Time Concepts for UML 2.0 Based Testing

Zhen Ru Dai and **Ina Schieferdecker**
Fraunhofer FOKUS, Berlin

Abstract

The *UML 2.0 Testing Profile* (U2TP) is the first profile based on UML 2.0. Although UML 2.0 itself provides means for time specifications, U2TP added new time concepts for testing specific needs. The motivation for the U2TP time concepts come from different languages and profiles with real-time features, including the *UML Profile for Schedulability, Performance and Time Specification* (SPT).

This paper provides an overview about time concepts, primarily concentrating on the UML 2.0 and U2TP time concepts. Since time concepts are not only important for test specifications, the U2TP time concepts may be considered for adoption in a UML 2.0 real-time profile.

1) Introduction

The *UML 2.0 Testing Profile (U2TP)* [U2TP] provides means for model-driven testing with UML 2.0 [UML2.0]. The work on U2TP has been initiated in 2001 in order to develop a UML 2.0 profile for the testing domain. Currently, the FTF work has just been finalized and is ready for vote by end of April 2004 in St. Louis.

Although UML 2.0 itself provides means to specify time events, U2TP added new time concepts according to test specific needs. When defining time concepts for the profile, the U2TP consortium looked into various existing languages and profiles with time concepts, including the *UML Profile for Schedulability, Performance and Time Specification* (SPT). The U2TP consortium was not able to adopt the time concepts defined in SPT for the following reasons: Firstly, SPT is still based on UML 1.4 and U2TP should be set upon UML 2.0. Secondly, SPT uses stereotypes and tag values as its extension specification formats, where U2TP defines its concepts by metaclasses and stereotypes. Last, the time concepts defined in SPT were too complex for the needs of testing.

In this paper, we will give an introduction to the UML 2.0 and U2TP time concepts and propose further real-time concepts. Although the U2TP time concepts have been defined for test specific needs, they can also be useful for software system specification. Therefore, this paper can serve as an input for advanced real-time concepts for UML 2.0.

2) Basic Real-Time Concepts

Basic real-time concepts should provide means to the following tasks:

- to *express* time values,
- to *specify* and *constrain* time events,
- to *observe* and *measure* time events.

Time is related to clocks which provide the actual time value. Clocks can belong to a whole test system, where all test components has access to the global clock and the global time value can be retrieved from. Clocks can also be bound to one or a subset of test components. In this case, test components with the same clock get the same time value. Time value can be expressed in different ways. Time can be absolute, e.g. (*Year:Month:Day:Hour:Minutes:Seconds)* or relative to a certain time point, e.g. (*OffsetTime + 3s)*.

In order to constrain time events, a timer provides good means. The main benefit of a timer is to influence the system behaviour actively, i.e. if the timer runs out, an alternative system behaviour can be executed. A timer can incorporate the following tasks:

- ***Assuring a proper termination:*** In order to assure a proper termination of a test case, a *test case timer* is started at the beginning of a test case. Its duration is chosen to be longer then the expected execution time of the test case. At the end of each possible test sequence, the timer is stopped. A fail or error verdict is generated if the testcase timer expires.

- ***Constraining response time:*** In every reactive system, the response time of an event should be restricted by a timer. For this purpose, a *constraining timer* is started after a signal is sent. It is stopped if the response is received within the timer duration. Otherwise a timeout is triggered and an alternative behavior, i.e. default behavior, is activated.
- ***Delaying message sending:*** A *delaying timer* is specified by inserting a timer start operation immediately followed by a timeout event into the test sequence. The most common uses for this kind of timers is to delay the sending of a signal to the system under test (SUT) because:
  - the SUT needs some time to get into a state where it can receive the next signal;
  - The test case is to check the reaction of the SUT if a signal is delayed too long (invalid behavior specification);
  - The test case is to check that the SUT does not send any signal for a given amount of time.

Time measurement is used to observe the execution time of one or several events. Time measurements relate to the value of a local and global clock, respectively. When measuring time, there should be action provided to save the measured time value. There are two methods to measure time: *Absolute* time measurement estimates the time point when an action is performed. *Relative* time measurement evaluates the interval of execution time of two events. The relative time measurement can also be calculated by two absolute time points.

3) Simple Time Concepts in UML 2.0

The latest version of UML, version 2.0, defines a sub-package called *SimpleTime* where new metaclasses for time are added. The new concepts provide means to express absolute time values, but unfortunately no means for relative time values. Events can be triggered absolutely or relatively to a certain time point. Events can be observed and constrained within a time interval and duration.

