# An Active Testing Tool for Security Testing of Distributed Systems

Mohamed H.E AOUADI
Software and Networks department
TELECOM SudParis
EVRY 91000, France
mohamed.aouadi@telecom-sudparis.eu

Khalifa Toumi
Software and Networks department
TELECOM SudParis
EVRY 91000, France
khalifa.toumi@telecom-sudparis.eu

Ana Cavalli
Software and Networks department
TELECOM SudParis
EVRY 91000, France
ana.cavalli@telecom-sudparis.eu

***Abstract*—This paper describes the TestGen-IF tool, that allows the automatic generation of test cases based on model-based active testing techniques. This paper describes the overall functionality and architecture of the tool, discusses its strengths and weaknesses, and reports our experience with using the tool on a case study, the Dynamic Route Planning (DRP) service of Vehicular Networks. This case study demonstrates how to use our testing tool to verify the system implementation against its security requirements. This paper also proposes improvements to this tool in the form of a GUI interface to facilitate its use and an approach which permits a gain in time and efficiency by generating test objectives.**

## I. Introduction

In recent decades, the complexity of distributed and real time systems has greatly increased. Therefore, formal testing is becoming a critical activity to guarantee that such systems respect the functional and security requirements. Among the formal testing techniques, there is the active testing method [1], [2] which validates a system's implementation by applying a set of test cases and analyzing its reaction.

This paper reports on a tool called TestGen-IF developed by our search team in Telecom SudParis. This tool aims to ease the active testing process and includes algorithms for automated test generation from a formal specification of the System-Under-Test (SUT).

The TestGen-IF tool has 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 exchange correctly, as well as to decide whether different entities can cooperate. In addition, it can also be used to decide whether a service is properly delivered and deployed.

The TestGen-IF tool is based on active testing techniques that are used if the interaction with the entity being tested 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 these kinds 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 on formal specification of the system described in IF (Intermediate Format) formal language [3], [4] implemented with TestGen-IF.

The TestGen-IF tool has some weaknesses. For instance, it can only run on Linux and the test objectives must be specified manually by the tester. To overcome these limitations, we propose a GUI interface that permits the TestGen-IF tool to be run on any operating system (Windows, Linux, Mac etc.) and that eases the use of the tool by using a human machine interface. We also propose an approach that permits the automatic generation of test objectives, which saves time and improves the efficiency. This paper is organised as follows: In section II and III we present the main characteristics of the TestGen-IF tool. In section IV we present a case study to show how this tool works. In sections V we propose an approach that permits the automatic generation of test objectives. In section VI we propose a GUI interface that facilitates the use of this tool. Finally, section VII concludes the paper an proposes some lines for future work.

## II. TestGen-IF Background

### A. IF language

A distributed system described using the IF language is composed of active processes running in parallel and interacting asynchronously through shared variables and message exchanges via communication channels or by direct addressing. Each IF process is described as a automaton extended with discrete data variables, various communication primitives, dynamic process creation and destruction.

### B. Hit-or-Jump exploration strategy

TestGen-IF implements an automated test generation algorithm based on a Hit-or-Jump exploration strategy [5] and a set of test purposes. This algorithm constructs test sequences efficiently with high fault coverage, avoiding the state explosion and deadlock problems encountered in exhaustive or exclusively random searches respectively. It enables the production of a partial accessibility graph of the SUT specification in conjunction with the IF simulator [4]. 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).

## III. TestGen-IF architecture

The active testing tool is illustrated in Figure 1. The Properties (*Test Purposes*) box represents the system objectives to be tested. For example, $action = input$ sig in $state = s$

978-1-4673-6590-1/15 $31.00 © 2015 IEEE
DOI 10.1109/ARES.2015.97

when clock $c = d$ describes an action constraint that expresses that the signal can be received in the state s when the valuation of clock c is d. The *Automatic Test Generation*

