Timed Test Cases Generation based on MSC-2000 Test Purposes

Abdeslam En-Nouaary and Gang Liu
Department of Electrical and Computer Engineering, Concordia University
1455 de Maisonneuve W., Montréal, Québec H3G 1M8, Canada
{ennouaar,g_liu}@ece.concordia.ca

Abstract

This paper addresses timed test cases generation using test purposes given as Message Sequence Charts (MSCs). A test purpose is a partial behaviour of the system under test. Test purposes are very important in testing because they help reduce the number of test cases while guaranteeing acceptable faults coverage. The adoption of MSCs as a model for test purposes is very important because they have a graphical representation that can be used to express clearly what the user wants to test. The approach presented in this paper is fundamentally based on timed input output automata (TIOA) theory. An example is used to illustrate the concepts and the steps of the approach presented in this paper.

1. Introduction

Testing is one of the most important and crucial activities of software life cycles. The objective of testing is to make sure that the functionalities of a system are correctly implemented. This is generally done in three steps. First of all, test cases are generated from the specification of the system and/or test purposes. Then, the generated test cases are applied to the implementation of the system under test (IUT for short) and the reactions of IUT are observed. Finally, the test results are analyzed and a verdict is concluded: if the outputs of each test case match those expected (i.e., derived from the specification) the implementation is said to be conform to the specification; otherwise, the implementation is faulty and the diagnosis process is started to locate and fix the fault.

Over the past three decades, a lot of works have been done in testing. Particularly, many algorithms have been developed to generate test cases from different formal models such as (extended) finite states machines. Unfortunately, these works cannot be used to test timed specifications because the models on which they are based lack time expressiveness. So, many researchers, over the last past decade, have been investigating timed testing with different backgrounds and different formal timed models (see for instance [2, 4, 5, 6, 12, 13, 14, 15, 16, 17]). Although the proposed methodologies are successful in testing timed specification with different faults coverage, most of them suffer from the state explosion problem and an exorbitant number of test cases. That is one of the reasons why the search for a new approach for timed testing is still needed.

This paper addresses timed test cases generation using test purposes given as Message Sequence Charts (MSCs). A test purpose is a partial behaviour of the system under test [2, 8, 13, 15]. Test purposes are very important in testing because they help reduce the number of test cases while guaranteeing acceptable faults coverage. The adoption of MSCs as a model for test purposes is very important because MSCs have a graphical representation that helps a user clearly express what he/she wants to test. Moreover, MSCs are widely used in the telecommunication industry and so one can easily see the application that the proposed approach might have. In this paper, we will introduce a methodology for timed test cases generation based on MSC-2000. It should be recalled that MSC-2000 provides designers and testers with constructs to express timing behaviour by not only timers but also time constraints between any pair of events appearing on MSC diagrams [1, 4, 7].

The rest of this paper is organized as follows. Section 2 discusses the issues to be investigated in timed testing. Section 3 presents our approach for timed test cases generation. Section 4 concludes the paper and presents future work.

2. Issues to be investigated in timed testing

The main problem faced when testing against timed specification is the existence of time constraints in the specification of the system to be tested. The generated test cases may not be executable due to the existence of the clock conditions associated with the transitions in the specification. Moreover, test execution consists of not only observing the outputs and verifying the target states of the transitions but also checking that the implementation of the system under test does not accept the inputs and does not respond with the outputs outside the interval fixed in the specification of the system. This is difficult because the time is not under the direct control of the tester. Moreover, this difficulty varies with the timed model used and the time semantics adopted. For example, discrete timed models are much easier to test than dense/continuous timed models. Indeed, discrete time models can easily be transformed into a FSM by introducing a special time event “tick” that represents the progression of time from t to t+1. However, under dense time models the time values are real numbers and hence infinite.

In general, a certain number of issues should be investigated when testing timed specifications. The first issue is the executability of test cases. This means that the time constraint of each transition traversed by a test case should be satisfied by the values of clocks during the execution of the test case. In this paper, the specification is modeled as a TIOA and test cases are generated not directly from TIOA but from an automaton that represents the executions of

TIOA. Such an automaton, called the grid automaton, is automatically constructed by the tool implementing our approach.

The second issue to be considered is the fault model. It refers to the set of potential faults that can be encountered in an implementation of a timed specification. The fault model is much related to the specification model used to describe the behaviour of the system under test. In the case of timed systems, the fault model consists of four types of faults: the output faults, the transfer faults, the time constraints widening faults, and the time constraints narrowing faults. The fault model is very important because it helps in developing efficient methods. In this paper, we will not talk in more details about fault models. However, the interested reader is invited to consult the already published works on timed fault models [18].