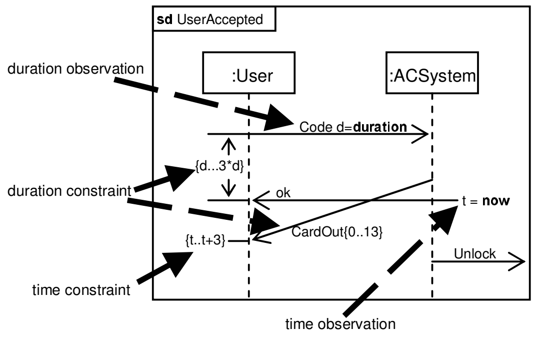

Figure 1: An Example on Time Concepts

Figure 1 shows in a sequence diagram how time events can be observed and constrained in a UML 2.0 model. The *:User* sends a message Code and its duration is measured. The *:ACSystem* will send two messages back to the *:User*. *CardOut* is constrained to last between 0 and 13 time units. Furthermore, the interval between sending of *Code* and the reception of *OK* is constrained to last between *d* and *3*d* where *d* is the measured duration of the *Code* signal. The observation of the time point *t* at the sending of *OK* and the constraint of *CardOut* reception time point are shown.

In contrast to SPT [SPT], a model using *SimpleTime* metaclasses does not account for the more complex aspects for time and time measurement, e.g. relativistic effects in distributed systems, imperfect clocks with finite resolution, overflows, drift, skew, etc. It assumes that applications for which such characteristics are relevant will use a more sophisticated model of time provided by an appropriate profile.

4) Time Concepts in U2TP

For test specifications, the simple time concepts of UML 2.0 do not satisfy all needs. Therefore, U2TP provides two additional time concepts: *timer* and *timezone*.

Timers are needed to manipulate and control test behavior, or to assure that a test case terminates properly. A timer is a predefined interface which is owned by a test component. A test component may own multiple timers.

A timer can be started with certain duration and stopped before it expires. When a timer expires after its predefined duration, a timeout is triggered automatically. A timeout is only allowed to be sent to the owning class of the timer. The actual remaining time value of an active timer and its status can also be checked by actions. In order to provide more flexibility, no restriction is made about the visibility of a timer. If a timer is private, the timer can only be accessed by its owning class. If the timer is public, any other active classes of the test system is allowed to manipulate the timer. The graphical syntax for starting, stopping a timer and receiving the timeout message within the owner class of the timer are adopted from the Message Sequence Charts (MSC). The notations for the timer actions are shown in Figure 2.

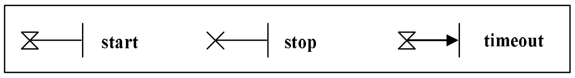

Figure 2: U2TP Timer Notations for Interactions and Activities

Figure 3 shows the usage of the different kinds of timers mentioned in Section 3) in an example: The test case starts with the setting of a test case timer ***TC*** with duration of 10 s. It is stopped before the test verdict is set and the test case terminates. Timer ***DT*** is a delaying timer. It specifies the minimum time period to pass before a second signal *Unlock* is sent to the SUT. Here, it is important to wait for the timeout signal of ***DT***. Timer ***CT*** is a constraining timer to make sure that the response event *OK* is sent from the SUT to the test component within 3 seconds.

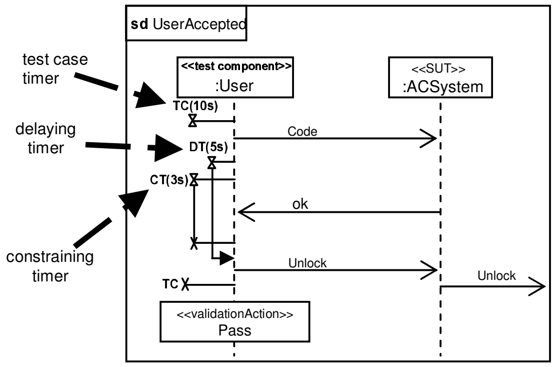

Figure 3: Example for Timer Usage

Timezones serve as a grouping mechanism. Timezones are used to group and synchronize test components within a distributed test system. Each test component belongs at most to one timezone. Test components in the same timezone have the same perception of time, i.e. test components of the same timezone are considered to be time synchronized. Comparing time-critical events within the same timezone is allowed. There are actions to set and to retrieve the timezone of a test component. Figure 4 and Figure 5 show the metaclasses of the U2TP time concepts.