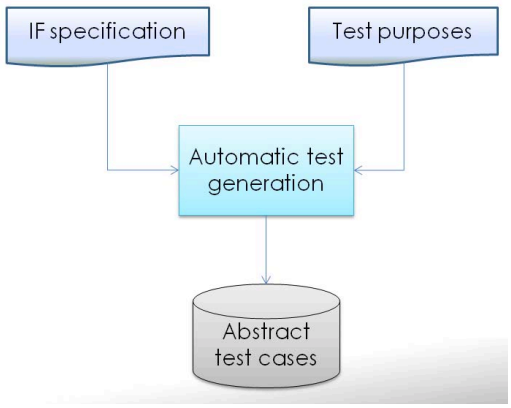


Fig. 1. TestGen-IF architecture

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 [2]. During this generation, when a test purpose is satisfied, a message 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 *Abstract 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.out file in Aldebaran [6] 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 [7] format) containing the timed test cases. The test generation with TestGen-IF derives its benefits from Hit-or-Jump characteristics. It is faster than classical test generation tools (a gain of almost 20%) and less memory consuming. In addition, it avoids the state explosion and deadlock problems.

## IV. A CASE STUDY: INTELLIGENT TRANSPORTATION SYSTEMS

TestGen-IF is a recent tool that was applied to several case studies [8], [9], [10], [11]. In the context of this paper, we choose to present the experiments we carried out on the Dynamic Route Planning (DRP) service.

DRP is a service used by vehicular networks, an example of intelligent transportation systems (ITS). ITS [12][13][14] is a global phenomenon, attracting worldwide interest from transportation professionals, the automotive industry and political decision makers. It applies advanced communication, information and electronics technology to solve transportation problems such as traffic congestion, safety, transport efficiency and environmental conservation. DRP aims at providing the drivers with an optimal route to reach their destination. This optimal route must take into account different changing factors such as the traffic, the weather, the condition of the roads. This route also allows for a reduction in travel times by choosing the most efficient succession of roads.

The user (driver) wants to reach his destination by the optimal route requiring the activation of the service through the client interface. The user must choose and activate only the DRP service. Once the service is activated, the control center (the server) checks if the user is authorized to access the service. To do so, the control center asks the user and the vehicle for some personal information such as identity and location. Due to security reasons, the vehicle must trust the control center before exchanging sensitive information. Therefore, a trust negotiation process takes place. If the trust negotiation process is successful, then the vehicle sends the required information to the control center which checks the information and either grants or declines access to the user. This scenario is formally defined by the finite state machine presented in Figure 2.

The specification of this process is performed using the IF formalism. Several test purposes are proposed informally and are specified according to our methodology to automatically generate test cases that are later executed on a prototype version of DRP system. In the following, we present a demonstration of how this test generation is made using TestGen-IF.

### A. IF specification

IF is a language based on temporized machines, allowing the description of existing concepts into specification formalisms. In order to use the TestGen-IF tool, we transcribed the formal model into an IF model. The description of a system in IF consists of the definition of data types, constants, shared variables, communication signals, and processes. In order to explain and illustrate these concepts, we exemplify in Figure 3 a sample code of our system (Vehicle-to-Infrastructure(V2I) under DRP service) in IF. In any IF model we have a system and some processes. In our model the system is *V2I*. The processes are the actors in the scenario. In our model we will consider the *vehicle* and the *control center* processes. The other actors will be considered as the environment. The vehicle process is described as extended finite state machine with data variables (cf. Figure 2. A transition includes output messages, and the assignment of variables values, etc. The vehicle in the negotiation scenario has a set of control states, as described in the formal model (*off-line, wait, wait-certificate, wait-decision, wait-info, wait-access, logged-in*) and has local data such as discrete variables. When a transition is executed, a process may change its state. The transition is caused by the reception of a message and the emission of a message. These messages are specified in our IF code as signals. Our specification contains the declaration of 7 signals describing the messages exchanged between the control center and the vehicle (i.e. `response_info`, `request_service`, `request_certificate`, `disagree_certificate`, `request_information`, `response`, `access_ok`) and 8 signals describing

