Some Implications of MSC, SDL and TTCN Time Extensions for Computer-aided Test Generation

Dieter Hogrefe, Beat Koch, and Helmut Neukirchen

Institute for Telematics, University of Lübeck
Ratzeburger Allee 160, D-23538 Lübeck, Germany
Tel.: +49 451 500 3721 Fax: +49 451 500 3722
{hogrefe,bkoch,neukirchen}@itm.mu-luebeck.de

Abstract. The purpose of this paper is to describe how computer-aided test generation methods can benefit from the time features and extensions to MSC, SDL and TTCN which are either already available or currently under study in the EC Interval project. The implications for currently available test generation tools are shown and proposals for their improvement are made. The transformation of MSC-2000 time concepts into TTCN-3 code is described in detail.

1 Introduction

Computer-aided test generation (CATG) from system specifications has been an active field of research for many years [1, 5, 10, 24]. This research has resulted in the development of a number of test generation tools [2, 8, 9]. Today, two industrial-strength, commercially available CATG applications exist [6, 18]. These tools take formal system specifications with the 1992 edition of the *Specification and Description Language (SDL-92)* and test purpose descriptions with the 1996 version of *Message Sequence Charts (MSC-96)* as input and produce test suites based on the second edition of the *Tree and Tabular Combined Notation (TTCN-2)* [14].

Meanwhile, the standards of both SDL and MSC have been updated (SDL-2000 [16], MSC-2000 [15]) and a thoroughly new version of TTCN has been standardized (TTCN-3 [7]). In addition, the European Commission has set up the *Interval* project [21] to prototype an SDL, MSC and TTCN-based tool chain for the development and testing of systems with real-time constraints. During the first project stage, the Interval consortium identifies constructs which are suitable for capturing, specifying, modelling and testing real-time requirements. Based on these constructs, the consortium proposes time extensions to the formal languages as ITU-T recommendations. In the second project stage, tools will be developed which include the new time constructs.

Taking existing test generation tools as reference implementations, this paper evaluates the implications of existing and proposed time extensions to CATG. It is structured as follows: Section 2 shows when and why time constructs are

needed during testing. Section 3 contains an overview of the timer concepts in MSC, SDL and TTCN. In Section 4, timer support of the test generation tools TestComposer and Autolink is discussed. Section 5 is the main part of this paper. It examines first how the test generation process may be improved through the use of SDL-2000 together with the proposed extensions. Second, the benefits of using MSC-2000 for test purpose description are shown and a concrete mapping of MSC-2000 time concepts to TTCN-3 is presented. Section 6 concludes this paper.

2 Timer in Test Purpose Descriptions

Timers in test sequences have one of the following purposes:

- assuring that test cases end even if they are blocked due to unexpected behavior of the system under test (SUT);
- checking constraints on the response time of the SUT;
- delaying the sending of messages to the SUT in order to
  - allow the SUT to get into a state where it can receive the next signal (if the tester is too fast);
  - check the reaction of the SUT if a signal is delayed too long (invalid behavior specification);
  - check that the SUT does not send any signal for a given amount of time.

To guarantee the conclusion of a test case, one or more *global timers* are used. In case of a single-tester test architecture, one timer is started at the beginning of test case execution. Its duration is chosen to be longer than the expected execution time of the test case. At the end of each possible test sequence, the timer is reset. In the exception handling section of the test case, the timeout of the global timer is caught and handled. If a distributed test system is used, a global timer is started within each test component participating in the test execution. In case of a timeout in any test component, the other test components have to be notified in order to let them conclude the test execution gracefully.

Time constraints are checked through the use of one or a pair of *guarding timers*. Guarding timers are started when a signal is sent to the SUT. If a lower bound is specified in the time constraint, one timer has to expire before the response signal from the SUT is received. The second timer — which checks the upper bound of the time constraint — is reset immediately upon reception of the response signal. Premature reception of the response signal or the expiration of the second timer are caught in the exception handling section of the test case and result in a FAIL verdict.

A *delaying timer* is specified by inserting a timer start operation immediately followed by a timeout event into the test sequence.

