From Data Mining to User Models in Evolutionary Databases*

César Andrés, Manuel Núñez, and Yaofeng Zhang

Departamento Sistemas Informáticos y Computación
Universidad Complutense de Madrid, E28040 Madrid, Spain
`c.andres@fdi.ucm.es, mn@sip.ucm.es, yaofeng@fdi.ucm.es`

Abstract. Testing is one of the most widely used techniques to increase the quality and reliability of complex software systems. In this paper we refine our previous work on passive testing with invariants to incorporate (probabilistic) knowledge obtained from users of the system under test by using data mining techniques. We adapt our previous approach in order to reduce costs. We emphasize the use of data mining to discover knowledge, a technique that is not only accurate but also comprehensible for the user.

1 Introduction

Testing is a major issue in any software development project because faults are indeed difficult to predict and find in real systems. In testing, there is usually a distinction between two approaches: *Passive* and *Active*. The main difference between them is whether a tester can interact with the System Under Test (SUT). If the tester can interact with the SUT, then we are in the active testing paradigm. On the contrary, if the tester simply monitors the behavior of the SUT, then we are in the passive testing paradigm. Actually, it is very frequent either that the tester is unable to interact with the SUT or that the internal nondeterminism of the system makes difficult to interact with it. In particular, such interaction can be difficult in the case of large systems working 24/7 since it might produce a wrong behavior of the system.

Therefore, passive testing usually consists in recording the trace produced by the SUT and trying to find a *fault* by comparing this trace with the specification [10,12,11,13,6,9]. A new methodology to perform passive testing was presented in [7,5]. The main novelty is that a set of *invariants* is used to represent the most relevant expected properties of the specification. An invariant can be seen as a restriction over the traces allowed to the SUT. We have recently extended this work to consider the possibility of adding time constraints as properties that traces extracted from the SUT must hold [2].

* Research partially supported by the Spanish MEC projects WEST/FAST (TIN2006-15578-C02-01) and TESIS (TIN2009-14312-C02-01), and UCM-BSCH programme to fund research groups (GR58/08 - group number 910606).

We can consider different methods to obtain a set of invariants. The first one is that testers propose 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, another option is to assume that the set of invariants provided by the testers are correct by definition. In this paper we consider an alternative method: We automatically derive invariants from the specification. This approach was introduced in [4] and improved in [3]. Initially, we consider an adaptation of the algorithm presented in [7]. 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 any criteria to decide which are *the best ones*. Therefore, we improve our contribution by using *data mining* techniques to provide a probabilistic model [1] to guide the extraction of these properties. The underlying idea was that invariants should check the most frequent user actions. We provided a methodology to extract from a specification, by using its associated probabilistic model, a set of invariants with a relevance degree. In [3] it was showed that *the cost* of building a user model from a database is big. Thus, we are not able to build a new user model each time that new data is added.

In this paper we study when a user model, extracted from an original database, continues being valid with respect to a timed evolution of the original one. That is, we study the *conformance* of a probabilistic user model with respect to a temporal evolution of the initial database. With this methodology we guarantee that the set of extracted invariants using the algorithm in [3] remains with a high degree of relevance. In addition, we study a feasible approach to update *on the fly* the user model in order to save time and computational resources.

The rest of the paper is structured as follows. In Section 2 we present our passive testing framework of timed systems. In Section 3 we present our formalism to define user models. In Section 4 we define the conformance of a user model with respect to a database and introduce a dynamic adaptation taking into account the new recorded data. We conclude with Section 5 where we present our conclusions.

2 Testing Framework

In this section we review our framework [2]. We adapt the well known formalism of Finite State Machines (`FSMs`) to model our specifications. The main difference with respect to usual `FSMs` consists in the addition of *time* to indicate the lapse between offering an input and receiving an output. We use positive real numbers as time domain for representing the time units that the output actions take to be executed.