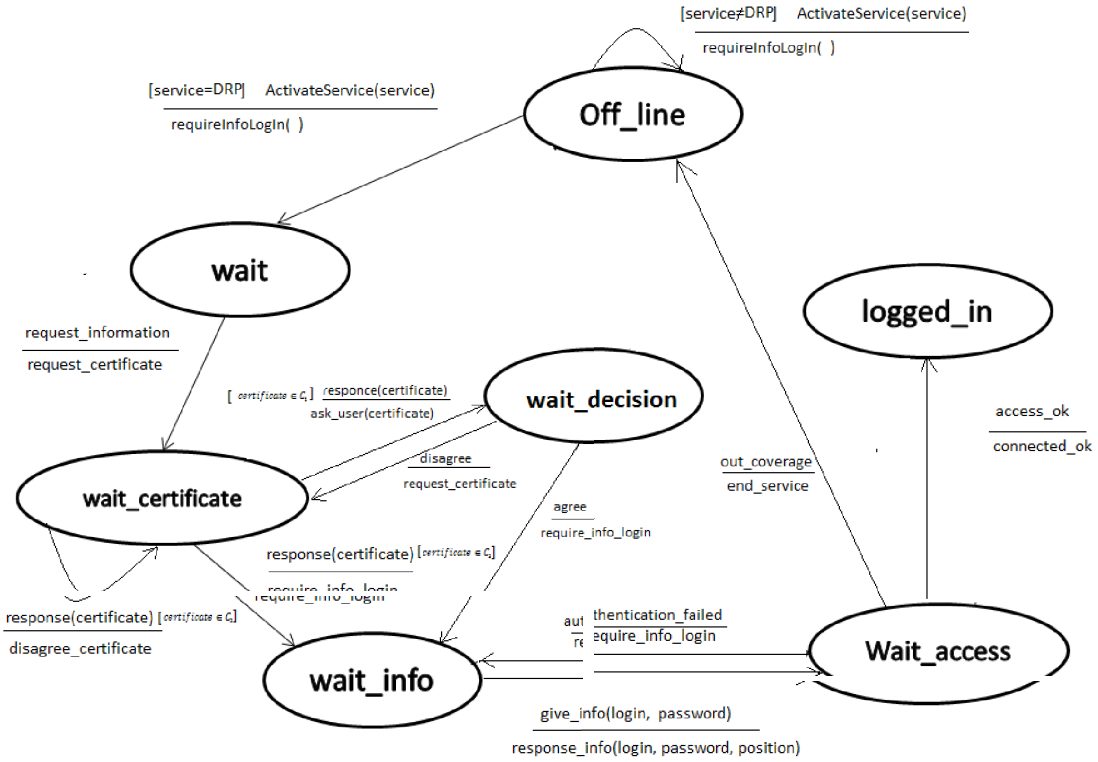


Fig. 2. Extended finite state machine of the vehicle

the messages exchanged between the vehicle and the environment ( i.e. `activate_service`, `give_info`, `agree`, `disagree`, `require_info_login`, `connected_ok`, `error_service`, `delegate_user`).
The IF code and the test purposes are the inputs required by TestGen-IF to generate test cases. A part of the IF code is presented in Figure 3.

### B. Test purposes

A test purpose or test objective describes a particular functionality of the implementation under test, by specifying the property to be checked in the system implementation. It is an observable action of the system that once described in IF language is used for guiding the space exploration of the system's states.
A test objective is described as a conjunction of conditions, including the following optional conditions: instance of a process with an identifier, a state of the system (a source state or a destination state), an action of the system (a message sent, a message received, an internal action), a variable of the process or a clock of the process, specifying a value and its state (active or inactive).
In order to generate test-cases, it is important first to define the set of test-purposes that the system under test has to respect. In the following we provide both informal and formal test purposes related to security and interoperability properties in a vehicular network system under DRP service. Next we show the generated corresponding test-case. Figure 4 shows an example of a formal specification of a test purpose which will be explained in the following subsection IV-C.

### C. Generation of abstract tests with testGen-IF

We consider the following testing scenario as an example: The vehicle is in the *wait_certificate* state. It receives a certificate from the control center with the value `certificate3` so it must delegate control to the user who will make a decision. Moreover, the vehicle must connect successfully to the service if the control center allows it (i.e. when the vehicle receives the message `connected_ok` from the control center). A test-case will be generated to check these two properties. To do so, a set of test objectives associated with this scenario, noted as OBJ(1), will be formulated. These are formally described in the Figure 4.

After editing the test-purposes, they were made available to the TestGen-IF tool as input together with the IF specification of the vehicular networks under DRP service. Consequently, the corresponding test-cases were derived. The abstract test-cases are composed of observable actions like:

- `?message` which represents the reception of a message.
- `!message` which represents the transmission of a message.