3 Timer in Formal Languages

The formal languages MSC, SDL and TTCN all contain timer support. In this Section, an overview of the timer concepts of these languages in given.

3.1 MSC-96

Timer support in MSC-96 [13] is very basic: there exist events to *set* and *reset* a timer, and a *timeout* event. Timer events are identified by a mandatory timer name and an optional timer instance name. The specification of a timer duration is optional; if it is specified, it has no semantics. Pairs of timer set and reset/timeout events must be specified on the same MSC instance.

3.2 MSC-2000

MSC-2000 [15] supports the same basic timer events as MSC-96, with some changes and refinements. First of all, the *set* event has been renamed to *start-timer* and *reset* is now called *stoptimer*. If a duration is specified, then it must be done in the form of an interval with an optional lower bound (default value: zero) and an optional upper bound (default value: infinite). This means that the timer can expire within the specified period.

In addition to the basic timer concepts, MSC-2000 also provides a timed semantics for constraining and measuring the time of events. (However, a formal semantics for MSC-2000 is still missing.) Using the external data language approach introduced in MSC-2000, variables of type *Time* may be declared. There are two operators to measure time and store it in time variables: one to determine the absolute time at the moment of the execution of a given event, and one to determine the amount of time which passes between two events. It is also possible to specify time constraints: the lower and upper bound of a time interval between a pair of events may be defined in order to specify the allowed delay between those events. For a single event, the absolute time of occurrence can be constrained, too.

An extension to the MSC-2000 standard has been proposed by the Interval consortium to ITU-T Study Group 10 in [20]. A new symbol is proposed to express periodic occurrence of repetitive events which are folded into a loop.

3.3 SDL-2000

In SDL-2000 [16], timer declarations are mandatory. As part of the declaration, a constant default duration may be defined. With the *set* statement, a timer is activated. With the *reset* statement, a timer is put back to the inactive state. If an active timer expires, a signal with the same name as the timer is put into the input queue of the process which contains the timer. This corresponds to the *timeout* event in the MSC language. Whereas timers in SDL-2000 and MSC-2000 are basically equivalent, the new time constraint concept of MSC-2000 has no equivalent in standard SDL-2000.

Timer handling has been a weak point of SDL since its first introduction and there has been no improvement with the publication of SDL-2000. Therefore, several timer and time semantics related extensions to the SDL standard have been proposed by research groups [19] and Interval consortium members [3, 4, 11]. The latter proposals include

– the addition of cyclic timers which are automatically restarted after expiration;
– mechanisms to read a timer value;
– the introduction of interruptive signals and timeouts.

Furthermore, a real-time semantics is introduced. This semantics allows to assign urgencies to transitions and to model time progression caused by actions which are annotated with a corresponding assumption on time consumption.

3.4 TTCN-2

In TTCN-2 [12], there are three timer operations: the common *START* and *CANCEL* operations to activate and deactivate a timer, as well as the *READ-TIMER* operation which returns the amount of time which has passed since a timer has been activated. Timer expiration is caught with the *TIMEOUT* event.

There are several problems with the implementation of timers in TTCN-2. First, timers have to be declared at test suite level. According to the standard, a full set of timers must be allocated for each test component, potentially wasting scarce hardware resources. The second problem concerns the applicability of TTCN-2 to the testing of real-time time constraints: timeout events are stored in a list until they match an alternative in the test sequence. As a consequence, timeout events may remain unnoticed for some time. This in turn may lead to incorrect test execution and verdict.

Moreover, due to the snapshot semantics of TTCN, it has to be noted that when using the existing timer concepts, a coherent and valid test verdict for real-time tests can only be found if the test equipment is reasonably fast. Since the snapshot semantics may summarize time-critical events arriving at different queues into one snapshot, important timing or ordering information might get lost. In this case, it is not decidable whether a violation of real-time constraints has occurred or not. The test verdict will rather depend on the question of how the triggering events of an alternative are ordered in the TTCN dynamic behavior description.