The third issue to investigate is the fault coverage of test cases. This refers to the power of a test cases generation method to detect the potential faults in the implementation of the system under test (i.e., the faults listed in the fault model). Therefore, test cases generation methods can be compared based on their fault coverage. A method A is said more efficient than a method B if A detects more faults than B. However, for a more accurate comparison other criteria, such as the length of test suite, should be taken into account. For the sake of space, a comparison between existing timed test cases generation methods will be left to a future paper.

The fourth issue is the conformance relation between the implementation and its specification. The conformance relation is a very important aspect in testing because it gives the meaning of the conformance between an implementation and its specification. Indeed, a conformance relation is a mathematical relation between the implementation and its specification. To be able to formalize such a relation, we always assume that both the specification and the implementation are described in the same formal model. This paper is based on test purpose, so the idea is to check whether or not the IUT includes the behaviour expressed by the test purpose. In other words, the formalization of conformance relation is not the main concern in this paper. Notice that the infinity/density of time domain makes it difficult to choose feasible conformance relation.

Now that we presented the problems related to timed testing, we will introduce in the next section our approach for timed test cases generation. Such an approach solves the executability problem and guarantees acceptable fault coverage.

3. Our approach for Timed Test Cases Generation

Our approach for timed test cases generation is fundamentally based on TIOA and MSC. TIOA is used to describe the specification of the system while MSC2000 is used to specify the test purpose of the user. MSC2000 provides the user with constructs to specify the timing behaviour of a real-time system by not only timers but also with time constraints between any pair of events. The formal definition of TIOA and test purpose can be stated as follows.

Definition 1: Timed Input Output Automaton

A Timed Input Output Automaton (TIOA) A is a tuple (I_A, O_A, L_A, $l^0{}_{\_A}$, C_A, T_A) [2, 9, 14], where:

– *I_A is a finite set of input actions. Each input action begins with "?".*
– *O_A is a finite set of output actions. Each output action begins with "!".*
– *L_A is a finite set of location. The term "location" is chosen instead of the term "state" because the latter is used to define the operational semantics of the TIOA.*
– *$l^0{}_{\_A} \in L_A$ is the initial location.*
– *C_A is a finite set of synchronous clocks set to zero in $l^0{}_A$. We assume that the time is dense, which means that the clocks values are real numbers.*
– *T_A is the set of transitions. Each transition consists of a source location, an input or an output action, a clock guard that should hold in order to execute the transition, a set of clocks to be reset when the transition is executed, and a destination location. We assume that the transitions are instantaneous.*

Figure 1 is an example of TIOA that describes the behaviour of a simple multimedia system. The system receives an image and its sound within two time-units, sends an acknowledgment in less than five time-units after the reception of the image, and then sends the message *reset* and starts waiting for another image. If the time constraints are not satisfied, the system issues the message *error* and goes back to its initial state. The TIOA that describes the system has four locations *l0* (the initial location), *l1*, *l2*, and *l3*, six transitions and two clocks *x* and *y*. The transition from *l0* to *l1*, denoted by $l0 \xrightarrow{?image, x<=2, x:=0, y:=0} l1$, is executed when the system receives the message *image* and the value of clock *x* is less than or equal to *2*. When the transition is fired, the clocks *x* and *y* are set to *0*.

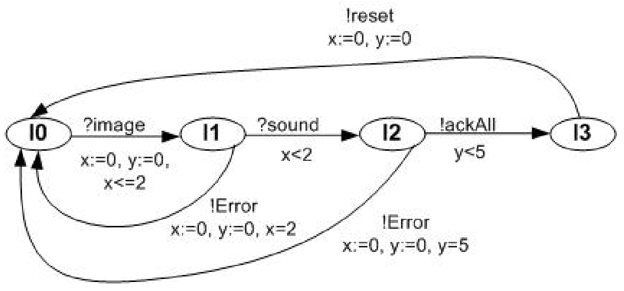


Figure 1: Specification of Multimedia System

Definition 2: Test Purpose

A Test purpose is a description of the property to be tested by a user. Such a property represents a sequence of interactions among the components of the system and with their environment as well as the time constraints on these interactions.

