Two Complementary Tools for the Formal Testing of Distributed Systems with Time Constraints *

Ana Rosa Cavalli[1], Edgardo Montes de Oca[2], Wissam Mallouli[1,2] and Mounir Lallali[1]
[1] Institut Telecom / Telecom SudParis, CNRS / SAMOVAR, France
{ana.cavalli, wissam.mallouli, mounir.lallali}@it-sudparis.eu
[2] Montimage Research Labs, 39 rue Bobillot, 75013, Paris, France
edgardo.montesdeoca@montimage.com

Abstract

The complexity and the variety of the deployed time-dependent 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. In this paper, we present two different tools to test systems with time constraints. The first one allows to automatically generate test cases based on model-based active testing techniques. Whereas the second tool is based on passive testing approach to check that the collected system traces respect a set of formal properties called Invariants.

Keywords: Formal Testing, Active and Passive Testing, Real-Time Systems.

1 Introduction

Due to the increasing complexity of distributed and real-time systems, formal testing is becoming a critical activity to guarantee the respect of functional and security requirements by such systems. This formal testing can be divided into two categories: (i) the active testing [6, 8] which validates a system implementation by applying a set of test cases and analyzing its reaction and (ii) the passive testing [3, 9] (or monitoring) that consists in observing, during the execution, whether the system behavior conforms to a set of formal properties.

In order to ease both active and passive testing processes, we have developed two software tools called *TestGen-IF* and *TestInv* to automate all the phases of our active and passive testing approaches. In particular the tools include, for active testing, algorithms for automated test generation from a formal specification (with time constraints) of the system under test (SUT) and, for passive testing, algorithms to check whether the collected system trace respects a set of properties called invariants. These invariants may contain time requirements.

Both tools have been integrated into a testing platform that offers several capabilities and allows real experimentation for validating protocols, services and applications in a distributed soft real-time environment. By using different techniques, the aim is to check whether some protocols exchanges are correct, as well as to decide if different entities can cooperate. In addition, it can also be used to decide whether a service is properly delivered and deployed.

TestGen-IF and *TestInv* are respectively presented in sections 2 and 3. Some features of these tools applications are briefly provided in section 4.

2 Active Testing tool

TestGen-IF tool is based on active testing techniques that are used if the interaction with the entity under test is feasible. In this case, the tester is a program that communicates with the system to apply a set of a test scenarios and study its reaction according to its formal specification. The automation of this kind of tests needs an exhaustive test suites generation including all possible scenarios. To reach this aim, we rely on an automated test generation algorithm based a formal specification of the system described in IF (Intermediate Format) formal language [4, 5] implemented in *TestGen-IF*.

2.1 TestGen-IF Background

2.1.1 IF Language

A real-time system described using IF language is composed of active processes running in parallel and interacting

*This work has been supported in part by POLITESS French project (http://www.rnrt-politess.info/) and SHIELDS European project (http://www.shields-project.eu/).

asynchronously through shared variables and message exchanges via communication channels or by direct addressing. Each IF process is described as a timed automaton extended with discrete data variables, various communication primitives, dynamic process creation and destruction, and urgency attributes on transitions (used to control the time progress).

2.1.2 Hit-or-Jump exploration strategy

TestGen-IF implements an automated test generation algorithm based on a Hit-or-Jump exploration strategy [7] and a set of timed test purposes. This algorithm efficiently constructs tests sequences with high fault coverage, avoiding the state explosion and deadlock problems encountered respectively in exhaustive or exclusively random searches. It allows to produce a partial accessibility graph of the SUT specification in conjunction with the IF simulator [5]. At any moment it conducts a local search from the current state in a neighborhood of the reachability graph. If a state is reached and one or more test purposes are satisfied (a Hit), the set of test purposes is updated and a new partial search is conducted from this state. Otherwise, a partial search is performed from a random graph leaf (a Jump).