To solve this problem, a refinement of the standard snapshot semantics is proposed in [25]. Instead of testing time constraints using standard timers, additional columns for earliest and latest execution times of TTCN events are proposed. Since this way of specifying time constraints is orthogonal to the evaluation of alternatives, the test verdict does not depend on the speed of the tester or the ordering of alternatives.

Nevertheless, even with [25] the test system has to be fast enough in order to avoid the overflow of input queues. Therefore, sufficient processing capabilities of the tester are in any case a necessary prerequisite of real-time testing.

3.5 TTCN-3

TTCN-3 [7] renames some of the timer operations of TTCN-2: the keyword to deactivate a timer is now *stop* and the elapsed time of an active timer can

be queried with the *read* operation. In addition, the *running* operation returns *true* if a given timer is running, *false* otherwise. The *start* operation and *timeout* event remain unchanged. Timer functionality is also included in the synchronous *call* operation. A timeout value may be provided as an optional parameter to this operation. If a timeout occurs, it may be handled as an exception with the *catch* operation.

No concrete proposals have been published so far regarding the extension of time concepts in TTCN-3. However, since TTCN-3 uses the same snapshot semantics as TTCN-2, the weakness of this semantics concerning real-time testing still holds for TTCN-3. A forthcoming proposal to overcome this problem is currently under study by the Interval consortium. Rather than using the standard timers to test real-time requirements, it is intended to separate the description of functional requirements (e.g. signal reception and "functional" timeouts) and non-functional (i.e. real-time) constraints. Since these extensions are currently under study, the test cases given in this paper are written using standard TTCN-3 notation.

4 Timer in Current Test Generation Tools

At the time of writing this paper, there are two major test generation tools on the market which take SDL-92 and MSC-96 specifications as input and produce TTCN-2 as output: TestComposer [18] and Autolink [6]. In this Section, the current status of these tools with respect to timer support is presented.

4.1 TestComposer

TestComposer automatically generates four types of timers during the computation of test cases:

- a timer TAC is set whenever a test component waits for a response from the SUT. A fail verdict is assigned in case of a timeout. With respect to timer purposes introduced in Section 2, TAC corresponds to a guarding timer;
- the timer $TWAIT$ is another guarding timer: it checks that time to execute an implicit send does not exceed a predefined amount of time;
- the timer $TNOAC$ is set to check that the SUT does not send a message to the tester for a specific amount of time. $TNOAC$ is a delaying timer;
- $TEMPTY$ is a delaying timer which is used to force a timeout in the SUT.

4.2 Autolink

Autolink generates the declaration of a global timer *T_Global* automatically. Depending on the test architecture, timer statements for *T_Global* are added to the test case behavior description and the top-level test steps of all parallel test components.

Guarding and delaying timers can be specified by the user in test purpose MSCs with timer set, reset and timeout events; these events are translated into corresponding TTCN-2 statements during test generation.

4.3 Discussion

With the methods available in the current test generation tools, the common cases for using timers in test cases can be handled fairly well. However, both tools do not offer optimal timer support. On the one hand, unnecessary timer events may be generated with the fully automatic method in TestComposer. These events have to be removed from the test suite manually. On the other hand, while Autolink offers complete flexibility regarding timers, the manual specification with MSC-96 may be laborious. This is especially true if an SUT response has to fall within a time interval: with the MSC-96 notation used by Autolink, two timers must be drawn, which increases the effort to specify the test purpose MSC and reduce its readability (see Figure 1). Neither tool supports the reading of timer values.