An example of test purpose in MSC 2000 is given in Figure 2. Here, the user is interested in checking if the implementation of the multimedia system can accept an image followed by its sound within one time-unit, and respond with an acknowledgment in less than two time-units after the reception of the sound. As one can easily see from this example of test purpose, the user is not interested in verifying all the functionalities of the system (i.e., the whole specification) but only a subset of them. The functionalities wanted by the user can be, for instance, the most critical functions of the system or the most frequently executed parts of the system.

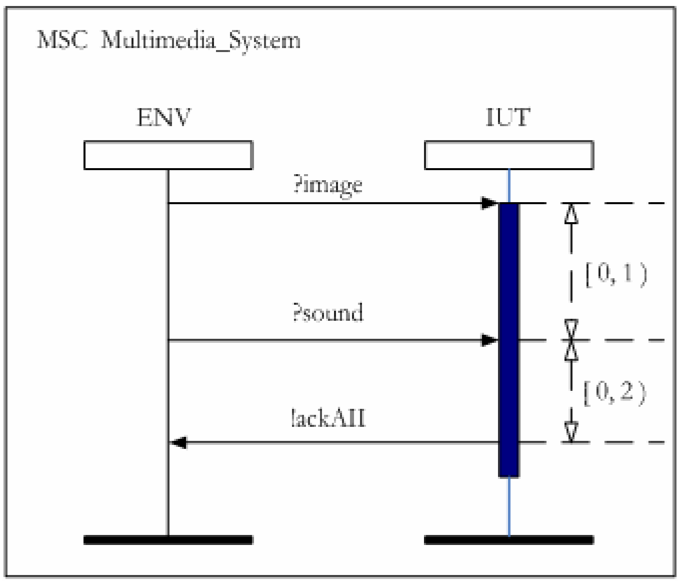


Figure 2: Test Purpose of Multimedia System

It should be mentioned that we use only a subset of MSC-2000 for the expression of test purposes. Indeed, we limited ourselves to Basic MSC (BMSC). Moreover, the BMSC used consists of two instances only (one for the IUT and the other one for the environment), and contains neither co-regions nor inline expressions. For time constraints, our approach takes into consideration the use of timers and the relative and absolute time intervals on the occurrence of events in test purpose. BMSC with more than two instances as well as HMSC and inline expressions are left to future work.

In order to generate test cases systematically from test purposes, we have to convert the MSCs of those test purposes into the model of the specification (i.e., TIOA). Such a conversion is done as follows. Each message received by the IUT in MSC is translated to an input action in TIOA, and each message sent by the IUT is translated to an output action; the state between each pair of exchanged messages is identified as the location in TIOA. The timer events and the time constraints in MSC can be described by the replacing clocks of TIOA. A critical problem of the resulting TIOA is that the number of clocks could be unnecessary larger than what we need. Therefore a process is needed to minimize the number of clocks for the resulting TIOA [3].

The TIOA model introduced above is an abstract model because it doesn't explicit all the possible executions of the described system. Therefore, test cases cannot be generated directly from TIOA specification. We need to use the executions of TIOA for such purpose. Such executions, called the operational semantics, can informally be stated as follows. The TIOA starts at its initial location with all clocks initialized to zero. Then, the values of clocks increase synchronously and measure the amount of time elapsed since the last initialization. At any time, the TIOA can make a transition if the values of clocks in the current location satisfy the clock guard of the transition. In this case, all the clocks in the reset set of the transition is initialized to zero and the TIOA moves to the destination location of the transition. The formalization of the operational semantics of TIOA is based on the concepts of state and clock valuation. A clock valuation v is a mapping from the set of clocks of TIOA to the set of real numbers (i.e., each clock is assigned a value, which is a real number). However, a state is couple (l, v) where l is a location in TIOA and v is a clock valuation.

Since we assume dense time model, one can easily see that the number of states of a TIOA is infinite. This is mainly due to the infinity of delay transitions upon the progression of time. Hence, it is impossible to generate test cases directly from the semantics of TIOA. To address the problem, we will not base our test cases generation method on the whole operational semantics of TIOA but on a subset of it, called Grid Automaton (GA) [2, 5, 10, 14]. The GA limits the delay transitions to a fixed time delay g, which is called the time granularity. The construction of GA is called the sampling of

TIOA. The definition of GA is as follows.

