Using a mining frequency patterns model to automate passive testing of real-time systems *

César Andrés, Mercedes G. Merayo, Manuel Núñez
Departamento Sistemas Informáticos y Computación
Universidad Complutense de Madrid
{c.andres,mgmerayo}@fdi.ucm.es, mn@sip.ucm.es

Abstract

Testing is one of the most widely used techniques to increase the confidence on the correctness of complex software systems. In this paper we extend our previous work on passive testing with invariants to incorporate (probabilistic) knowledge obtained from users of the system under test. In order to apply our technique, we need to obtain a set of invariants compiling the relevant properties of the system under test, and this is a difficult task. First, we present an algorithm to extract invariants from the specification without assuming any additional condition. Since the number of obtained invariants is huge we study an alternative. Based on the idea that an invariant is better than another one if it can be checked more times in the same log, we present an adaptation of the previous algorithm in order to sort sets of representative invariants.

1 Introduction

With the growing significance of software systems, techniques that assist in the production of reliable software are becoming increasingly important within software engineering. The complexity of current software systems requires the application of sound techniques. Among these techniques, testing [13] is the most widely used in industrial environments. Unfortunately, testing is still a mainly manual activity, prone to errors that can mask real errors of the tested system. Traditionally, formal methods and testing have been seen as rivals. Thus, there was very little interaction between the two communities. In recent years, however, these approaches are seen as complementary [9, 8].

*Research supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01), the MATES project (CCG08-UCM/TIC-4124) and by the UCM-BSCH programme to fund research groups (GR58/08 - group number 910606).

In most cases, testing is based on the ability of a tester that stimulates the implementation under test (IUT) and checks the correction of the answers provided by the implementation. However, in some situations this activity becomes difficult and even impossible to perform. For example, this is the case if the tester is not provided with a direct interface to interact with the IUT. Another conflictive situation appears 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 these situations, there is a particular interest in using other types of validation techniques such as *passive testing*. Therefore, the main different between active and passive testing is that in active testing testers can interact, by providing inputs, with the IUT and observe the obtained result.

Passive testing usually consists in recording the trace produced by the IUT and trying to find a *fault* by comparing this trace with the specification [10, 14, 12, 5]. A new methodology to perform passive testing was presented in [6, 4]. The main novelty is that a set of *invariants* is used to represent the most relevant properties of the specification. An invariant can be seen as a restriction over the traces allowed to the IUT. We introduced in previous work [3] the possibility of adding time constraints as properties that traces extracted from the IUT must hold.

Normally, there are several ways to obtain a set of invariants. The first one is that testers provide a set of representative invariants. Then, the correctness of these invariants, with respect to the specification, has to be checked. The main drawback of this approach is that we still have to rely on the manual work of the tester. If we do not have a complete formal specification, a variant of this approach is to assume that the set of invariants provided by the testers are correct by definition. In this paper we consider an alternative approach: We automatically derive invariants from the

specification. First, we consider an adaptation of the algorithms presented in [6]. The problem with this first attempt is that the number of invariants that we extract from the specification is huge and we do not have a criterion to decide which ones are *better*. We propose an alternative algorithm to use knowledge extracted from *standard* users of the system, so that invariants can be sorted according to their *quality* to fit the behaviuor of those users.

The rest of the paper is structured as follows. In Section 2 we present our passive testing framework of timed systems. In Section 3, containing the bulk of the paper, we present two algorithms for extracting a set of correct invariants from the specification. We conclude with Section 4, where we present the conclusions and some lines of future work.

2 A formal framework for passive testing of timed systems

In this section we introduce our framework to specify and (passively) test timed systems. We extend the well-known Finite State Machines model by adding time information concerning the amount of time that the system needs to perform transitions.