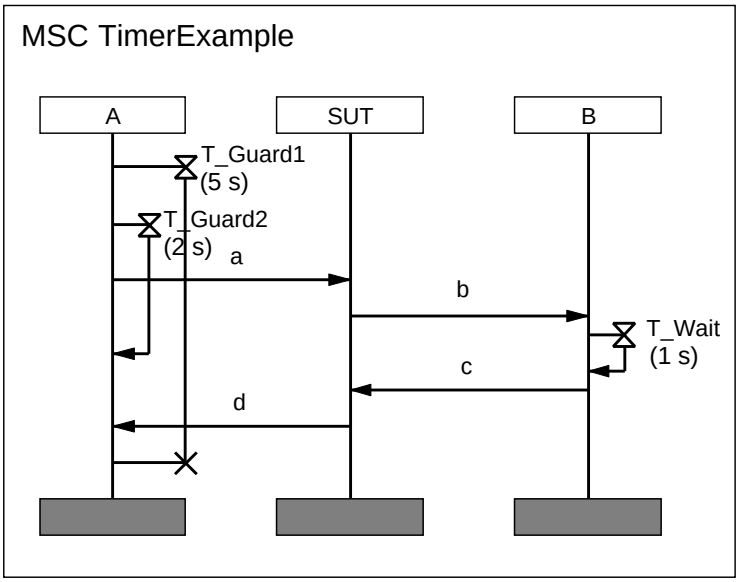

Fig. 1. Timer specification for Autolink with MSC-96

5 Improving the Test Generation Process

Both TestComposer and Autolink are test-purpose-based test generation tools. This means that they need a formal description of the test purpose which they can transform into a TTCN test case. The transformation is done either by direct translation from MSC to TTCN or by performing a state space exploration of

an SDL specification. Test purpose descriptions may be provided in the form of MSC-96 diagrams for both tools.

CATG tools may benefit from the use of formal languages with time extensions in a number of ways:

- reduction of the state space during exploration-based test generation with timed SDL;
- automatic generation of time requirements for test equipment with timed SDL;
- improvement of the capabilities to efficiently describe timing constraints in test purpose descriptions by using MSC-2000.

5.1 Test Case Generation with Timed SDL

The extensions proposed by the Interval consortium for SDL are mainly intended for verifying and validating a specification with respect to time properties. Nevertheless, automatic test generation benefits for two reasons from such time annotations.

First, the state space of a timed SDL model can be reduced in comparison to an untimed specification. The reason is that an untimed specification allows a lot of unrealistic scenarios which cannot occur in practice, because it contains paths where time does not progress at all. By using a real-time semantics and a timed SDL model, the state space can be reduced to the realistic scenarios. As CATG is mainly based on representing observable events of paths allowed by an SDL model, unrealistic test cases can be avoided.

Second, if additional timing information is given for all symbols contained in an SDL transition, the exact moment when observable events are allowed to take place can be determined. Test cases which take this information into account can be derived automatically. However, this topic is subject to further study. If additional timing information is not available for a whole transition, it is still possible to specify real-time requirements using MSC-2000. The usage of MSC-2000 for test description is shown in Section 5.3.

5.2 Generation of Time Requirements for Test Equipment

TTCN assumes that the test equipment is always fast enough to test the IUT. While this assumption is legitimate if only time non-critical functional behavior is tested, it may not hold for real-time applications. The processing speed of the tester may not be fast enough to keep track with the test events that happen at the PCOs. As an example, during the development of the GSM test suite at ETSI, there were various occasions where the possible lack of sufficient speed of the test devices had to be taken into account. In some cases, this problem was resolved by letting the tester respond to a signal from the SUT before it even receives the signal, just by assuming that the signal will arrive eventually. If the tester had to wait for the reception of the signal, it would be not fast enough to respond to it. While such workarounds are possible, they are problematic,

because the order of test events has to be changed. As a consequence, the prose test purpose description does no longer correspond to the formal description.

In general, it seems more feasible to require some speed of the tester and treat these requirements as part of the test suite. If this approach is taken, time constraints for the tester have to be defined somehow. This means that the tester is required to execute test events within a certain time interval in order to test the SUT accurately and successfully.

