
Test Execution Logging and Visualisation Techniques¹

George Din¹, Justyna Zander¹ and Stephan Pietsch²

¹Fraunhofer Fokus, TIP
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany
din@fokus.fraunhofer.de,
j.zander@fokus.fraunhofer.de,

²Testing Technologies IST GmbH
Rosenthaler Str. 13
10119 Berlin, Germany
pietsch@testingtech.de

Abstract

Traditionally, log traces of test cases execution are stored in textual format and their analysis is done in a post-execution phase. Our approach bases on the concept of graphical presentation of test execution which enables better analysis of log traces and also gives the opportunity to consider them on-line. In particular, we discuss the graphical symbols we chose, we present the logging interface we defined and describe details concerning this interface. This work is done in the context of Testing and Test Control Notation and its Graphical Presentation Format.

Keywords: Testing, TTCN-3, GFT, Test trace, Logging Interface, Visualisation, TraceViewer, SVG

1. Introduction

Testing is the process which is needed to be performed in each phase of software engineering. Test objectives are created during requirements analysis, software designing covers also the test project, implementation and integration phases demands appropriate testing procedures (including deployment), last, but not least operation/maintenance needs ongoing testing.

These are major reasons, to put attention on testing activities. Following the above process it is also useful to consider visualisation of test procedures, executions and results so as to provide better view of particular details for test operators and users. Graphical techniques are gaining more acceptance and wide spread use as they are easier to develop, understand and maintain. Prominent examples are Message Sequence Charts – MSC , and UML Testing Profile– U2TP . With the work on graphical issues some guidance is available to make use of graphics also for the presentation of test execution behaviour.

Nowadays, software applications consisting of multiple communicating programs and distributed data, represent a complex and dynamic environment. Trace events provide a mechanism to gather an insight about the dynamics of the system as a whole, prove if the system follows a foreseen execution pattern, help detect hardware and software faults within the system or contribute to statistical analyses among the performance of the system,

Trace events are emitted by code within the system to make visible internal conditions and states at key places. By examining the trace events emitted by the system, developers can gain a picture of the system flow.

We reuse the tracing mechanisms in the context of testing by enabling the test execution tools to produce fine granular log events in order to evaluate, from outside, the test execution. We couple these mechanisms with

¹ This work is partly sponsored by the ITEA project TT-Medal. <http://www.tt-medal.org/>

graphical visualizations techniques, thus we benefit both from the traditional logging and tracing technique and the more intuitive graphical visualization.

The following paper is divided into six sections. The first one - describes structure of the work, introduces motivation to development of new concepts concerning test logging. In the second part, requirements and related works are discussed. Then, the most important conceptual schemes are provided. Test execution log traces are described, their textual and graphical presentation as well as advantages resulting from visualisation. In the following part Logging Interface is considered. Pros and cons of some alternatives are discussed. Next chapter is devoted to implementation of issues presented before. There are described some visualisation techniques as well as special features of the tool are given. In the last part of this paper main problems are underlined, most important results marked and conclusions taken. Finally future work challenges are foreseen.

2. Requirements and related work

As far as we consider realisation of test visualization we should focus attention on requirements demanded by users. Reliability, functionality, performance are only few of them. We challenge to provide reliable tool working under different conditions. We have to take into account that open source interfaces are needed. Moreover, we want it to run in possibly a lot of environments or at least under conditions governing most often in our computer systems.

We need also some significant functions included in the implementation. These are filtering of some issues or selection of the most important test cases. Due to the filtering facilities the user should be able to restrict the presented information to the particular interest. Performance should face the expectations of users, too. Esthetical aspects of visualization are a challenge as we have to define appropriate colors of different symbols as well as general user friendly outlook. Moreover, naming, graphics and generally view conventions should correspond to the appropriate standards.

Technical issues are of huge interest in the case of a viewer. Possibility to rebuild or reuse considered tool would be appreciated by people trying to develop test visualization techniques application. Finally, installation of newly created product should be relatively easy and understandable.

The need to create trace visualization tools has been observed not only by testing communities. Starting from computer systems, going through specific software monitoring or any other traces resulting from program execution, there is a huge demand on visualisation.

A close related work provides a solution to handle and visualize huge amount of events collected from program traces. They use an XML database to store the events, then with the help of XQuery language retrieve the useful information for visualization purposes.

Another approach for observation of distributed system activities, i.e. execution traces, deployment traces have been investigated for a long time . As the main monitoring feature the flexible graphical facilities has been discussed. Such facilities are needed for the presentation of monitored information to human users in a form which aids comprehension and meets specific application requirements .