The test-case associated with this test-objective is represented in Figure 5.

```
system V2I;
/*constant definition*/
const SD = 100;
/*const NbService = 10;
...
/*type definition*/
type states = enum off_line, wait, wait_certificate,
          wait_info, wait_access, logged_in
        endenum;
....
 /* Signals definitions */
 /* channel 1 --- vehicle to CC */
signal response_info(login, password,position) ;
signal request_service(service);
....
/* Main process */
 process vehicle(1);
 /*local variables*/
var servicex service private;
...
 /* States specification */
state wait ;
    input GiveInfoLogIn(loginx,passx);
    output Forward_Inf_log_in(loginx,passx) ;
    nextstate waitForLogIn ;
.....
endstate;
....
endprocess;
endsystem;
```

Fig. 3. A sample code of the V2I under DRP system specification in IF

```
OBJ(1) = OBJ(ord) = {obj1, obj2}
    obj1 = cond1^cond2^cond3^cond4^cond5
        cond1 = process: instance = {vehicle}0
        cond2 = state: source: wait_certificate
        cond3 = state: destination: wait_decision
        cond4 = action: input response(certificate3)
        cond5 = action: output delegate_user ( )

    obj2 = cond1^cond2^cond3^cond4^cond5
        cond1 = process: instance = {vehicle}0
        cond2 = state: source: wait_access
        cond3 = state: destination: logged_in
        cond4 = action: input access_ok ( )
        cond5 = action: output connected_ok ( )
```

Fig. 4. The test objective OBJ(1)

## V. A FORMAL APPROACH TO GENERATING TEST OBJECTIVES

### A. Motivation

We consider the scenario of subsectionIV-C. As seen in this subsection, to verify a security rule we need a set of test objectives. This set of test objectives is specified manually.

```
?activate_service{DRP} !request_service{DRP}
?request_information{} !request_certificate{}
?response{certificate3} !delegate_user{certificate3}
?agree{} !require_info_login{}
?give_info{login1,pass1} !response_info{login1,pass1,currentPosition}
?access_ok{} !connected_ok{}
```

Fig. 5. The test case corresponding to the OBJ(1)

This specification is possible in this case. However, it becomes time consuming with the risk of errors if we have a large number of possible parameter values. For instance, if we have infinite possible values (or a very large number of possible values) of the `certificate` parameter the test objectives specification becomes impossible (or time consuming). To tackle this problem we propose a formal approach that permits the automatic generation of a set of test purposes whenever a parameter has many possible values.

### B. The objective generation approach

The approach is based on an algorithm which takes as input the signal (input or output), the initial state, and the parameter. Based on the specification of the system provided by the EFSM, the algorithm will generate a set of test objectives.

**Algorithm 1: Test Objectives Generation**

Require: The initial state $S1$, the signal $input$, the process $process$, and the variable $parameter$.

1: for each $value_i$ of the $parameter$
$output_i$=$g(value_i, S1, input)$
$destination_i$=$f(value_i, S1, input, output_i)$

2: function f(p,s,i,o){
return the state S2 destination of the transition ?i(p)!o which leaves from s }

3: function g(p,s,i){
return the output o sent by the system when we apply i(p) on the state s }

4: write
$obj_i$ = cond1 $\wedge$ cond3 $\wedge$ cond4 $\wedge$ cond5
cond1 = process: instance = $\{process\}0$
cond2 = state: source: $S1$
cond3 = state: destination: $destination_i$
cond4 = action: input $input(value_i)$
cond5 = action: output $output_i$

### C. Example

Consider the system shown in Figure2 which is in a `wait_certificate` state. Under this state, the vehicle is waiting for the certificate from the control center. The control center sends a certificate via the message (input to the vehicle)