A very detailed idea about the timing behavior of the SUT is required to determine the time intervals between test events. The test designer or a test generation tools need to know at which points in time the SUT may be stimulated or events from the SUT may be observed. Traditionally, this timing information has not been part of the SDL specification. However, if such timing information is added to the SDL specification, the minimal time interval between test events can be derived which the test equipment must be able to process. Based on this information, it is possible to generate benchmarks for the tester. An example for a tester benchmark is given below using the TTCN-3 notation. This benchmark checks whether the test equipment is fast enough to send two consecutive messages within a duration specified by *required_time*:

```
1:  timer T;
2:  T.start(required_time);
3:  A.send(a);
4:  A.send(b);
5:  if (T.running)
6:  {
7:    T.stop;
8:    verdict.set(pass);
9:  }
10: else
11: {
12:   verdict.set(fail);
13:   MyComponent.stop;
14: }
```

5.3 Using MSC-2000 for Test Purpose Description

Figure 2 shows the test description of Figure 1 in MSC-2000 notation. The use of the time interval notation instead of four separate timer symbols (two set events, one reset and one timeout) to specify two guarding timers makes the diagram much more readable. Given a time interval where the start event is a send to the SUT and the end event is a receive from the SUT, the test generation tool has to perform the following actions:

1. Check the time interval to establish the number of timers which are needed to represent the interval with TTCN. If just one boundary value is specified, only one timer is needed. If both a minimal and a maximal time point are

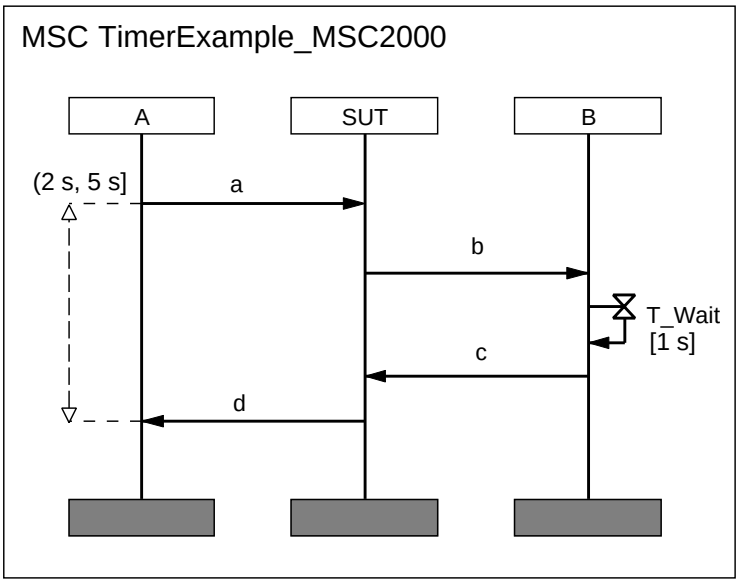

Fig. 2. Time constraint specification with MSC-2000

specified, two TTCN timers are needed. Since no timer name is specified with the time interval notation, the tool has to select the timer names by itself. Preferably, the user should be able to define timer name templates such as *T_Guard_Min* and *T_Guard_Max*. The tool then must check if timers with these name are in use already. If they are, new timers (e.g., *T_Guard_Min_2* and *T_Guard_Max_2*) must be declared. In order to minimize the number of timers which have to be declared, the test configuration has to be taken into account in this step.

2. If the time interval contains expressions with measurements (see Section 5.3), replace the time patterns with the corresponding variable identifier. If necessary, convert time values to seconds.
3. Create the appropriate TTCN timer statements. This step depends on the number of time points specified in the MSC. In the examples below, TTCN-3 is produced from the MSC in Figure 2, assuming that there is a test component handling just PCO *A*. A similar transformation can be done for TTCN-2.

If only the lower boundary value is specified, create the following statements:

```
1:  timer T_Guard_Min;
2:  A.send(a); T_Guard_Min.start(2);
3:  alt {
4:  []  T_Guard_Min.timeout;
5:  []  A.receive(d)
```

```
 6:      {  verdict.set(fail);
 7:         MyComponent.stop;
 8:      }
 9:  }
10:  A.receive(d);
```

The case where d is received prematurely by PCO A (lines 5 to 8) may as well be handled in a default. If only the upper boundary value is specified, create the following statements:

```
 1:  timer T_Guard_Max;
 2:  A.send(a); T_Guard_Max.start(5);
 3:  alt {
 4:  []  A.receive(d)
 5:      {  T_Guard_Max.stop; }
 6:  []  T_Guard_Max.timeout
 7:      {  verdict.set(fail);
 8:         MyComponent.stop;
 9:      }
10:  }
```