Definition 1. A *Timed Finite State Machine* is a tuple $M = (\mathcal{S}, s_0, \mathcal{I}, \mathcal{O}, \mathcal{T})$, where $\mathcal{S}$ is a finite set of states, $s_0 \in S$ is the initial state, $\mathcal{I}$ and $\mathcal{O}$, with $\mathcal{I} \cap \mathcal{O} = \emptyset$, are the finite sets of *input* and *output* actions, respectively, and

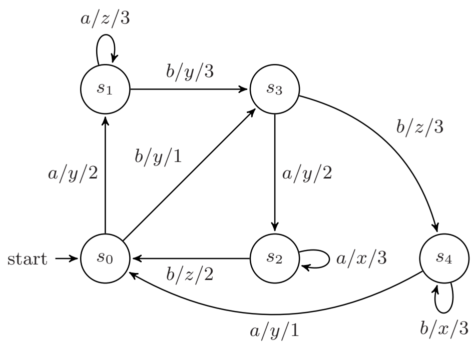

Fig. 1. Example of `TFSM`

$\mathcal{T} \subseteq \mathcal{S} \times \mathcal{I} \times \mathcal{O} \times \mathbb{R}_+ \times \mathcal{S}$ is the set of transitions. Along this paper we will use $s \xrightarrow{i/o}_t s'$ as a shorthand to represent the transition $(s, i, o, t, s') \in \mathcal{T}$.

We say that M is *deterministic* if for all state s and input i there exists at most one transition labeled by i departing from s. We say that M is *input-enabled* if for all state $s \in \mathcal{S}$ and $i \in \mathcal{I}$ there exist $s' \in \mathcal{S}$, $o \in \mathcal{O}$ and $t \in \mathbb{R}_+$ such that $s \xrightarrow{i/o}_t s'$. □

A transition belonging to $\mathcal{T}$ is a tuple (s, i, o, t, s') where $s, s' \in \mathcal{S}$ are the initial and final states of the transition, $i \in \mathcal{I}$ and $o \in \mathcal{O}$ are the input and output actions, respectively, and $t \in \mathbb{R}_+$ denotes the time that the transition needs to be completed. In this paper we consider that all defined `TFSMs` are input-enabled and deterministic.

Example 1. Let us consider the `TFSM` depicted in Figure 1. The set of inputs is $\{a, b\}$ while the set of outputs is $\{x, y, z\}$. For example, let us consider the transition $s_0 \xrightarrow{a/y}_2 s_1$ denotes that if the system is in state s_0 and the input a is received, then the output y will appear after 2 time units and the system moves to s_1. Let us also remark that this `TFSM` is input-enabled, that is, for all states we have that all inputs belonging to the set of inputs are defined. □

Next we introduce the notion of *trace*. A trace represents a sequence of actions that the system may perform from any state.

Definition 2. Let M be a `TFSM`. We say that $\omega = \langle i_1/o_1/t_1, \ldots, i_n/o_n/t_n \rangle$ is a *trace* of M, if there is a sequence of transitions such that

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

□

In the previous definition let us mention that we do not force traces to start in the initial state.

Example 2. Let us consider the TFSM defined in Figure 1. Some traces of this TFSM are $\langle a/x/3, b/z/2\rangle$ (traversing the states s_2, s_2, s_0) and $\langle b/z/3, b/x/3\rangle$ (traversing the states s_3, s_4, s_4) but not $\langle a/x/3, b/x/2\rangle$. $\square$

Next we introduce the notion of *invariant.* An invariant allows us to express properties that must be fulfilled by the SUT. For example, we can express that the time the system takes to perform a transition always belongs to a specific interval. In our framework, an invariant expresses the fact that each time the SUT performs a given sequence of actions, it must exhibit a behavior reflected in the invariant. Invariants can be seen as a kind of temporal logic that expresses both timed and untimed properties of the specification, modeled by a TFSM. Untimed properties are represented by the sequences of input/output pairs that conform the invariant while timed properties are given by the intervals that indicate when these actions must be performed. Next, we present the intuitive idea of invariants by using examples.

Example 3. Let us consider the specification presented in Figure 1. The invariant $\varphi = a/y/[2,3], a \mapsto \{x, z\}/[2.5, 3.5] \rhd [4, 7]$ means that every time that we observe the input a followed by the output y within a timed value belonging to $[2, 3]$ then, if we observe the input a then we have to observe either the output x or the output z in a time belonging to $[2.5, 3.5]$. The last time restriction makes reference to the whole length of the checked log, that is the elapsed time from the input action a to the last output of the invariant, must belong to $[4, 7]$. $\square$

3 User Model

As we discussed in the introduction, there are several ways to obtain a set of invariants. In this paper we propose that invariants can be automatically derived from the specification. The main problem to perform this approach is that the set of correct extracted invariants is huge and potentially infinite. Thus, we need a methodology to select *good* invariants among the set of candidates.