Definition 3: Grid Automaton

Let A = (I_A, O_A, L_A, l^0_A, C_A, T_A) be a TIOA. The Grid Automaton (GA) of A is a finite input output automaton GA = (I_{GA}, O_{GA}, S_{GA}, s^0_{GA}, T_{GA}), where:

- *$I_{GA} = I_A \cup \{g\}$, where g is a special time delay (g is a rational number).*
- *$O_{GA} = O_A$.*
- *S_{GA} is finite set of system states. Each state is a pair (l, v), where $l \in L_A$ and v is a clock valuation in which the value of each clock is a multiple of g.*
- *s^0_{GA} is the initial state that consists of the initial location , l^0_A with all clocks values set to 0.*
- *T_{GA} is a finite set of Transitions.*

Each transition consists of a source state, an action (input, output, or time delay *g*), and a destination state. There are two types of transitions in GA: the delay transitions on time delay *g* and the explicit transitions on input and output actions. Each state in GA has an outgoing delay transition on time delay *g*. However, a state *(l, v)* has an outgoing explicit transition on input or output action *a* if and only if there is a transition $l \xrightarrow{\{?,!\}a,G,\lambda} l'$ in A and *v* satisfies the clock guard *G*. After the execution of a delay transition on *g*, the value of each clock is incremented with *g* time-units. However, after the execution of an explicit transition the value of each clock in λ (i.e., the set of clocks to be reset by the transition) in A is zero.

Figure 3 shows the partial GA of the TIOA in Figure 1 sampled with a granularity *g = 0.5.*

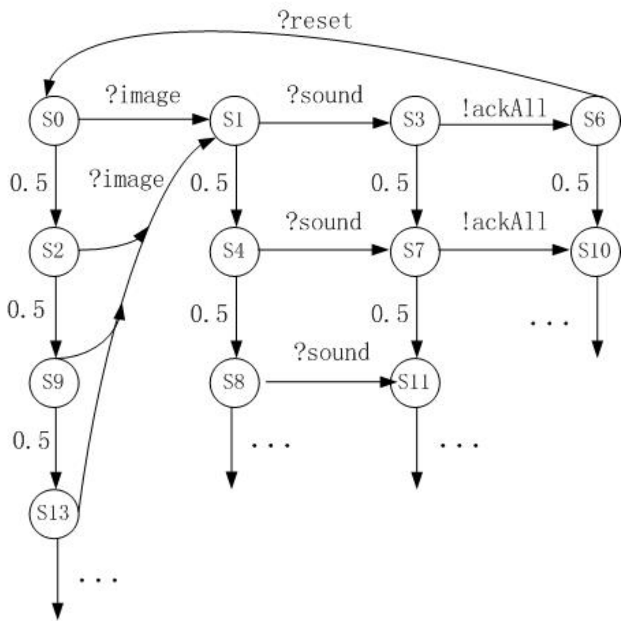


Figure 3: Grid Automaton of Specification

When testing is based on test purpose, it's likely that the property to be checked (i.e., the test purpose) is not a subset of the specification or is inconsistent with the specification. For this reason, we have to validate the test purpose against the specification, which is always assumed to be correct. To validate the test purpose against the specification of the system under test, we use a special composition of the test purpose and the specification. Such a composition is called synchronous product and is formally defined as follows.

Definition 4: Synchronous Product of Two TIOAs

Let $S = (I_S, O_S, L_S, l_S^0, C_S, T_S)$ and $T = (I_T, O_T, L_T, l_T^0, C_T, T_T)$ be two TIOAs. The synchronous product of *S* and *T* is a special composition $SP = (I_{SP}, O_{SP}, L_{SP}, l_{SP}^0, C_{SP}, T_{SP})$ of *S* and *T* such that [15]:

- $I_{SP} = I_S \cup I_T$ and $O_{SP} = O_S \cup O_T$.
- $L_{SP} \subseteq L_S \times L_T$.
- $l^0_S = (l^0_S, l^0_T)$.
- $C_{SP} = C_S \cup C_T$.
- L_{SP} and T_{SP} are the smallest relations defined by the following two rules:

– Suppose $l_{SP} = (l_S, l_T) \in L_{SP}$, $l_S \in L_S, l_T \in L_T, l_S^{'} \in L_S$, then

$$\left.\begin{array}{l} a \in I_S \cup O_S \\ a \notin I_T \cup O_T \\ l_S \xrightarrow{\{?,!\}a, G1, \lambda 1} l_S^{'} \in T_S \end{array}\right\} \Rightarrow l_{SP}^{'} = (l_S^{'}, l_T) \in L_{SP} \text{ and } l_{SP} \xrightarrow{\{?,!\}a, G1, \lambda 1} l_{SP}^{'} \in T_{SP}$$

