Both tools, *TestComposer* and *SIRIUS*, generated tests that are composed of a preamble (shortest path between the initial state and the starting state for the transition to be tested), followed by the transition being tested. The structure and length of the tests produced by both tools correspond therefore to the usual standards. As a final remark, it is interesting to note that tests can be produced in TTCN [13] and MSC [1] notation facilitating the portability of the tests.

5.6 Automatic Test Execution

If the phase of automatic tests generation is a problem that can be adequately treated by test generation tools, because of the nature of reactive system of the specification, the phase of automatic tests execution encounters many difficulties.

The first difficulty, central with the problem, is related to the heterogeneity of the platforms executing the services. This heterogeneity, characterized by the lack of a standardized API to access to the platform, implies that a generic solution for the tests automation must be based on an emulation of the interaction between the human and the service.

This consists in emulating the input of tests starting from text to speech or recorded sound files, and to observe/control the outputs by using techniques of voice recognition. However, even if it is technically rather easy to associate each tests input with sound files, the recognition of the output is not good enough.

Concretely, France Telecom R&D division developed a test automaton based on the voice recognition. This automaton ensures the validity of the PASS and FAIL verdicts, but produced many false positive characterized by verdict UNCONCLUSIVE. Typically, a verdict is FAIL if the service produced an output instead of a silence, and UNCONCLUSIVE if the output is well envisaged, but not recognized by the voice recognition.

Taking into account these difficulties, many tests must be carried out manually at the time of the validation of the service. However, the manual execution of tests is expensive in resources, because it implies the mobilization of people and time. In order to optimize the resources as well as possible, a great interest is related in advance to the quality of the generated tests. Those tests must cover in a minimum of occurrences the functionalities of the service, and must especially comprise a minimum preamble to validate the test objective.

In this context, the quality of the preambles to the generated tests is a discriminating element at the time of the selection of a tool for generating tests.

6 Conclusion

In this paper, we attempt to compare the performances of two test generation tools, *TestComposer* and *SIRIUS*, by applying them to a real case study provided by France Telecom, a reverse directory telephone service . These tools perform automatic test generation based on test objectives. The test objectives used for the experiments were provided by the France Telecom test plan. Results of the experimentation show that these test objectives covered quasi completely the tests provided by the test plan, showing the interest of the use of formal testing methods. Performance analysis shows that even if test generation time could be no so important for industrials (they are looking for pertinent and correct test sequences), this criteria could be important from an academic point of view in order to compare the algorithm performances. In addition, experiment show that the length of test sequences is an important criterion to evaluate the test sequences. In particular, for this case study this element was imperative because the tests were executed manually by France Telecom. Tests were executed manually because the reverse directory is a vocal service and it was very difficult to automate test execution and mainly to automate voice recognition that remains non deterministic. Finally, tools comparison allows us to improve our understanding of the strong points and limitations of each tool.

It must be mentioned also, that this study could be extended to other test generation tools. France Telecom's authors are ready to give access to the specification and the original test plan. This could provide a realistic problem to designers of automatic test generation tools.

References

1. *ITU-T Rec. Z. 120 Message Sequence Charts, (MSC).* Geneva, 1996.
2. ISO/IEC 9646-1. *Information Technology - Open Systems Interconnection - Conformance testing methodology and framework Part 1: General Concepts.*
3. R. Anido and al. Engendrer des tests pour un vrai protocole grâce à des techniques éprouvées de vérifications. In Proceeding of CFIP96/Cinquième Colloque Francophone sur l Ingénierie des Protocoles, editor, *In ENSIAS*, pages 499–513, Rabat, Maroc, octobre 1996.
4. A.V.Aho, A.T.Dahbura, D. Lee, and M.U.Uyar. An optimization technique for protocol conformance test generation based on uio sequences and rural chinese postman tours. In *IEEE transactions on Communications, 39(3), pages 1604-1615.*
5. C. Bourhfir, R. Dssouli, E. Aboulhamid, and N. Rico. Automatic executable test case generation for EFSM specified protocols. In Chapman & Hall, editor, *IWTCS97*, pages 75–90, 1997.
6. A. Cavalli and D. Hogrefe. Testing and validation of SDL systems : Tutorial. In *SDL'95 forum*, 1995.
7. Ana Cavalli, David Lee, Christian Rinderknecht, and Fatiha Zaïdi. Hit-or-Jump: An Algorithm for Embedded Testing with Applications to IN Services. In Jianping Wu, Samuel T. Chanson, and Qiang Gao, editors, *Formal Methods for Protocol Engineering And Distributed Systems*, pages 41–56, Beijing, China, october 1999.
8. 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.
9. J.-C. Fernandez, H. Garavel, A. Kerbat, L. Mounier R. Mateescu, and M. Sighireanu. Cadp : A Protocol Validation and Verification Toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *The 8th Conference on Computer-Aided Verification, CAV'96*, New Jersey, USA, August 1996. Springer Verlag.
10. G. Rethy I. Schieferdecker A. Wiles J. Grabowski, D. Hogrefe and Colin Willcock. An introduction to the testing and test control notation (ttcn-3). In *Computer Networks 42(3)*, pages 375–403, 2003.
11. A. Kerbrat, T. Jeron, and R. Groz. Automated test generation from SDL specifications. In R. Dssouli, G.V. Bochman, and Y. Lahav, editors, *SDL'99.* Elsiever Science, 1999.
12. J. Tretmans and A. Belinfante. Automatic testing with formal methods. In *Proceedings of the 7th European International Conference on Software Testing, EuroSTAR'99*, November 1999.
13. ETSI. TTCN-3. *TTCN-3 – Core Language.*