A first approach was given in [3] . In that paper we assume that there exists a big set of recorded logs into a database, which corresponds to the set of user interactions with the system. In order to process this information, a probabilistic model can be defined to represent the users of the system. This new model is called *user model* [1], and it can be automatically extracted from the database by applying *data mining techniques.*

Intuitively, a user model includes the probability that a user chooses each input in each situation. We use a particular case of *Probabilistic Machine* (PM) to represent user models. Let us note that the interaction of a user with a SUT ends whenever the user decides to stop it. Consequently, models denoting users must represent this event as well. Therefore, given a PM representing a user model, we will require that the sum of the probabilities of all inputs associated to a state is lower than or equal to 1. The remainder up to 1 represents the probability of stopping the interaction at this state.

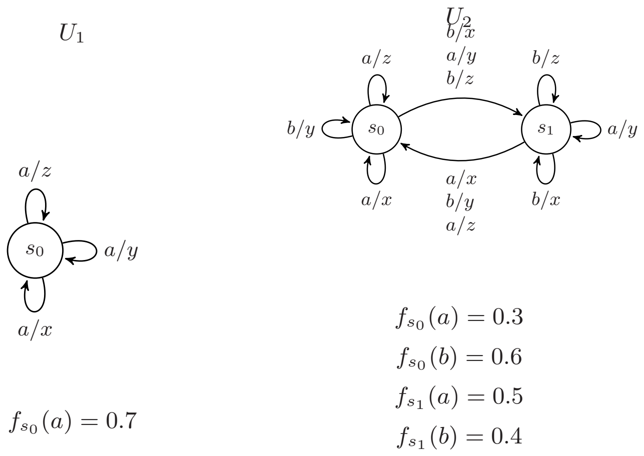

Fig. 2. Example of user models

Definition 3. A *user model* is represented by a tuple $U = (\mathcal{S}, s_0, \mathcal{I}, \mathcal{O}, F, T)$ where $\mathcal{S}$ is the finite set of states and $s_0 \in \mathcal{S}$ is the initial state. $\mathcal{I}$ and $\mathcal{O}$, with $\mathcal{I} \cap \mathcal{O} = \emptyset$, are the finite sets of *input* and *output* actions, respectively. $F = \{f_s | s \in S\}$ is a set of probabilistic functions such that for all state $s \in \mathcal{S}$, $f_s : \mathcal{I} \mapsto [0,1]$ fulfills that $\sum_{i \in \mathcal{I}} f_s(i) \leq 1$. Finally, $T \subseteq \mathcal{S} \times \mathcal{I} \times \mathcal{O} \times \mathcal{S}$ is the set of transitions.

We say that U is *deterministically observable* if there do not exist in T two transitions $(s, i, o_1, s_1), (s, i, o_2, s_2) \in T$ with $o_1 = o_2$ and $s_1 \neq s_2$. □

In this paper, we assume that user models are deterministically observable. We adopt $s \xrightarrow{i/o}_p s'$ as a shorthand to express $(s, i, o, s') \in T$ and $p = f_s(i)$.

Example 4. Let us consider the user models depicted in Figure 2. Both models represent a type of user that interact with the TFSM presented in Figure 1. The user model U_1 represents users who always are interacting with the SUT by applying the a input action. The probability to apply this input is represented by the function $f_{s_0}(a)$. The remainder to 1, that is 0.3, represents the probability to stop in this state.

U_2 represents a complex kind of users. These users can perform any input at any time. The probability associated with each state and input is represented by the associated functions $f_{s_0}(a)$, $f_{s_0}(b)$, $f_{s_1}(a)$, and $f_{s_1}(b)$, respectively. □

Next, we identify the probabilistic traces of user models. These traces will be used in order to provide a coverage degree of a possible user behavior.

Definition 4. Let $\mathcal{I}$ be a set of input actions and $\mathcal{O}$ be a set of output actions. A *probabilistic trace* is a sequence $\langle (i_1/o_1, p_1), (i_2/o_2, p_2), \ldots, (i_n, o_n, p_n) \rangle$, with $n \geq 0$, such that for all $1 \leq k \leq n$ we have $i_k \in \mathcal{I}$, $o_k \in \mathcal{O}$, and $p_k \in [0,1]$.