If the lower and the upper boundary values are specified, create the following statements:

```
 1:  timer T_Guard_Min;
 2:  timer T_Guard_Max;
 3:  A.send(a); T_Guard_Min.start(2); T_Guard_Max.start(5);
 4:  alt {
 5:  []  T_Guard_Min.timeout;
 6:  []  A.receive(d)
 7:      {  verdict.set(fail);
 8:         MyComponent.stop;
 9:      }
10:  }
11:  alt {
12:  []  A.receive(d)
13:      {  T_Guard_Max.stop; }
14:  []  T_Guard_Max.timeout
15:      {  verdict.set(fail);
16:         MyComponent.stop;
17:      }
18:  }
```

The case where d is received prematurely by PCO A (lines 6 to 9) may as well be handled in a default. TTCN-3 has a special notation to put a timeout value on a procedure call. If the start event of a timer interval in an MSC is a method call (cf. Figure 3), then the following code should be generated in order to guard the call by an upper time bound of e.g. 3 seconds:

```
1:  A.call(x, 3);
2:  {
3:    [] A.getreply(x);
4:    [] A.catch(timeout);
5:       {  verdict.set(fail);
6:          MyComponent.stop;
7:       }
8:  }
```

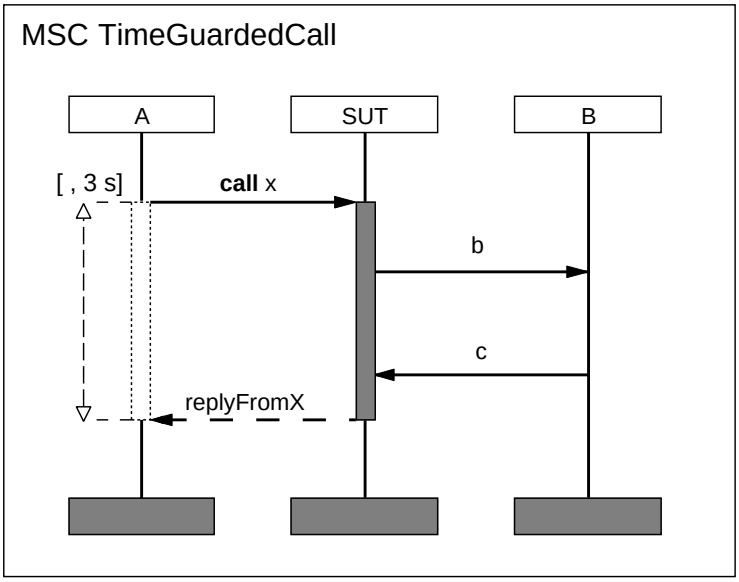

Fig. 3. Time constraint for a method call in MSC-2000

Time Measurement. In MSC-2000, time can be measured and stored in variables. These measurements can be reused, e.g. to specify time intervals. Figure 4 shows the two kinds of time measurements provided by language: *&t1* is a relative measurement; the time which passes between the sending of *a* and the reception of *d* is stored in a variable *t1* of type *Time*. *@t2* is an absolute measurement, which means that the value of an existing global clock is stored in a variable *t2* of type *Time*. The global clock is started when the first event in the MSC is executed.

Transforming a relative time measurement into TTCN is straight-forward. The test generation tool needs to declare a special timer used for the measurement. This timer is activated after the first event in the MSC has occurred. After