Definition 1 A *Timed Finite State Machine (`TFSM`)* is a tuple $(\mathcal{S}, s_0, \mathcal{I}, \mathcal{O}, \mathcal{T})$, where $\mathcal{S}$ is a finite set of states, $s_0 \in S$ is the initial state, $\mathcal{I}$ and $\mathcal{O}$, with $\mathcal{I} \cap \mathcal{O} = \varnothing$, are the finite sets of *input* and *output* actions, respectively, and $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{I} \times \mathcal{O} \times \mathbb{R}_+ \times \mathcal{S}$ is the set of transitions.

A `TFSM` $M = (\mathcal{S}, s_0, \mathcal{I}, \mathcal{O}, \mathcal{T})$ is *deterministic* if for all state s and input i there exists at most one transition (s, i, o, t, s'). We say that M is *input-enabled* if for all state $s \in \mathcal{S}$ and $i \in \mathcal{I}$ there exists a transition (s, i, o, t, s'). □

Let us consider the `TFSM` depicted in Figure 1 and the transition $(s_1, a, x, 3, s_2)$. Intuitively, if the machine is at state s_1 and it receives the input a, then it will produce the output x after 3 time units. We usually write $s \xrightarrow{i/o}_t s'$ as a shorthand of $(s, i, o, t, s') \in \mathcal{T}$. As usual in formal testing approaches, we assume that both specifications and implementations can be represented by the same formalism. In our case, the formalism is the set of deterministic input-enabled `TFSMs`.

Next we introduce the notion of *trace* of a system. A trace captures the behaviour of an implementation. Traces collect the outputs obtained after sending some inputs to the implementation and the amount of time between each input/output pair. The classical notion of trace, which does not include any time information, is called a *non-timed trace*.

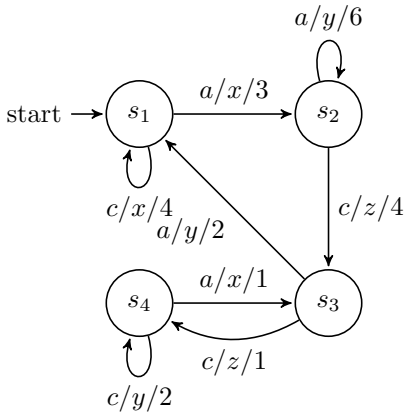

Figure 1. Example of `TFSM`.

Definition 2 Let $M = (\mathcal{S}, s_0, \mathcal{I}, \mathcal{O}, \mathcal{T})$ be a `TFSM`. We say that $\theta = \langle i_1/o_1/t_1, i_2/o_2/t_2, \ldots, i_n/o_n/t_n \rangle$ is a *timed trace*, or simply trace, of M if there is a sequence of transitions such that

$$s_1 \xrightarrow{i_1/o_1}_{t_1} s_2 \xrightarrow{i_2/o_2}_{t_2} s_3 \ldots s_{n-1} \xrightarrow{i_n/o_n}_{t_n} s_n$$

We denote the empty trace by $\langle\rangle$ and by $\mathtt{tr}(M)$ the set of all traces of M.

If $\theta = \langle i_1/o_1/t_1, \ldots, i_n/o_n/t_n \rangle$ is a timed trace of M, then the sequence $\omega = \langle i_1/o_1, \ldots, i_n/o_n \rangle$ is a *non-timed trace* of M. We denote by $\mathtt{nttr}(M)$ the set of all non-timed traces of M. □

For example, if we consider the machine M depicted in Figure 1, we have that $\langle c/x/4, c/x/4, a/x/3, c/z/4, c/z/1, a/x/1 \rangle$ is a timed trace of M. If we remove time information, we obtain *non-timed* traces. For instance, $\langle c/x, c/x, a/x, c/z, c/z, a/x \rangle$ is the non-timed trace associated with the previous timed trace.

Our passive testing approach is similar to other methodologies since the basic idea consists in recording traces from the `IUT` to detect unexpected behaviours. The main novelty is that a set of *invariants* is used to represent the most relevant properties of the specification. Intuitively, an invariant expresses the fact that each time the `IUT` performs a given sequence of actions, then it must exhibit a behaviour reflected in the invariant. In order to express traces in a concise way, we will use the wild-card characters ? and $\star$. The wild-card ? represents any value in the sets of inputs or outputs, while $\star$ represents a sequence of input/output pairs.