– Suppose $l_{SP} = (l_S, l_T) \in L_{SP}$, $l_S \in L_S, l_T \in L_T, l_S^{'} \in L_S, l_T^{'} \in L_T$, then

$$\left.\begin{array}{l} a \in I_S \cup O_S \\ a \in I_T \cup O_T \\ l_S \xrightarrow{\{?,!\}a, G1, \lambda 1} l_S^{'} \in T_S \\ l_T \xrightarrow{\{?,!\}a, G2, \lambda 2} l_T^{'} \in T_T \end{array}\right\} \Rightarrow l_{SP}^{'} = (l_S^{'}, l_T^{'}) \in L_{SP} \qquad \text{and}$$

$$l_{SP} \xrightarrow{\{?,!\}a, G1\&G2, \lambda 1 \cup \lambda 2} l_{SP}^{'} \in T_{SP}$$

As an example, Figure 4 is the synchronous product of test purpose and specification TIOAs for the multimedia system of Figure 1 and Figure 2.

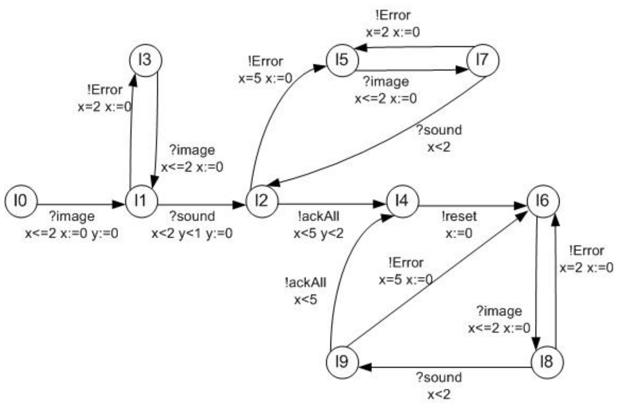


Figure 4: Synchronous product of specification and test purpose.

In addition to the synchronous product of two TIOAs, we also define the synchronization between two GAs as follows.

Definition 5: Synchronous Product of Two GAs

Let $S = (I_S, O_S, S_S, s_S^0, T_S)$ and $T = (I_T, O_T, S_T, s_T^0, T_T)$ be two TIOAs. The synchronous product of S and T is a special composition $SP = (I_{SP}, O_{SP}, S_{SP}, s_{SP}^0, T_{SP})$ of S and T such that:

- $I_{SP} = I_S \cup I_T$ and $O_{SP} = OS \cup O_T$.
- $L_{SP} \subseteq L_S \times L_T$.
- $l^0_S = (l^0_S , l^0_T)$.
- $C_{SP} = C_S \cup C_T$.
- L_{SP} and T_{SP} are the smallest relations defined by the following two rules:

- Suppose $s_{SP} = (s_S, s_T) \in S_{SP}$,
$s_S \in S_S, s_T \in S_T, s_S^{'} \in S_S$, then:

$$\left.\begin{array}{l} a \in I_S \cup O_S \\ a \notin I_T \cup O_T \\ s_S \xrightarrow{a} s_S^{'} \in T_S \end{array}\right\} \Rightarrow s_{SP}^{'} = (s_S^{'}, s_T) \in S_{SP} \text{ and } s_{SP} \xrightarrow{a} s_{SP}^{'} \in T_{SP}$$

– Suppose $s_{SP} = (s_S, s_T) \in S_{SP}, s_S \in S_S, s_T \in S_T, s_S^{'} \in S_S, s_T^{'} \in S_T$, then

$$\left.\begin{array}{l} a \in I_S \cup O_S \\ a \in I_T \cup O_T \\ s_S \xrightarrow{a} s_S^{'} \in T_S \\ s_T \xrightarrow{a} s_T^{'} \in T_T \end{array}\right\} \Rightarrow s_{SP}^{'} = (s_S^{'}, s_T^{'}) \in S_{SP} \text{ and } s_{SP} \xrightarrow{a} s_{SP}^{'} \in T_{SP}$$

Overall, our approach for timed test cases generation consists of four main phases:

- The conversion of test purpose into TIOA.
- The construction of a synchronous product.
- The sampling of the TIOA of test purpose and the TIOA of specification.
- The traversal of the resulting GA.