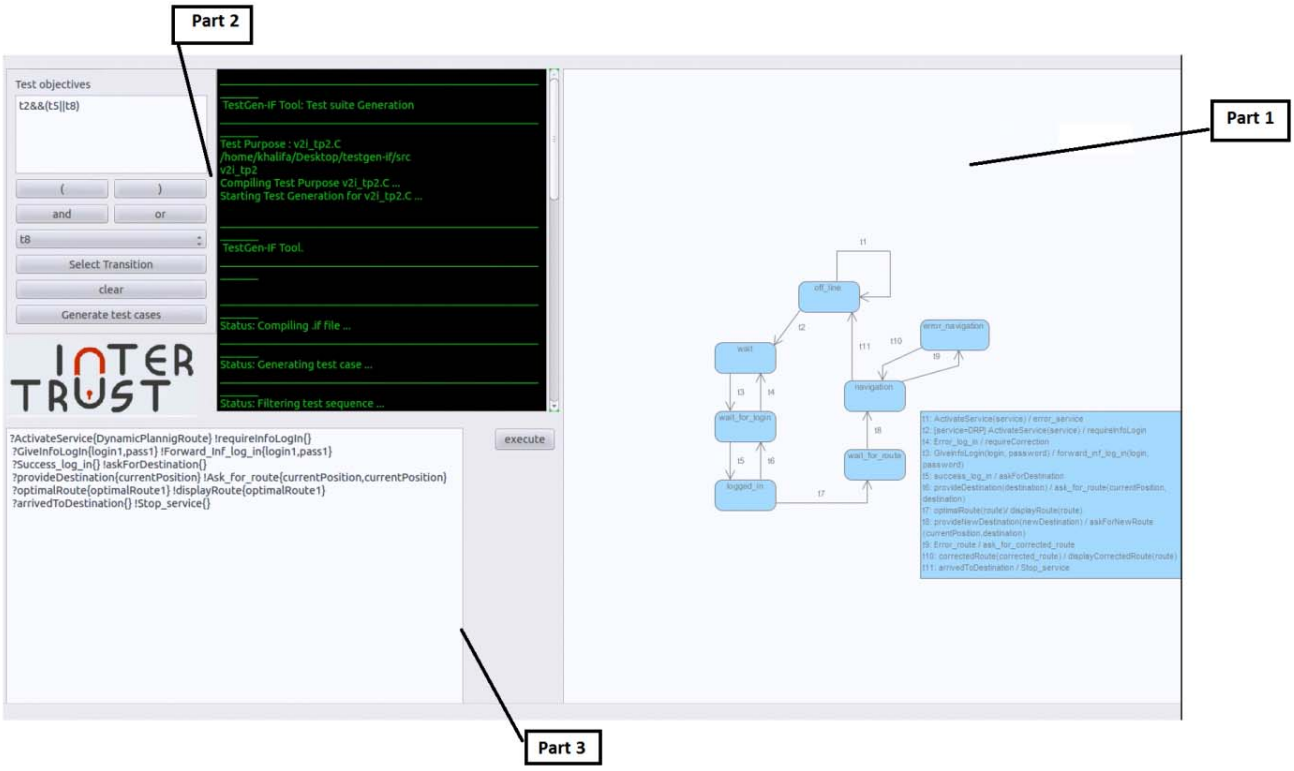


Fig. 6. The GUI interface of the TestGen-IF tool

`response(certificate)`. Depending on the certificate value the system can go to the state `wait_decision`, `wait_info`, or stay in its current state. Our algorithm takes as input the current state `wait_certificate`, the input without parameter `response(certificate)`, and the parameter `certificate`. Then the algorithm defines all possible test objectives by calculating for each test objective the output and the destination state. By applying this algorithm to our case study we obtain the three test objectives shown below.

$obj_1$ cond1 $\wedge$ cond3 $\wedge$ cond4 $\wedge$ cond5
cond1 = process: instance = $\{vehicle\}0$
cond2 = state: source: $wait\_certificate$
cond3 = state: destination: $dwait\_info$
cond4 = action: input $response(certificate01)$
cond5 = action: output $require\_info\_login$

$obj_2$ cond1 $\wedge$ cond3 $\wedge$ cond4 $\wedge$ cond5
cond1 = process: instance = $\{vehicle\}0$
cond2 = state: source: $wait\_certificate$
cond3 = state: destination: $wait\_decision$
cond4 = action: input $response(certificate02)$
cond5 = action: output $ask\_user(certificate02)$

$obj_3$ cond1 $\wedge$ cond3 $\wedge$ cond4 $\wedge$ cond5
cond1 = process: instance = $\{vehicle\}0$
cond2 = state: source: $wait\_certificate$
cond3 = state: destination: $wait\_certificate$
cond4 = action: input $response(certificate03)$
cond5 = action: output $disagree\_certificate$