Let U be a user model and $\sigma = \langle (i_1/o_1, p_1), (i_2/o_2, p_2), \ldots, (i_n/o_n, p_n) \rangle$ be a probabilistic trace. We say that σ is a trace of U, denoted by $\sigma \in \mathtt{ptr}(U)$, if there is a sequence of transitions of U

$$s_0 \xrightarrow{i_1/o_1} s_1 \xrightarrow{i_2/o_2} s_2 \cdots \xrightarrow{i_n/o_n} s_n$$

and for all $1 \leq k \leq n$ we have $p_k = f_{s_{k-1}}(i_k)$. We define the *stopping probability of* U *after* σ as

$$\mathtt{s}(U, \sigma) = 1 - \sum \{ f_{s_n}(i) \mid i \in \mathcal{I} \}$$

We denote the *complete probability* of σ as

$$\mathtt{prob}(U, \sigma) = \left(\prod_{1 \leq j \leq n} p_j \right) \cdot \mathtt{s}(U, \sigma)$$

□

Let us remark that the set of probabilistic traces of a user model is related to the database of logs, that has been used to generate this model. The probability of finding a log in the database has to be equivalent to the probability to produce this sequence in the user model, and stop.

Example 5. Let us consider the user models U_1 and U_2 presented in Figure 2. Let DB_1 and DB_2 be the two databases from where these models where extracted. Next we present the relation between finding a log in a database and the probability to generate this sequence from a user model.

For example, we calculate the probability of finding the log $\langle a/y/2 \rangle$ in DB_1. It has to be the same value as $\mathtt{prob}(U_1, \langle a/y/0.7 \rangle) = 0.21$ computed as $0.7 \cdot 0.3$. Another example is to calculate the probability of finding the log $\langle a/y/2, a/x/3 \rangle$. In this case we have $\mathtt{prob}(U_1, \langle a/y/0.7, a/y/0.7 \rangle) = 0.147$, since $0.7 \cdot 0.7 \cdot 0.3$. Let us remark that the probability of finding a log into DB_1 can be 0. For example, this happens if we look for the log $\langle a/y/2, b/y/3 \rangle$.

We have that the probability of finding any log in the user model U_2 (extracted from DB_2) is always bigger than 0. For example, if we use the previous log $\langle a/y/2, b/y/3 \rangle$ we have that, its frequency in DB_2 has to be the same as the probability $\mathtt{prob}(U_2, \langle a/y/0.5, b/y/0.4 \rangle) = 0.02$, computed as $0.5 \cdot 0.4 \cdot 0.1$. □

We use user models to guide the algorithm of extracting invariants from the specification. Let us note that if we generate invariants that reflect only a finite set of input sequences, then we can calculate the proportion of covered interactions between the user and the `SUT`. In terms of *number of interactions*, this proportion is 0 because we are considering a finite number of cases out of an infinite set of possibilities. However, the *probability* that the user interacts with the `SUT` and performs one of the traces considered in a given finite set of cases is not 0 in general. Thus, we may use this value as a *coverage measure* of a finite set of invariants.

4 Comparation and Evolution of User Models

The use of user models to guide the algorithm of invariant extraction from the specification is feasible. The problem of this approach is that building by applying data mining techniques is a *hard* task, that is, we need a lot of resources of the database and of computation to generate it. This implies that the extraction of the set of invariants continues being very expensive. Our goal is to reduce the number of times that a user model should be rebuilt. Also, we provide an adaptive method based on the social insects algorithms, called Ant Colony Optimization [8], to update the model.

4.1 Comparation of User Models

We present a measure function to decide how closed are the users from two databases. The idea is that databases are changing along the time. We can assume that a user model is correct whit respect to a database when the users from this database and the users from the original database, that is the one used to generate the user model, are similar. When this measure is higher than a certain bound, then we should drop the oldest user model and build a new one.

In this approach we propose that users that use the `SUT` may be structured in different classes. These classes denote the degree of experience that this user has. In a database composed by logs recorded from the `SUT`, the distribution of different users among classes depend on the ammount of traces that belongs to the database, the number of different days that these users interact with the `SUT`, and the distance (in time) between the first and the last interaction. Each class is a set of sorted users (according to their relevance). So, the set with the biggest relevance, that is *experts* users, is u_1, the set labeled with u_2 represents people that normally use the system, and so on. As it usually happens in knowledge communities, the amounts of users in each class should follow a pyramidal structure. Thus, the condition $\mid u_1 \mid \; < \; \mid u_2 \mid \; < \ldots < \; \mid u_n \mid$ will be assumed. It is important to remark that the distribution of users among classes is dynamically done by taking into account their behavior. However, when creating a hierarchy for the first time there is no information about the previous behaviour of the users.