Definition 3 We say that $\hat{p} = [p_1, p_2]$ is a *time interval* if $p_1 \in \mathbb{R}_+$, $p_2 \in \mathbb{R}_+ \cup \{\infty\}$, and $p_1 \leq p_2$.

We assume that for all $t \in \mathbb{R}_+$ we have $t < \infty$ and $t + \infty = \infty$. We consider that $\mathcal{IR}$ denotes the set of time intervals.

Let $M = (\mathcal{S}, s_0, \mathcal{I}, \mathcal{O}, \mathcal{T})$ be a TFSM. We say that the sequence φ is an *invariant* for M if the following two conditions hold:

1. φ is defined according to the following EBNF:

$$\begin{array}{ll} \varphi & ::= a/z/\hat{p}, \varphi \mid \star/\hat{p}, \varphi' \mid i \mapsto O/\hat{p} \rhd \hat{t} \\ \varphi' & ::= i/z/\hat{p}, \varphi \mid i \mapsto O/\hat{p} \rhd \hat{t} \end{array}$$

   In this expression we consider $\hat{p}, \hat{t} \in \mathcal{IR}$, $i \in \mathcal{I}$, $a \in \mathcal{I} \cup \{?\}$, $z \in \mathcal{O} \cup \{?\}$, and $O \subseteq \mathcal{O}$.

2. φ is *correct* with respect to M (formally defined in [3]).

□

Even though, our machines present time information expressed as fix amounts of time, time conditions established in invariants are given by intervals. This fact is due to consider that it can be admissible that the execution of a task sometimes lasts more time than expected: If most of the times the task is performed on time, a small number of delays can be tolerated. Concerning the notion of correctness, the idea is that an invariant is correct with respect to a machine M if M cannot perform a sequence of transitions that would contradict what the invariant expresses.

Intuitively, the previous EBNF expresses that an invariant is either a sequence of symbols where each component, but the last one, is either an expression $a/z/\hat{p}$, with a being an input action or the wild-card character ?, z being an output action or ?, and $\hat{p}$ being an interval, or an expression $\star/\hat{p}$. There are two restrictions to this rule. First, an invariant cannot contain two consecutive expressions $\star/\hat{p}_1$ and $\star/\hat{p}_2$. In the case that such situation was needed to represent a property, the tester could simulate it by means of the expression $\star/(\hat{p}_1 + \hat{p}_2)$. The second restriction is that an invariant cannot present a component of the form $\star/\hat{p}$ followed by an expression beginning with the wild-card character ?, that is, the input of the next component must be a *real* input action $i \in \mathcal{I}$. In fact, $\star$ represents any sequence of inputs/outputs pairs such that the input is not equal to i.

The last component, corresponding to the expression $i \mapsto O/\hat{p} \rhd \hat{t}$ is an input action followed by a set of output actions and two timed restrictions, denoted by means of two intervals $\hat{p}$ and $\hat{t}$. The former is associated to the last expression of the sequence. The latter is related to the sum of times values associated to all input/output pairs performed before. For example, the meaning of an invariant as $i/o/\hat{p}, \star/\hat{p_\star}, i' \mapsto O/\hat{p}' \rhd \hat{t}$ is that if we observe the transition i/o in a time belonging to the interval $\hat{p}$, then the first occurrence of the input symbol i' after a lapse of time belonging to the interval $\hat{p_\star}$, must be followed by an output belonging to the set O. The interval $\hat{t}$ makes reference to the total time that the system must spend to perform the whole trace. Next we introduce some examples in order to present how invariants work.

Example 1 Let us consider the TFSM presented in Figure 1. One of the most simple invariants that we can define within our framework follows the scheme $a \mapsto \{x, y\}/[1, 6] \rhd [1, 6]$. The idea is that each occurrence of the symbol a is followed by either the output symbol x or y and this transition is performed between 1 and 6 time units.