2.1.3 IF Simulator

The *TestGen-IF* tool is based on the IF-2.0 simulator [5] that allows to construct the accessibility graph from an IF specification. This simulator is developed by a research team at Verimag [1], for modeling and simulating asynchronous timed systems such as telecommunication protocols or distributed applications. It uses the IF-2.0 simulator libraries which provides some functionalities for on-the-fly state-space traversal (state representation and successors computation). The tool is implemented in the same implementation language as the IF-2.0 simulator, i.e. C++ language.

2.2 TestGen-IF Architecture

The active testing tool is illustrated by the Figure 1. The Properties (Test Purposes) box represents the timed system objectives to be tested. For example, "*action = input* sig *in state =* s *when clock* $c = d$" describes an action constraint that expresses that the signal sig can be received in the state s when the valuation of clock c is d.

The Automatic Test Generation box represents the test generation procedure combined with the IF specification (.if file) and the test purposes (.tp file). For the test generation, it is up to the user to choose the exploration strategy of the generated partial graph he wants to perform during the test generation: in depth or in breath exploration [8]. During this generation, when a test purpose is satisfied, a message

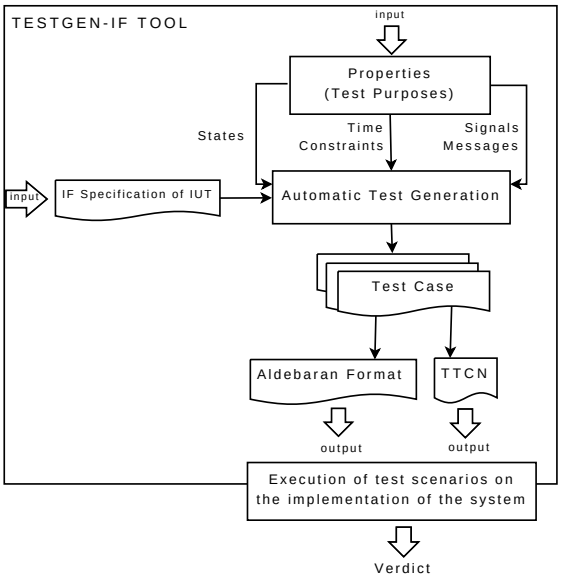

Figure 1. Basic architecture of the TestGen-IF tool.

is displayed to inform the user. This message is followed by the description of the test purpose that was found. The number of test purposes already found and the number of those missing are also provided.

Based on this approach, a test suite is generated (represented by the Test Case Suite box). A test suite is composed of a finite set of test cases (or scenarios) described in a standard format. It is used to stimulate the implementation under test (IUT) to validate its reaction. As output of *TestGen-IF* tool, three files can be generated: (i) the "*output.aut*" file in Aldebaran [10] format containing the system behavior as labeled transition system, (ii) the "*output.xml*" file containing some information about the system execution (states, event, values of clocks, etc.), and (iii) the "*output.sequences*" file (in Aldebaran or TTCN [11] format) containing the timed test cases. This last output is represented in the Figure 1 by the boxes Aldebaran format and TTCN.

The test generation with *TestGen-IF* derives its benefits from Hit-or-Jump characteristics. It is faster that classical test generation tools (a gain of almost 20%) and less memory consuming. In addition, it avoids the state explosion and deadlock problems.

3 Passive Testing Tool

In some cases, active testing becomes difficult to perform. This is the case when the tester is provided without any direct interface (called also Observation and Control

Point) to interact with the IUT or when the implementation is built from components that are running in their environment and cannot be shutdown or interrupted for a long period of time. In such situations, there is a particular interest in using passive testing techniques. Indeed, *TestInv* tool does not need to interact with the IUT, it only need to collect the execution traces and then analyze them according to a set of *invariants* [3], without perturbing the IUT's behavior.