The conversion of test purpose into TIOA is simple and is explained at the beginning of this section. For the construction of the synchronous product, we distinguish between two operations: the construction of the synchronous product of the TIOA of the specification and the TIOA of the test purpose, and the construction of the synchronous product of the GA of the specification and the GA of the test purpose. The choice of which synchronous product to be constructed depends on the position of the sampling operation. Indeed, sampling can be done either before the construction of the synchronous product or after it.
In the first case, the TIOA of the specification and the TIOA of the test purpose are sampled to construct their respective GAs; then, the resulting GAs are synchronized following definition 5. In the second case, the TIOA of the specification and the TIOA of the test purpose are synchronized first according to definition 4; then the GA of the resulting TIOA is constructed.

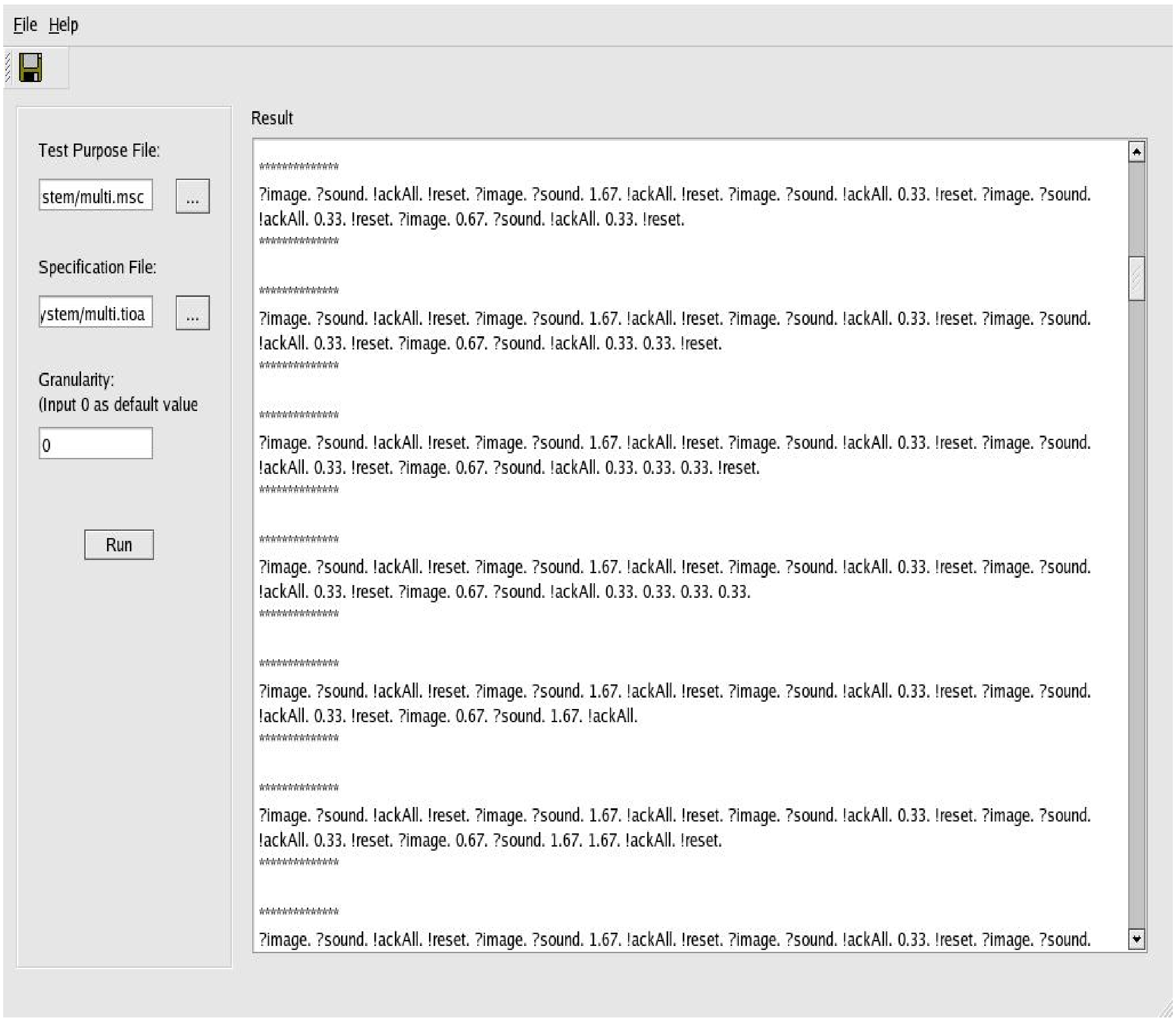
The position of sampling in the whole process is crucial because the sampling of a TIOA depends on the number of clocks used. Indeed, the granularity of sampling is $\frac{k}{(n+1)}$, where *k* is a non-null natural and *n* is the number of clocks in the TIOA to be sampled. One can easily see that when sampling is done after synchronous product the granularity is bigger and so is the number of states in the resulting GA. Consequently, the number of generated test cases is bigger than if sampling would have been done before synchronous product. This is true because the number of clocks in the synchronous product of two TIOAs is the sum of the number of clocks in each of them (see Figure 4). It should be mentioned that we implemented the two variants of sampling method in order to have quantitative measures on the number of test cases generated in each case. This will help us later, in a future work, in the assessment of fault coverage since the latter might be strongly related to the way sampling is done.