We can specify a more complex property by taking into account that we are interested in observing the output x after the input a when the pair c/x was previously observed. In addition, we include time intervals corresponding to the amount of time the system takes for each of the transitions and to the total time it spends in the whole trace. We could express this property by means of the invariant $c/x/[3, 5], \star/[0, 5], a \mapsto \{x\}/[2, 4] \rhd [4, 13]$. An observed trace will be correct with respect to this invariant if each time that we find a (sub)sequence starting with the c/x pair, performed in an amount of time belonging to the interval $[3, 5]$, if there is an occurrence of the input a before 5 time units pass, then it must be paired with the output symbol x and the lapse of time between a and c must belong to the interval $[2, 4]$. In addition, the whole sequence must take a time belonging to the interval $[4, 13]$. □

3 Two approaches for the extraction of invariants from specifications

Normally, invariants should be supplied by the expert/tester. Then, the first step is to check that the invariants are correct with respect to the specification. Another approach consists in extracting invariants from the specification, so that their correctness is ensured. In this section we present two algorithms to extract invariants from specifications. The first algorithm is based on the adaptation to our framework of the algorithms given in [6]. The main drawback of applying this approach is that, in general, we have a huge set of invariants. The second algorithm, uses extra information about the users of the system and improves the first algorithm in two features. The first one is that the set of invariants is smaller. The second one is that

in : $M = (\mathcal{S}, s_0, \mathcal{I}, \mathcal{O}, \mathcal{T})$: TFSM
out: ϕ : Set of correct invariants
1 $\Omega = \mathtt{nttr}(M)$;
2 $\phi = \varnothing$;
3 **while** $(\Omega \neq \varnothing)$ **do**
4 | **Choose** $\omega \in \Omega$;
5 | $\Omega = \Omega \setminus \{\omega\}$;
6 | $\phi = \phi \cup \mathtt{generateT}(M, \omega)$;
7 **end**
8 **return** ϕ;

Figure 2. Algorithm for generating invariants from a TFSM adapting algorithms in [6].

we can establish a minimum *significance degree* value for the set of invariants.

The adaptation to our framework of the algorithms given in [6] is presented in Figure 2. The next auxiliary functions are used in the algorithm.

Definition 4 Let $M = (\mathcal{S}, s_0, \mathcal{I}, \mathcal{O}, \mathcal{T})$ be a TFSM and $\omega = \langle i_1/o_1, \ldots, i_n/o_n \rangle$ be a non-timed trace of M. The function $\mathtt{outputs}(M, \omega)$ computes the set of all possible outputs associated with the last input of ω.

$$\mathtt{outputs}(M, \omega) = \left\{ o \;\middle|\; \langle i_1/o_1, \ldots, i_n/o \rangle \in \mathtt{nttr}(M) \right\}$$

The functions $\mathtt{mT}(\omega, j, O)$ and $\mathtt{MT}(\omega, j, O)$ compute the minimum, respectively maximum, time value associated to the pair i_j/o_j in all the timed traces of M corresponding to ω, except the last output that must belong to O, that is,

$$\mathtt{mT}(\omega, j, O) = \mathtt{min}\left\{ t_j \;\middle|\; \langle i_1/o_1/t_1, \ldots, i_n/o/t_n \rangle \in \mathtt{tr}(M) \wedge o \in O \right\}$$

$$\mathtt{MT}(\omega, j, O) = \mathtt{max}\left\{ t_j \;\middle|\; \langle i_1/o_1/t_1, \ldots, i_n/o/t_n \rangle \in \mathtt{tr}(M) \wedge o \in O \right\}$$

The function $\mathtt{generateT}(M, \omega)$ computes a set of correct invariants with respect to M corresponding to the sequence ω.