3.1 TestInv Background

3.1.1 Invariants

The distributed system informal description is studied in order to disclose particular properties that are true at any moment: that is why they are called invariants. These invariants can be classified into three categories: (i) simple invariants that define packets flow properties via input/output sequences (ii) obligation invariants that define conditionally mandatory events (iii) timed invariants that express input/output sequences with timed constraints.

3.1.2 Algorithms

Some algorithms that decide the correctness of the proposed invariants with respect to a given formal specification are provided in [3]. In *TestInv* tool, we also check whether the execution traces observed from the implementation respect the invariants. In order to perform this phase, we rely on three algorithms, one for each kind of invariant (simple, obligation or timed). These algorithms are inspired from [2].

3.2 TestInv Architecture

The main task of the *TestInv* prototype is to automate the process of checking the correctness of invariants on real system traces. This prototype code has been completely written in JAVA (J2SE 1.4.0 API specification) and the graphical interfaces have been developed using the Awt and Swing Java packages. The Regex2 Java package has been used to express invariants as regular expressions. This package proposes classes to match character sequences against patterns described by regular expressions. A high level description of the tool is given in Figure 2.

In order to start the passive testing process, we first have to obtain real traces from a running implementation (the gray box). To obtain such traces, different points of observation (POs) are set-up depending on the needs of the distributed testing architecture of the SUT (System Under Test).

The Pre-processing module processes the collected trace. The input file or data stream is transformed to a suitable format and is filtered in order to obtain information concerning

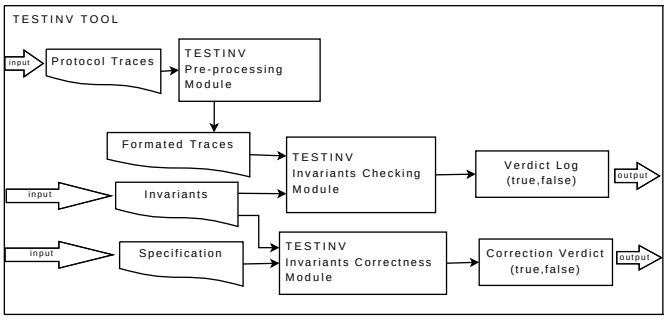

Figure 2. Basic architecture of the TestInv tool.

input and output primitives names as well as relevant data (e.g. source address, destination address, etc.). This module includes an interface allowing the user to parametrize the format of the traces to be collected. In other words, it is not necessary to modify the module code to process log files or data streams coming from different implementations (protocols, databases, ...) that use different formats.

The Invariant Correctness Module checks the correctness of the invariants on a given specification of the studied system, represented as a Timed Automata. The TestInv tool can perform this verification or not, depending on the user's choice. If not, then the IF specification of the IUT is not needed.

Finally, the Invariant Checking module determines if the captured traces satisfy the given list of invariants. The model of the output file or data stream can be customized. For each primitive name we assign the type in terms of input or output. The use of invariants in *TestInv* were first introduced to check properties on communication protocols, where relevant properties can be expressed as interactions between protocols entities and with the environment. These concepts are extended to be able to express properties that concerns other systems where properties are not only based on inputs and outputs, but also on actions, time stamps, predicates and references to different parts of the code.

Do must be noted that the first version of *TestInv* allowed protocol based trace analysis and detecting unauthorized communication packet sequences but was closely tied to the WAP protocol. The new version of the tool, has been generalized to deal with any protocol or service. Currently, it is in the process of being adapted to application based trace analysis (i.e. database log messages to detect unauthorized SQL injections) and code debugging based trace to detect misuse of memory. The use of invariants is being enriched to include variables and time constraints. It will also be possible to define a testing architecture to deploy several POs and to correlate their traces to monitor complex architectures and interactions.