The traversal of the resulting GA is the last step in the whole process. It aims at the generation of the final test cases. We use a variant of depth-first traversal to generate timed test cases. The modification of the original depth-traversal algorithm is done depending on the test selection criterion to be used. In fact, we have different traversal strategies to generate test cases from the resulting GA. In particular, we can use all-pass-verdict test selection criteria to generate all test cases that lead to the verdict Pass, one-pass verdict test selection criteria to derive one test case that leads to the verdict Pass, all-fail-verdict test selection criteria to generate all test cases that lead to the verdict fail, etc. A verdict “Pass” is concluded if the IUT satisfies both the specification and the test purpose. However, a verdict “Fail” is concluded if the IUT does not satisfy the specification. Finally, a verdict “Inconclusive” is concluded if the IUT satisfies the specification but not the test purpose.

For the traversal of the GA based on all-pass-verdict selection criterion, we start at the initial state of the GA and we move downward until a leaf with pass-verdict is reached. This will result in a new test case after which we go one level up in the hierarchy and we try to generate another test case. The process stops when all pass-verdict paths are covered.

Note that each test case generated by our method consists of input actions, time delays, and output actions. Moreover, all test cases are executable and can be easily represented in TTCN by using ordinary timers only. For each time delay in a test case, we set a timer and we wait for its expiration before proceeding to the processing of the next event in the test case.

Figure 5 shows a simple GUI of our tool. On the inner windows of it appear some test cases for the specification of Figure 1 and the test purpose of Figure 2. Here, the granularity used is 0.33 and the test selection criterion used is all-pass-verdict.



```
***************
?image. ?sound. !ackAll. !reset. ?image. ?sound. 1.67. !ackAll. !reset. ?image. ?sound. !ackAll. 0.33. !reset. ?image. ?sound.
!ackAll. 0.33. !reset. ?image. 0.67. ?sound. !ackAll. 0.33. !reset.
***************

***************
?image. ?sound. !ackAll. !reset. ?image. ?sound. 1.67. !ackAll. !reset. ?image. ?sound. !ackAll. 0.33. !reset. ?image. ?sound.
!ackAll. 0.33. !reset. ?image. 0.67. ?sound. !ackAll. 0.33. 0.33. !reset.
***************

***************
?image. ?sound. !ackAll. !reset. ?image. ?sound. 1.67. !ackAll. !reset. ?image. ?sound. !ackAll. 0.33. !reset. ?image. ?sound.
!ackAll. 0.33. !reset. ?image. 0.67. ?sound. !ackAll. 0.33. 0.33. 0.33. !reset.
***************

***************
?image. ?sound. !ackAll. !reset. ?image. ?sound. 1.67. !ackAll. !reset. ?image. ?sound. !ackAll. 0.33. !reset. ?image. ?sound.
!ackAll. 0.33. !reset. ?image. 0.67. ?sound. !ackAll. 0.33. 0.33. 0.33. 0.33.
***************

***************
?image. ?sound. !ackAll. !reset. ?image. ?sound. 1.67. !ackAll. !reset. ?image. ?sound. !ackAll. 0.33. !reset. ?image. ?sound.
!ackAll. 0.33. !reset. ?image. 0.67. ?sound. 1.67. !ackAll.
***************

***************
?image. ?sound. !ackAll. !reset. ?image. ?sound. 1.67. !ackAll. !reset. ?image. ?sound. !ackAll. 0.33. !reset. ?image. ?sound.
!ackAll. 0.33. !reset. ?image. 0.67. ?sound. 1.67. 1.67. !ackAll. !reset.
***************

***************
?image. ?sound. !ackAll. !reset. ?image. ?sound. 1.67. !ackAll. !reset. ?image. ?sound. !ackAll. 0.33. !reset. ?image. ?sound.
```