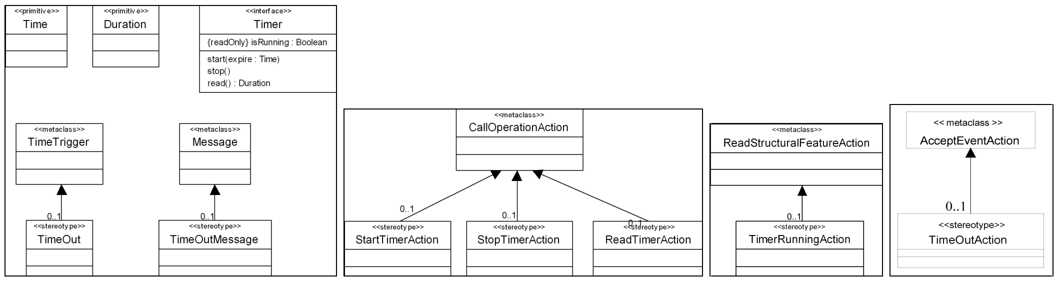

Figure 4: U2TP Timer Concepts for Interaction and Activity Diagrams

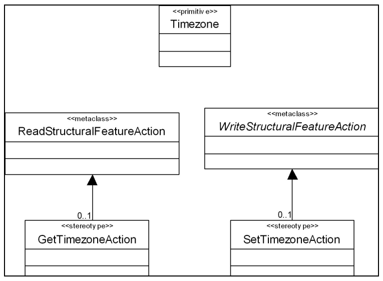

Figure 5: U2TP Timezone Concepts

5) Conclusion

In this paper, we outlined some important real-time aspects which should be considered when defining a new real-time profile for UML 2.0. Additionally, we introduced the existing simple time model defined in UML 2.0 and gave an insight into the extended time concepts defined in the UML 2.0 Testing Profile.

As a conclusion, we think that the simple time model in UML 2.0 should be extended in order to provide means for the specification of more detailed real-time models. Real-time concepts which we consider important are: representing relative time points (UML 2.0 can only represent absolute time points), constraining time events (e.g. by timers), time synchronization (e.g. by timezone).

References

[Koch01] Beat Koch, *Test-purpose-based Test Generation for Distributed Test Architectures*, PhD Thesis, Lübeck, 2001

[MSC2000] ITU-T, Geneva, Switzerland. *Message Sequence Charts*, ITU-T Recommendation Z.120. Version 2000-06-23, March 2000

[RTMSC] H. Neukirchen. Corrections and extensions to z.120. Technical report, Institute for Telematics, University of Luebeck, November 2000. Contribution to ITU-T Study Group 10, Questions Q.9/10

[SG10-MSC] H. Neukirchen. Corrections and extensions to Z.120, November 2000. Delayed Contribution No. 9 to ITU-T Study Group 10, Question 9

[SPT] *UML Profile for Schedulability, Performance, and Time Specification*, OMG Final Adopted Specification, September 2003, Version 1.0, formal/03-09-01

[TIMEDTTCN] Z. R. Dai, J. Grabowski and H. Neukirchen. *TIMEDTTCN-3 – A Real-Time Extension for TTCN-3*, Testcom 2002, Berlin

[TTCN-CN] *The Testing and Test Control Notation version 3, Part 1: TTCN-3 Core Language*, ETSI ES 201 873-1 V2.2.1, Reference: RES/MTS-00063-1 [2], February 2003, ETSI, Sophia Antipolis, France. Methods for Testing and Specification (MTS)

[TTCN-GFT] *The Testing and Test Control Notation version 3, Part 3: TTCN-3 Graphical Presentation Format (GFT)*, v2.2.2, ETSI ES 201 873-3, Reference RES/MTS-00063-3 [2], April 2003, ETSI, Sophia Antipolis, France. Methods for Testing and Specification (MTS)

[U2TP] *UML 2.0 Testing Profile Specification*, version 2.0, OMG Final Adopted Specification, ptc/2004-04-02, April 2004

[UML2.0] *Unified Modeling Language: Superstructure, version 2.0*, Revised Final Adopted Specification, ptc/04-04-08