4 Test Tools Applications

TestGen-IF is a recent tool that was applied to several case studies. In the context of this paper, we choose to present the experiments we carried out on the 'Travel' service. 'Travel' is an internal service used by France Telecom to manage 'missions' corresponding to travels by its employees. In this case study, a potential traveler can connect to the system to request for a travel ticket and a hotel reservation during a specific period according to a specific objective (called mission). This request can be accepted or rejected by his/her hierarchical superior. In the case of an acceptance, the travel ticket and hotel room are booked by contacting a specific travel agency. The specification of this 'Travel' system is performed using the IF formalism. Several test purposes with time constraints are proposed informally by France Telecom and are specified according to our methodology to automatically generate test cases that are later executed on a prototype version of Travel system.

The first prototype version of TestInv has been applied for validating protocols and services related to mobile environments (i.e. GSM/WAP, GPRS, UMTS). The aim was to check whether some protocol exchanges in WAP over GSM are correct, as well as to decide if different entities cooperate correctly. In addition, the tool was used to determine whether a given network service functions correctly. In the context of this paper, we want to present the application of TestInv to a real-time protocol called RTSP [12] (stands for Real Time Streaming Protocol). RSTP is a protocol for use in streaming media systems which allows a client to remotely control a streaming media server, issuing VCR-like commands such as ”play” and ”pause”, and allowing time-based access to files on a server.

5 Conclusion

In this paper, we present two complementary tools for the Formal Testing of Distributed Systems with Time Constraints. TestGen-IF allows to automatically generate test cases based on model-based active testing techniques. Whereas TestInv bases on passive testing approach to check the respect of collected system traces to a set of formal properties called *Invariants*. Both tools improves performances of classical tools with similar objectives and are the result of years of research in the testing field disseminated in several publications.

References

[1] http://www-verimag.imag.fr/˜async/if/.

[2] J. A. Arnedo, A. R. Cavalli, and M. Núñez. Fast Testing of Critical Properties Through Passive Testing. In D. Hogrefe and A. Wiles, editors, *TestCom*, volume 2644 of *Lecture Notes in Computer Science*, pages 295–310. Springer, 2003.

[3] E. Bayse, A. R. Cavalli, M. Núñez, and F. Zaïdi. A Passive Testing Approach Based on Invariants: Application to the WAP. *Computer Networks*, 48(2):235–245, 2005.

[4] M. Bozga, J.-C. Fernandez, L. Ghirvu, S. Graf, J.-P. Krimm, L. Mounier, and J. Sifakis. IF: An Intermediate Representation for SDL and its Applications. In *SDL Forum*, pages 423–440, 1999.

[5] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF Toolset. In M. Bernardo and F. Corradini, editors, *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 237–267. Springer, 2004.

[6] L. B. Briones and M. Röhl. Test Derivation from Timed Automata. In M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors, *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, pages 201–231. Springer, 2004.

[7] A. Cavalli, D. Lee, C. Rinderknecht, and F. Zadi. Hit-or-Jump: An Algorithm for Embedded Testing with Applications to IN Services. In *Formal Methods for Protocol Engineering And Distributed Systems*, pages 41–56, Beijing, China, october 1999.

[8] A. R. Cavalli, S. Maag, W. Mallouli, M. Marche, and Y.-M. Quemener. Application of Two Test Generation Tools to an Industrial Case Study. In *TestCom*, volume 3964 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2006.

[9] A. R. Cavalli and D. Vieira. An Enhanced Passive Testing Approach for Network Protocols. In *ICN/ICONS/MCL*, page 169. IEEE Computer Society, 2006.

[10] J.-C. Fernandez, H. Garavel, A. Kerbat, L. M. R. Mateescu, and M. Sighireanu. CADP : A Protocol Validation and Verification Toolbox. In *The 8th Conference on Computer-Aided Verification, CAV'96*, New Jersey, USA, August 1996. Springer Verlag.

[11] J. Grabowski, D. Hogrefe, G. Rethy, I. Schieferdecker, A. Wiles, and C. Willcock. An Introduction to The Testing and Test Control Notation (TTCN-3). In *Computer Networks 42(3)*, pages 375–403, 2003.

[12] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP), Internet RFC 2326, Apr. 1998.