Next, we relate the conformance of a user model, the database used to generate this user model, and a possible timed evolution of this database. The main parameter to be dynamically studied is the *deformation speed of the log*, that is, the speed at which the database is increasing its number of recorded fields, and the users start to change from one class to another.

Definition 5. Let U be a user models and Let DB_1, DB_2 be two databases, where DB_1 is the database used to produce U. We denote by $u_i(DB_j)$ the set of users of the database DB_j belonging to the class u_i, and by $U(DB_j)$ the total amount of users in DB_j.

We define the *correlation measure* between DB_1 and DB_2 as:

$$\mathcal{P}(DB_1, DB_2) = \sum_{i=1}^{n} \frac{(u_i(DB_1) - u_i(DB_2))^2}{U(DB_1)^2 + U(DB_2)^2}$$

□

With respect to the timed evolution of the database we find, on the one hand, if an interaction from a non yet classified users is stored into the database, then this user is classified into the level u_n. On the other hand, if the interaction is of an existing user, then the database can change her associated class, for example, due to being less/more participative. Let us remark that given two databases DB_1 and DB_2, we have that $\mathcal{P}(DB_1, DB_2)$ value belongs to $[0, 1]$.

Definition 6. Let U_1 be a user model generated from the database DB_1. Let DB_2 be a timed evolution of DB_1. We say that U_1 conforms DB_2 with a degree of confidence $\alpha \in [0, 1]$ if:

$$1 - \mathcal{P}(DB_1, DB_2) \geq \alpha$$

□

By requesting a higher value of α we ensure that the user model is more conforming with respect to the timed evolution database. This value can be also used to denote when the tester should generate a new set of invariants. For example, we can consider that when the degree of confidence of the current database with respect to a user model is less than 0.7, then we generate a new user model from this database, and we generate a new set of invariants using this new user model.

4.2 Dynamic Adaptation of a User Model

By using the previous correlation measure of the databases, testers know when a set of invariants is not valid with respect to a database. However, they still have the problem of the cost extraction algorithm of a new user model from the new database. In this section we present a dynamic adaptation of a user model with respect to the incoming new data into the database. This method provides a *low cost* and *dynamic* methodology that allows to always have an updated user model. For this approach, we assume that we are provided with an initial model. This one is defined as a graph where each path (and sub-path), by means of a probabilistic trace, has associated a *live counter*. This counter at the beginning contains the complete probability value associated with this path (see Definition 4). When a new user interaction is recorded into the database, this interaction is *recorded* also into the user model, that is, the live counter of this path increases while the value of the rest of paths decreases. By using this technique, a user model can obtain several possible evolutions. The most representative evolutions are used to build new paths in the user model and to remove old paths.