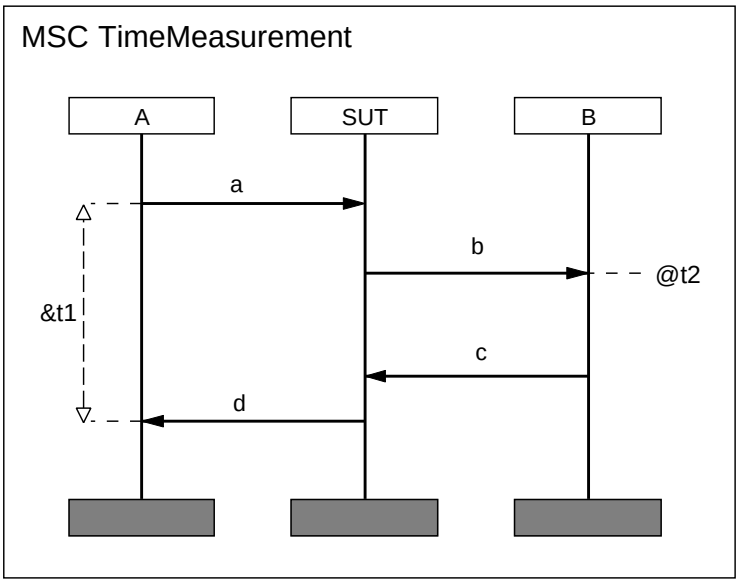

Fig. 4. Time measurements in MSC-2000

the second event, the timer is read and deactivated. The only problem is the fact that to start a timer in TTCN, a duration has to be defined, which in this case is not known in advance. As a solution, the test designer has to provide a maximum value for time measurement. Most likely, this value is available anyway because a timeout period has to be defined for a global test case timer. From the MSC in Figure 4, the tool should generate the following TTCN-3 statements:

```
1:  timer T_Measure := 10000;
2:  var float t1;
3:  A.send(a); T_Measure.start;
4:  A.receive(d); t1 := T_Measure.read;
5:  T_Measure.stop;
```

To measure an absolute time value, a global timer has to be started at the beginning of the test case. The measurement can then be done by using the *read* operation on the global timer. Below is the TTCN-3 code generated for the measurement of *t2*:

```
1:  timer T_Global = 10000;
2:  var float t2;
3:  T_Global.start;
4:  B.receive(b); t2 := T_Global.read;
5:  B.send(c);
```

MSC-2000 also allows to measure or to constrain the amount of time which passes between a pair of events on different instances. This kind of time interval cannot be represented with standard TTCN-3 timers, since the start and read or timeout operations of the same timer cannot be distributed between different parallel test components. If there is more than one test component, coordination messages might be used to synchronize the parallel test components concerning the two relevant events. However in this case, the time needed to transmit these coordination messages has to be taken into account.

6 Conclusion

In this paper, the current state of SDL, MSC, TTCN and test generation tools with regard to timer support has been presented. Commercially available test generation tools already allow to generate TTCN-2 test suites which reflect time requirements expressed by standard MSC-96 timers. However, due to the simple timer concepts of MSC-96, the specification of time constraints in test purpose descriptions may be quite laborious. It has been shown that the use of the MSC-2000 time interval notation can facilitate the specification of time constraints for events on the same instance. A translation of MSC-2000 time constructs into TTCN-3 code has also been presented. The mapping of MSC-96 timers to MSC-2000 time constructs and the transformation of TTCN-2 to TTCN-3 is straightforward. Therefore it will be possible to automatically generate TTCN-3 test cases which test the conformance to such MSC-2000 time constraints.

Nevertheless, many challenges remain. The testing of time constraints in a distributed test architecture has not been solved yet. Currently, it is neither possible to derive test cases in an automated way nor to test real-time requirements if several parallel test components observing time critical events are involved. Care has to be taken to synchronize these parallel test components not only in a functional manner but also with regard to their local clocks.

It has also not been shown yet that it is possible to generate real-time tests from time extended SDL models. The accuracy of automatically derived test cases depends on how exhaustively an SDL model is enriched with time annotations. Research by the Interval consortium will show whether this is feasible. Moreover, as an underlying basis, existing test theory has to be extended in the area of real-time testing.

Due to the problems introduced by the snapshot semantics of TTCN, standard timers should not be used to test real-time constraints where a high resolution of timing information is required. Therefore in the testing domain, the next step which will be done in the Interval project is to present a real-time extensions for TTCN-3 which allows deterministic real-time testing.

Acknowledgements

Part of this work has been sponsored by the European Commission under contract IST-1999-11557.