## VI. TestGenIF interface

Another contribution of this paper is the specification and the development of a graphical user interface. This interface aims to simplify the tester tasks and to enhance the application efficiency. Figure 6 shows the interface of our system. It is composed of 3 parts:

- The first one contains the EFSM of the application. This one contains all the details about the transitions and the states.
- The second one offers the possibility to choose the test purposes. With this tool, the tester has only to choose the transitions to be tested. No commands or a configuration should be done.
- The third part will show the results that are the abstract test cases. Moreover, we have added a new extension of the TestGen-IF tool in the form of an execution engine. This engine will permit the test to be executed automatically. However, this process would have to be

updated based on the application. This engine permits also the translation of the abstract test case into a concrete one for some web applications. However, this translation would need to be updated for use in other applications.

## VII. Conclusion

In this paper, we present a tool for the Formal Testing of Distributed Systems based in active testing techniques. TestGen-IF allows the automatic generation of test cases based on model-based active testing techniques. This tool is an improvement of the performance of classical tools with similar objectives and is the result of years of research in the testing field disseminated in several publications. We also propose improvements in our tool by an interface and an approach to automatic generation of test objectives. In future we plan to add to the testGen-IF tool an extension that permits the generation of concrete test cases rather than abstract test cases.

## References


[1] M. Broy, *Model-based testing of reactive systems: advanced lectures*. Springer, 2005.

[2] A. Cavalli, S. Maag, W. Mallouli, M. Marche, and Y.-M. Quemener, "Application of two test generation tools to an industrial case study," in *Testing of Communicating Systems*. Springer, 2006, pp. 134–148.

[3] M. Bozga, J.-C. Fernandez, C. L. Ghirvu, S. Graf, K. J. Pierre, L. Mounier, J. Sifakis *et al.*, "If: An intermediate representation for sdl and its applications," in *SDL'99 The Next Millennium, 9th International SDL Forum Proceedings*, 1999, pp. 423–440.

[4] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis, "The if toolset," in *Formal Methods for the Design of Real-Time Systems*. Springer, 2004, pp. 237–267.

[5] A. Cavalli, D. Lee, C. Rinderknecht, and F. Zaïdi, "Hit-or-jump: An algorithm for embedded testing with applications to in services," in *Formal Methods for Protocol Engineering And Distributed Systems*. Springer, 1999, pp. 41–56.

[6] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu, "Cadp a protocol validation and verification toolbox," in *Computer Aided Verification*. Springer, 1996, pp. 437–440.

[7] J. Grabowski, D. Hogrefe, G. Réthy, I. Schieferdecker, A. Wiles, and C. Willcock, "An introduction to the testing and test control notation (ttcn-3)," *Computer Networks*, vol. 42, no. 3, pp. 375–403, 2003.

[8] M. Lallali, F. Zaidi, and A. Cavalli, "Transforming bpel into intermediate format language for web services composition testing," in *Next Generation Web Services Practices, 2008. NWESP'08. 4th International Conference on*. IEEE, 2008, pp. 191–197.

[9] I. Hwang, M. Lallali, A. Cavalli, and D. Verchere, "Modeling, validation, and verification of pcep using the if language," in *Formal Techniques for Distributed Systems*. Springer, 2009, pp. 122–136.

[10] I. Hwang, A. R. Cavalli, M. Lallali, and D. Verchere, "Applying formal methods to pcep: an industrial case study from modeling to test generation," *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 343–361, 2012.

[11] M. Lallali, F. Zaidi, A. Cavalli, and I. Hwang, "Automatic timed test case generation for web services composition," in *on Web Services, 2008. ECOWS'08. IEEE Sixth European Conference*. IEEE, 2008, pp. 53–62.

[12] G. Dimitrakopoulos and P. Demestichas, "Intelligent transportation systems," *Vehicular Technology Magazine, IEEE*, vol. 5, no. 1, pp. 77–84, 2010.

[13] W. Barfield and T. A. Dingus, *Human factors in intelligent transportation systems*. Psychology Press, 2014.

[14] J. Zhang, F.-Y. Wang, K. Wang, W.-H. Lin, X. Xu, and C. Chen, "Data-driven intelligent transportation systems: A survey," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 12, no. 4, pp. 1624–1639, 2011.