



***TIMED*TTCN-3 – A Real-Time Extension for TTCN-3**

Jens Grabowski

Institute for Informatics,
Software Engineering Group
University of Göttingen



Outline

- Introduction
- The *TIMED*TTCN-3 Approach
- Example
- Summary, Status & Outlook



TIMEDTTCN-3=TTCN-3+Time Extensions

- Non-functional black-box testing:
 - Testing of real-time requirements, e.g.:
 - Response time/latency, jitter, throughput, ...
- TTCN-3 timer inadequate for real-time:
 - Intended to prevent blocking of test case
 - Snapshot semantics not designed for real-time

J. Grabowski: *TIMEDTTCN-3*

3



Description of Real-Time Requirements

- Separately from functional requirements
- Mathematical formulae, which describe the relationship of timestamps t_i , e.g.:
 - Latency: $\forall i : t_{bi} - t_{ai} \leq 10ms$
- Implemented as TTCN-3 functions

J. Grabowski: *TIMEDTTCN-3*

4



The *TIMED*TTCN-3 Approach

1. Instrument functional testcases to generate timestamps
2. Execute testcase
3. Apply evaluation functions to the generated timestamps:
 - Offline evaluation
 - Online evaluation
4. Assign a test verdict

J. Grabowski: *TIMED*TTCN-3

5



TTCN-3 Extensions

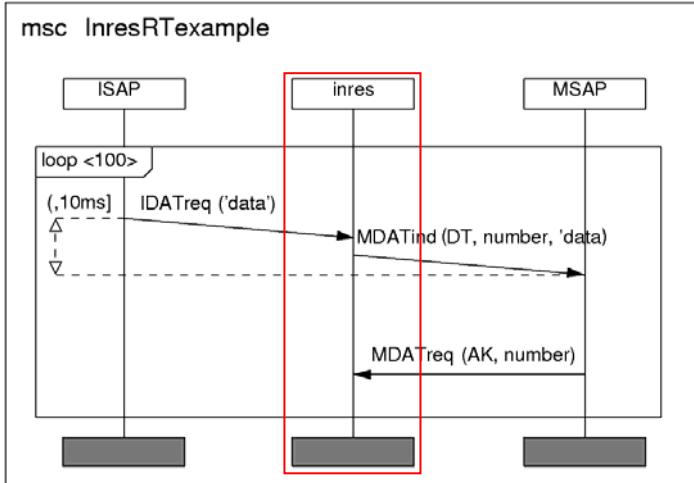
1. **Local clock:** read, wait
2. **Logfile:** log, sort, retrieve
3. **Timezones:** specification of clock synchronised test components
4. **Conf-Verdict:** `pass` \Rightarrow `conf` \Rightarrow `inconc` \Rightarrow `fail`

J. Grabowski: *TIMED*TTCN-3

6



Example



J. Grabowski: TIMEDTTCN-3

7



Testcase

User defined timestamp type

```

type record TimestampType{float logtime,
                         MessagesType messagename}

testcase InresRTexample
{
    var float sendTime:=self.now+5.0;
    for (i:=1; i<=100; i:=i+1)
    {
        resume(sendTime);
        log(TimestampType:{self.now, IDATreq});
        ISAP.send(IDATreqType:{ "data" });
        MSAP.receive(MDATindType:{DT,expected_num, "data" });
        log(TimestampType:{self.now, MDATind});
        MSAP.send(MDATreqType:{AK,expected_num});
        sendTime:=sendTime+0.1;
    }
}

```

The testcase defines a timestamp type and a loop that performs 100 iterations. Each iteration involves sending an IDATreq message from ISAP, receiving a MDATind message from MSAP, and then sending a MDATreq message back to MSAP. The timestamp type is annotated with 'User defined timestamp type'. Callouts provide annotations for the code: 'Read local clock' points to the self.now variable, 'Wait until time point' points to the resume statement, 'Dump timestamp to log' points to the log statements, and 'Calculate next resume time point' points to the sendTime+=0.1 assignment.

J. Grabowski: TIMEDTTCN-3

8



Logfile with Timestamp Objects

```

→ { 5.619, IDATreq}      logfile.first(TimestampType:{?,-},
                                TimestampType:{?,IDATreq} )

→ { 5.627, MDATind}     logfile.next(TimestampType:{?,MDATind} )

→ { 5.632, IDATreq}      logfile.next(TimestampType:{?,IDATreq} )

→ { 5.641, MDATind}     logfile.next(TimestampType:{?,MDATind} )

```

J. Grabowski: *TIMEDTTCN-3*

9



Offline Evaluation Function

```

function latency(logfile timelog, MessageType msg1,
                 MessageType msg2, int count, float upperbound)
{
    return verdicttype
    {
        var TimestampType stampA, stampB;
        var int i;
        timelog.first(TimestampType:{?,-},
                      TimestampType:{?,msg1});
        for(i:=1; i<=count; i:=i+1)
        {
            stampA:=timelog.retrieve;
            timelog.next(TimestampType:{?,msg2});
            stampB:=timelog.retrieve;
            if (not (stampB.logtime-stampA.logtime<=upperbound))
            {
                return conf;
            }
            timelog.next(TimestampType:{?,msg1});
        }
        return pass;
    }
}

```

The diagram illustrates the execution flow of the `latency` function. It starts with the function call and its parameters. The first step is `timelog.first`, which is annotated with "Sort logfile, reset cursor". This is followed by a loop that iterates `count` times. Inside the loop, the `retrieve` method is called twice, annotated with "Retrieve timestamp". After each `retrieve`, the `next` method is called, annotated with "Advance cursor". After the second `retrieve`, a condition is checked: `(not (stampB.logtime-stampA.logtime<=upperbound))`. If the condition is true, `conf` is returned. Otherwise, the `next` method is called again, annotated with "Retrieve timestamp". Finally, the `pass` value is returned.

J. Grabowski: *TIMEDTTCN-3*

10



Linking Execution and Evaluation

```

control
{
    var testrun myTestrunHandle, verdicttype myVerdict;
    myTestrunHandle:=execute(InresRTexample());
    myVerdict:=latency(myTestrunHandle.getLog,
                        IDATreq,MDATind,100,0.01);
    myTestrun.setverdict(myVerdict);
}

```

Handle for accessing logfile and verdict of testrun

Execute instrumented testcase

Evaluate timestamps

Set testrun verdict according to evaluation

J. Grabowski: *TIMEDTTCN-3*

11



Summary and Status

- *TIMEDTTCN-3*, a real-time extension for TTCN-3:
 - Local clocks, Logfile, Conf-verdict, (Timezones)
- Status:
 - *TIMEDTTCN-3* (Syntax, BNF, *TIMEDTTCN-3-GFT*) has been submitted to ETSI for standardization.
 - Ideas of *TIMEDTTCN-3* found its way into the UML testing profile, but not into TTCN-3.
 - Methodology for using *TIMEDTTCN-3* in combination with real-time communication has been presented to TestCom 2004. This work has been submitted to the ETSI work item on test patterns.

J. Grabowski: *TIMEDTTCN-3*

12



Outlook

- Outlook:
 - Extend existing TTCN-3 operational semantics towards real-time
(will only be done if *TIMEDTTCN-3* will find its way into standardization).
 - Refinement of methodology.
 - Extensions towards performance testing.



Credits

The presented work has been carried out together with:

Zhen Ru Dai
(Fokus Fraunhofer in Berlin)
and

Helmut Neukirchen
(University of Göttingen).



Thank you for your attention!

Any questions?