References

1. C. Bourhfir, R. Dssouli, E. Aboulhamid, and N. Rico. Automatic executable test case generation for extended finite state machine protocols protocols. In IWTCS'97 [17], pages 75–90.
2. C. Bourhfir, R. Dssouli, E. Aboulhamid, and N. Rico. A test case generation tool for conformance testing of SDL systems. In SDL'99 [23], pages 405–419.
3. M. Bozga, S. Graf, A. Kerbrat, L. Mounier, I. Ober, and D. Vincent. SDL for real-time: what is missing? In *SAM 2000 – 2^{nd} Workshop on SDL and MSC*, pages 108–122, Grenoble, France, June 2000.
4. M. Bozga, S. Graf, L. Mounier, I. Ober, J.-L. Roux, and D. Vincent. Timed extensions for SDL. In *SDL Forum 2001*, Copenhagen, Denmark, June 2001.
5. M. Clatin, R. Groz, M. Phalippou, and R. Thummel. Two approaches linking a test generation tool with verification techniques. In *Proceedings of IWPTS '95 (8th Int. Workshop on Protocol Test Systems*, pages 151–166, Evry, France, September 1995.
6. A. Ek, J. Grabowski, D. Hogrefe, R. Jerome, B. Koch, and M. Schmitt. Towards the industrial use of validation techniques and automatic test generation methods for SDL specifications. In SDL'97 [22], pages 245–259.
7. ETSI, Sophia Antipolis, France. *Methods for Testing and Specification (MTS); The Tree and Tabular Combined Notation version 3; TTCN-3: Core Language*, v1.0.10 edition, November 2000. DES/MTS-00063-1.
8. J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho. An experiment in automatic generation of test suites for protocols with verification technology. *Science of Computer Programming*, 29, 1997.
9. J. Grabowski. *Test Case Generation and Test Case Specification with Message Sequence Charts*. PhD thesis, University of Bern, Bern, Switzerland, February 1994.
10. J. Grabowski, D. Hogrefe, and R. Nahm. Test case generation with test purpose specification by MSCs. In *SDL'93: Using Objects*, pages 253–265, Darmstadt, Germany, October 1993. Elsevier Science Publishers B.V.
11. S. Graf. Timed extensions for SDL, November 2000. Delayed Contribution No. 13 to ITU-T Study Group 10, Questions 6&7.
12. ISO/IEC. *Information technology – Open Systems Interconnection – Conformance testing methodology and framework*, 1994. International ISO/IEC multipart standard No. 9646.
13. ITU-T, Geneva, Switzerland. *Message Sequence Charts*, 1996. ITU-T Recommendation Z.120.
14. ITU-T, Geneva, Switzerland. *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation*, 1997. ITU-T Recommendation X.293-ISO/IEC 9646-3.
15. ITU-T, Geneva, Switzerland. *Message Sequence Charts*, November 1999. ITU-T Recommendation Z.120.
16. ITU-T, Geneva, Switzerland. *Specification and Description Language (SDL)*, 1999. ITU-T Recommendation Z.100.
17. *Testing of Communicating Systems*, Cheju Island, Korea, September 1997. Chapman & Hall.
18. A. Kerbrat, T. Jéron, and R. Groz. Automated test generation from SDL specifications. In SDL'99 [23], pages 135–151.

19. A. Mitschele-Thiel. *Systems Engineering with SDL – Developing Performance-Critical Communication Systems*. Wiley, Chichester, England, 2001.
20. H. Neukirchen. Corrections and extensions to Z.120, November 2000. Delayed Contribution No. 9 to ITU-T Study Group 10, Question 9.
21. Interval Consortium Web Page. `http://www-interval.imag.fr/`, 2000.
22. *SDL'97 – Time for Testing*, Evry, France, September 1997. Elsevier.
23. *SDL'99 – The Next Millennium*, Montréal, Québec, Canada, June 1999. Elsevier.
24. G. v. Bochmann, A. Petrenko, O. Bellal, and S. Maguiraga. Automating the process of test derivation from SDL specifications. In SDL'97 [22], pages 261–276.
25. Th. Walter and J. Grabowski. Real-time TTCN for testing real-time and multimedia systems. In IWTCS'97 [17].