Figure 5: A GUI Implementation and Sample of test cases generated

The test case "?image.0.67.?sound.!ackall.0.33.!reset"means that the tester submits to the implementation under test the input *?image, waits 0.67 time-units, submits the input ?sound, observes the output !ackall*, waits 0.33 time-units, and then should observe the output *!reset.*

5. Conclusion

We presented a methodology to generate test cases for real-time systems specified by TIOA. Our method is based on test purposes expressed as MSCs and so helps the tester clearly specify what he/she wants to test. We implemented our approach in a tool using C++ and applied it on different examples with different sizes (different locations, different clocks, different transitions, etc.). For most of the examples used, our tool successfully generates test cases with acceptable execution time.

We are currently working on the assessment of the fault coverage of our method and all the factors that influence it. We are also planning to extend our methodology to non-deterministic TIOA as well as TIOA extended with data variables.

References

[1] ITU-T. Message Sequence Chart (MSC). International Telecommunications Union, Telecommunications Standards Sector (ITU-T). Recommendation Z.120, 2001.

[2] Abdeslam En-Nouaary and Rachida Dssouli. A Guided Method for Testing Timed Input Output Automata. TestCom2003, France, 2003

[3] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In 1996 IEEE RTSS'96, Dec. 4-6, 1996, Washington, DC, USA.

[4] Paul Baker, Paul Bristow, et al. Automatic Generation of Conformance Tests From Message Sequence Charts. 3rd SAM Workshop, University of Wales, 2002.

[5] Abdeslam En-Nouaary, Rachida Dssouli, and Ferhat Khendek. Timed Wp-Method: Testing Real-time Systems. IEEE Transactions on Software Engineering, November 2002.

[6] B. Nielsen and A. Skou. Automated Test Generation from Timed Automata. Proc. Workshop Tools and Algorithms for the Construction and Analysis of Systems, Apr. 2001

[7] Philipp Lucas. Timed Semantics of Message Sequence Charts Based on Timed Automata. Workshop on Theory and Practice of Timed Systems (TPTS'02), Grenoble, April 2002.

[8] Robert Nahm. Conformance Testing Based on Formal Description Techniques and Message Sequence Charts. March, 1995.

[9] R. Alur and D. Dill. A Theory of Timed Automata. Theoretical Computer Science, 126:183-235, 1994.

[10] K.G. Larsen and W. Yi. Time Abstracted Bisimulation: Implicit Specification and Decidability. Proc. Math. Foundations of Programming Semantics (MFPS 9), April 2001.

[11] ITU-T, TTCN-2. The Tree and Tabular Combined Notation (TTCN). Conformance Testing Methodology and Framework, Part 3, Recommendation X.292, 1997.

[12] D. Clarke and I. Lee. Automatic Generation of Tests for Timing Constraints from Requirements. In Proceedings of the Third International Workshop on Object-Oriented Real-Time Dependable Systems, California, February 1997.

[13] Sebastien Salva, Eric Petitjean, and Hacene Fouchal. A Simple Approach to Testing Timed Systems. In Proceedings of the Workshop on Formal Approaches to Testing of Software (FATES'01), Aalborg, Denmark, August 2001.

[14] J. Springintveld, F. Vaadranger, and P. Dargenio. Testing Timed Automata. Theoretical computer science journal, 2001.

[15] R. Castanet, O. Kone, and P. Laurencot. On the Fly Test Case Generation for Real Time Protocols. IC3N'98, Bordeaux University, France, 1998.

[16] A. Khoumsi, M. Kalay, R. Dssouli, A. En-Nouaary, and L. Granger. An Approach for Testing Real-Time Protocols. TESTCOM, Aug./Sept. 2000.

[17] T. Higashino et al. Generating test cases for a timed I/O automaton model. In G. Csopaki, S. Dibuz, and K. Tarnay, eds, Proc. IFIP Int'l Work. Test. Communicat. Syst. (IWTCS). 1999. Budapest, Hungary: MA: Kluwer Academic.

[18] Abdeslam En-Nouaary, Ferhat Khendek and Rachida Dssouli: *Fault Coverage in Testing Real-Time Systems,* In the 6th International Conference on Real-Time Computing Systems and Applications (RTCSA), Hong Kong, December 1999.