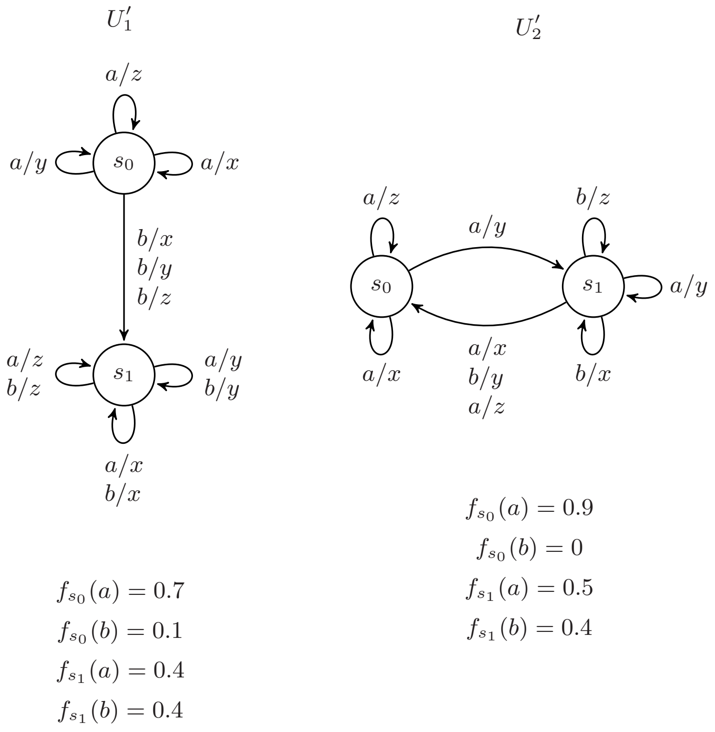

Fig. 3. Possible adaptation of the user models U_1 and U_2

Let us present the intuitive idea of adding new paths to the user U_1 presented in Figure 2. Let us consider that the database receives several user interactions. After performing the first b input action, we have that the probability of continuing with either the a action, or the b action is the same. A possible adaptation of U_1 with these conditions is presented as the model U_1' in Figure 3.

Next, let us present the idea of dropping paths. Let us consider that we are working with the second user model U_2 (see Figure 2) and the database receives several sequences that always start with the input action a. After processing these new logs, a possible adaptation of the model could be the one presented in U_2' (see Figure 3). The main difference is that users modeled by U_2' will never start with the input action b.

5 Conclussions

In this paper we have presented an improvement of a previous methodology to perform passive testing by using invariants. These invariants are automatically extracted from a specification helped with a user model that represent the usual behaviours of users interacting with the `SUT`. By using data mining techniques we can generate a user model but the cost of building it is very high. In this paper we present, on the one hand, an approach to decide when a user model continues being useful with respect to a database and, on the other hand, a dynamic algorithm to adapt a given user model with respect to new recorded data of the database.

References

1. Andrés, C., Llana, L., Rodríguez, I.: Formally transforming user-model testing problems into implementer-model testing problems and viceversa. Journal of Logic and Algebraic Programming 78(6), 425–453 (2009)
2. Andrés, C., Merayo, M.G., Núñez, M.: Passive testing of timed systems. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 418–427. Springer, Heidelberg (2008)
3. Andrés, C., Merayo, M.G., Núñez, M.: Supporting the extraction of timed properties for passive testing by using probabilistic user models. In: 9th Int. Conf. on Quality Software, QSIC 2009 (2009) (in press)
4. Andrés, C., Merayo, M.G., Núñez, M.: Using a mining frequency patterns model to automate passive testing of real-time systems. In: 21st Int. Conf. on Software Engineering & Knowledge Engineering, SEKE 2009, Knowledge Systems Institute, pp. 426–431 (2009)
5. Bayse, E., Cavalli, A., Núñez, M., Zaïdi, F.: A passive testing approach based on invariants: Application to the WAP. Computer Networks 48(2), 247–266 (2005)
6. Benharref, A., Dssouli, R., Serhani, M., Glitho, R.: Efficient traces' collection mechanisms for passive testing of web services. Information & Software Technology 51(2), 362–374 (2009)
7. Cavalli, A., Gervy, C., Prokopenko, S.: New approaches for passive testing using an extended finite state machine specification. Information and Software Technology 45(12), 837–852 (2003)
8. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man and Cybernetics B 26(1), 29–41 (1996)
9. Hierons, R.M., Bogdanov, K., Bowen, J.P., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., Luettgen, G., Simons, A.J.H., Vilkomir, S., Woodward, M.R., Zedan, H.: Using formal methods to support testing. ACM Computing Surveys 41(2) (2009)
10. Lee, D., Netravali, A.N., Sabnani, K.K., Sugla, B., John, A.: Passive testing and applications to network management. In: 5th IEEE Int. Conf. on Network Protocols, ICNP 1997, pp. 113–122. IEEE Computer Society Press, Los Alamitos (1997)
11. Petrenko, A.: Fault model-driven test derivation from finite state models: Annotated bibliography. In: Cassez, F., Jard, C., Rozoy, B., Dermot, M. (eds.) MOVEP 2000. LNCS, vol. 2067, pp. 196–205. Springer, Heidelberg (2001)
12. Tabourier, M., Cavalli, A.: Passive testing and application to the GSM-MAP protocol. Information and Software Technology 41(11-12), 813–821 (1999)
13. Ural, H., Xu, Z.: An EFSM-based passive fault detection approach. In: Petrenko, A., Veanes, M., Tretmans, J., Grieskamp, W. (eds.) TestCom/FATES 2007. LNCS, vol. 4581, pp. 335–350. Springer, Heidelberg (2007)