$$\left\{ \varphi \;\middle|\; \begin{array}{l} \varphi = i_1/o_1/\hat{p_1}, \ldots, i_n \mapsto O/\hat{p}_n \rhd \hat{q} \wedge \\ O = \mathtt{outputs}(M, \omega) \wedge \\ \forall 1 \leq k \leq n : \hat{p}_k = [\mathtt{mT}(\omega, k, O), \mathtt{MT}(\omega, k, O)] \wedge \\ \hat{q} = \sum \hat{p}_k \end{array} \right\}$$

□

Next, we briefly describe this algorithm. The input parameter is the specification of the system, while the output is a set of correct invariants generated from it. The algorithm applies the function generateT to all the non-timed traces Ω of the specification in order to produce a set of correct invariants ϕ. The drawback of this proposal is that the number of possible invariants is really huge, usually infinite, and most of them have low significance. Let us note that this set is usually infinite due to the fact that we compute all non-timed traces of the specification.

In order to reduce the number of invariants we propose an algorithm that uses extra information obtained by the application of data mining techniques to a set of data recorded from the interaction between different users and the IUT. First, a selection of data is made and preprocessed in order to check that there does not exist any incongruence, that is, the set of data represents traces of users that have been observed during their interaction with the IUT. We will focus on extracting sets of *relevant behaviors* from this set of data, that is, those sequences of interactions of the users with the system that appear more frequently. In order to determine these behaviors we use the usual techniques for obtaining frequent patterns from a database (see, for example, [1, 11]). The task of discovering the frequency of each behaviour in the database is performed by means of the applications of any of the existing tools. In our approach we use [7].

In order to determine the knowledge obtained from the data, we formally define a *frequency threshold* relative to a set of sequences. The frequency threshold represents the degree of relative frequency that a set of behaviours has with respect to the global set of behaviours [15]. Let us note that in order to define the user behvaiour, we abstract outputs and time values since the are not relevant for this phase.

Definition 5 A *user behaviour* is a sequence of input actions $\alpha = \langle i_1, i_2, \ldots, i_n \rangle$ where $n \geq 0$. A *database* $\mathcal{DB}$ is a multiset of user behaviours.

If α' is a subtrace of α then we denote it by $\alpha' \leq \alpha$. The set of all prefixes of α is denoted by $\mathtt{tu}(\alpha)$.

We denote by $\mathtt{prob}(\alpha, \mathcal{DB})$ the probability of finding the user behaviour α in the database $\mathcal{DB}$.

The *prefix-closed frequency* of a set of input sequences Σ over the database $\mathcal{DB}$ is defined as:

$$\mathtt{c}_{\mathcal{DB}}(\Sigma) = \sum_{\alpha \in \bigcup_{\alpha' \in \Sigma} \mathtt{tu}(\alpha')} \mathtt{prob}(\alpha, \mathcal{DB})$$

□

We assume that the world is *regular*. Therefore, if we have a big number of user behaviours in the database, we are able to consider that we have an acceptable percentage of the amount of usual interactions with the system. Let us note that the idea of being provided with a knowledge base which contains a representation

of an expertise in the domain, that is our database, is not new; it is a usual assumption considered in a expert system.

In Figure 3 we adapt the previous algorithm presented in Figure 2 to include the extra information that we have obtained by using our data mining process. The goal of this algorithm is still to obtain a set of correct invariants from the specification. The problem that we had in the previous algorithm, regarding to the high number of invariants that we generated, is solved by focusing only on the invariants that represent a *normal* behaviour of a user interacting with the IUT. Let us remember that we abstract the outputs and the time values.

Next we define a set of auxiliary functions which are used in the algorithm.

Definition 6 Let $\mathcal{DB}$ be a database. The function msTOs computes the set of user behaviours associated to a database, that is,

$$\texttt{msTOs}(\mathcal{DB}) = \{\alpha | \alpha \in \mathcal{DB}\}$$

The function moreR computes the most representative sequence of input actions of a database $\mathcal{DB}$ that does not belong to a set of input sequences Σ, that is,

$$\texttt{moreR}(\mathcal{DB}, \Sigma) = \begin{cases} \alpha & \textbf{if } \exists \alpha : \alpha \in \texttt{msTOs}(\mathcal{DB}) \setminus \Sigma \wedge \\ & \forall \alpha' : \ \alpha' \in \texttt{msTOs}(\mathcal{DB}) \setminus \Sigma \wedge \\ & \qquad \texttt{c}_{\mathcal{DB}}(\{\alpha\}) \geq \texttt{c}_{\mathcal{DB}}(\{\alpha'\}) \\ \langle\rangle & \textbf{otherwise} \end{cases}$$

Let $M = (\mathcal{S}, s_0, \mathcal{I}, \mathcal{O}, \mathcal{T})$ be a TFSM, and $\alpha = \langle i_1, \ldots, i_n \rangle$ be a single sequence of input actions. The function generateT'(M, α) computes a set of correct invariants with respect to M corresponding to the inputs sequence α, that is,

$$\left\{ \varphi \;\middle|\; \begin{array}{l} \varphi = h_1, \ldots, h_{n-1}, i_n \mapsto \{O\}/\hat{p}_n \rhd \hat{q} \wedge \\ h_1 = i_1/o_1/\hat{p}_1 \wedge \ldots \wedge h_{n-1} = i_{n-1}/o_{n-1}/\hat{p}_{n-1} \wedge \\ \exists \omega = \langle i_1/o_1, \ldots, i_n/o \rangle \in \texttt{nttr}(M) \wedge \\ O = \texttt{outputs}(M, \omega) \wedge \\ \forall 1 \leq k \leq n : \hat{p}_k = [\texttt{mT}(\omega, k, O), \texttt{MT}(\omega, k, O)] \wedge \\ \hat{q} = \sum \hat{p}_k \end{array} \right\}$$

□

Next, we briefly describe the main steps of this algorithm. As input parameters of the algorithm we have the specification, modelled using a TFSM, the database $\mathcal{DB}$ containing the stored sequences of interactions between users and the IUT and, the significance degree over the database that it is required. The results of our algorithm is a set of correct invariants ϕ.

in : $M = (\mathcal{S}, s_0, \mathcal{I}, \mathcal{O}, \mathcal{T})$: TFSM
$\mathcal{DB}$: Database.
$\mathcal{C} : \mathbb{R}_+$ Grade of relevance.
out: ϕ: Set of correct invariants.

```
1  Σ = ∅; φ = ∅;
2  while (c_DB(Σ) < C) ∧ (msTOs(DB) \ Σ ≠ ∅) do
3  |  α = moreR(DB, Σ);
4  |  Σ = Σ ∪ {α};
5  end
6  // We will enter this loop only if maximum
7  // coverage is not reached
8  while (c_DB(Σ) < C) do
9  |  Choose i ∈ I; Σ' = Σ; Σ = ∅;
10 |  while Σ' ≠ ∅ do
11 |  |  Choose α ∈ Σ'; Σ' = Σ' \ {α};
12 |  |  Σ = Σ ∪ {α} ∪ {i · α};
13 |  end
14 end
15 // Loop to generate invariants from stored
16 // information
17 while (Σ ≠ ∅) do
18 |  Choose α ∈ Σ; Σ = Σ \ {α};
19 |  φ = φ ∪ generateT'(M, α);
20 end
21 return φ;
```

Figure 3. Algorithm for generating invariants from a TFSM using statistical information.

We have two different phases in this algorithm. The first one selects a set of user behaviours Σ which represents at least the grade of relevance established. Initially we choose the most significant sequences of user interactions from the database applying the function moreR. If the grade of relevance that we obtain after computing all sequences of user interactions in the database is less than the one requied, we extend the traces in Σ until we reach it. The second phase uses the set Σ and generates a set of correct invariants with respect to the provided specification, storing them in ϕ which will be the set that the algorithm returns. The function generateT', given in Definition 6 is used for generating a set of correct invariants given a sequence of inputs. Let us remark that this function is different form the function generateT used in the previous algorithm. The new function considers a sequence of inputs, while the previous one considered a sequence of input/output pairs.

When we compare the set of invariants obtained from this second algorithm with respect to the set of invariants obtained from the first algorithm we see that

the cardinal of this new set is always less than or equal to the obtained with respect to the first one.

4 Conclusions and Future Work

In this paper we have extended our passive testing framework with two algorithms to generate invariants from the specification. Our invariants can be seen as rules that reflect functional and non-functional aspects of the specification that must be hold by any trace produced by the `IUT`. As usual, the set of all possible traces from an implementation is infinite and we want to generate a representative set of invariants which reflect the most often interactions between users and implementation. We present a methodology based on data mining from a database which contains the interactions between users and the system, in order to obtain the most frequent interactions sequences. We use this extra information to guide the second algorithm.

As future work we will focus on adapting a formal framework, such as the one presented in [2], to the task of representing a user model which reflects the behaviour of a user. We will use this model to reflect the probability to perform an action and we will adapt the second algorithm provided in this paper with this new formalism.

References

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *19th ACM Int. Conf. on Management of Data, SIGMOD'93*, pages 207–216. ACM Press, 1993.

[2] C. Andrés, L. Llana, and I. Rodríguez. Formally comparing user and implementer model-based testing methods. In *4th Workshop on Advances in Model Based Testing, A-MOST'08*, pages 1–10. IEEE Computer Society Press, 2008.

[3] C. Andrés, M.G. Merayo, and M. Núñez. Passive testing of timed systems. In *6th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'08, LNCS 5311*, pages 418–427. Springer, 2008.

[4] E. Bayse, A. Cavalli, M. Núñez, and F. Zaïdi. A passive testing approach based on invariants: Application to the WAP. *Computer Networks*, 48(2):247–266, 2005.

[5] A. Benharref, R. Dssouli, M. Serhani, and R. Glitho. Efficient traces' collection mechanisms for passive testing of web services. *Information & Software Technology*, 51(2):362–374, 2009.

[6] A. Cavalli, C. Gervy, and S. Prokopenko. New approaches for passive testing using an extended finite state machine specification. *Information and Software Technology*, 45:837–852, 2003.

[7] E. Frank, M. Hall, G. Holmes, R. Kirkby, and B. Pfahringer. Weka - a machine learning workbench for data mining. In O. Maimon and L. Rokach, editors, *The Data Mining and Knowledge Discovery Handbook*, pages 1305–1314. Springer, 2005.

[8] R.M. Hierons, K. Bogdanov, J.P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A.J.H Simons, S. Vilkomir, M.R. Woodward, and H. Zedan. Using formal methods to support testing. *ACM Computing Surveys*, 41(2), 2009.

[9] R.M. Hierons, J.P. Bowen, and M. Harman, editors. *Formal Methods and Testing, LNCS 4949*. Springer, 2008.

[10] D. Lee, A.N. Netravali, K.K. Sabnani, B. Sugla, and A. John. Passive testing and applications to network management. In *5th IEEE Int. Conf. on Network Protocols, ICNP'97*, pages 113–122. IEEE Computer Society Press, 1997.

[11] T. Mielikäinen. Frequency-based views to pattern collections. *Discrete Applied Mathematics*, 154(7):1113–1139, 2006.

[12] R. E. Miller, D. Chen, D. Lee, and R. Hao. Coping with nondeterminism in network protocol testing. In *17th Int. Conf. on Testing of Communicating Systems, TestCom'05, LNCS 3502*, pages 129–145. Springer, 2005.

[13] G.J. Myers. *The Art of Software Testing*. John Wiley and Sons, 2nd edition, 2004.

[14] M. Tabourier and A. Cavalli. Passive testing and application to the GSM-MAP protocol. *Information and Software Technology*, 41:813–821, 1999.

[15] D. Taniar and J. Goh. On mining movement pattern from mobile users. *International Journal of Distributed Sensor Networks*, 3(1):69–86, 2007.