The same approach is given by Helmbold : His TraceViewer is a graphical tool for browsing the trace of the execution of a parallel Fortran program written using IBM's parallel Fortran. TraceViewer is intended for examining the ordering relationships between events (which must come first if either) and for reporting data races detected by a companion trace analyzer.

The other point of view is presented in the paper devoted to Model-Based Trace-Checking . Rather than writing a special purpose trace analysis tool, the authors propose that traces can usefully be analysed by checking them against a formal model using a standard model-checker or else an animator for executable specifications.

When a program or system crashes, it may be necessary to analyse a trace recorded in a log file which can be tedious. A trace viewer can help with this task, but human intuition is still required, assume the authors. An alternative approach is to use an executable formal model of the program or system under test to automate the analysis of traces in support of testing. Trace code can be added semi-automatically to the program under test, so that any execution of the system (whether for testing purposes, or in deployment) is logged. Such logs can be played back through a standard model-checker or a special purpose trace analysis program. Erroneous executions will then be detected, even if they did not lead to any obvious and visible system failure or outage

3. From textual test log traces to graphical test log traces

Figure shows the three steps of specifying, executing and analysing test cases. Test specifications are described either in a textual format (e.g. programming language), or in a graphical format (e.g. UML) or ideally in a test language that allows multi-representations (e.g. TTCN-3). Traditionally, log traces of test cases execution are stored in textual format, their analysis is done in a post-execution phase.

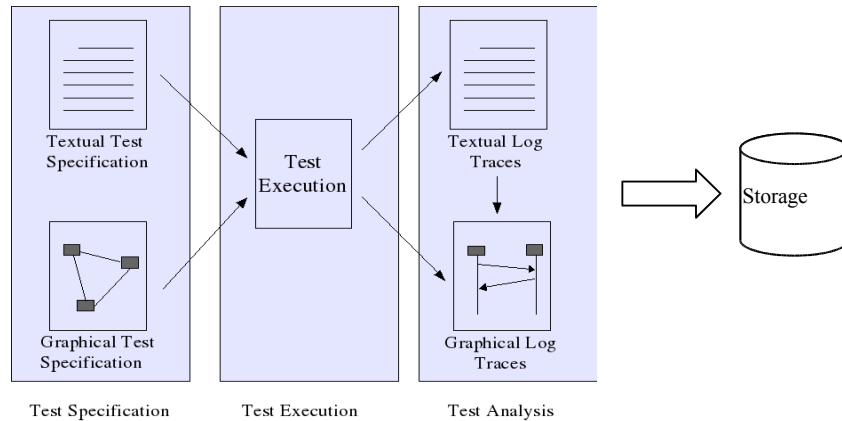


Figure 1 Text based test logging vs. graphic based test logging

Log events are understood as single logical information units generated during the test execution. Log traces are physical representation of the logs, mentioned above. In relation to the test execution their order represents the whole test against considered System Under Test. Log traces are given to the user or test operator in textual format. Logs related to test cases and components are presented with assigned identifiers (e.g. Component2). Different kinds of interoperation activities between test components are covered, too.

Analysis of such a textual log traces is done after test execution and implies manual or automatic review of previously generated text. Manual analysis requires a lot of time and effort as people involved have to understand every single detail and also recognize possible doubts, misunderstands or errors. It is clear that in case of huge test campaigns big effort is needed to analyze log traces. Automatic form, on the other hand, requires creation of appropriate tool, which could be reasonable in some cases.

Our approach yet bases on the concept of graphical presentation of test execution which benefits of easy understanding of log traces and may be considered on-line, too. In the following, we discuss the graphical symbols we chose and will present the logging interface we defined.

Moreover, the aims of our work are to elaborate and use some concepts of logging and visualisation of test campaign execution in the context of standards like Testing and Test Control Notation – TTCN-3 and its Graphical Presentation Format – GFT .

TTCN-3 is a flexible and powerful language applicable to the specification of all types of reactive system tests over a variety of communication interfaces. Typical areas of application are protocol testing (including mobile and Internet protocols), service testing (including supplementary services), module testing, testing of CORBA based applications, API testing etc. TTCN-3 is not restricted to conformance testing and can be used for many other kinds of testing including interoperability, robustness, regression, system and integration testing. GFT represents graphically the behavioural aspects of TTCN-3 like the behaviour of a test case or a function.

Our approach is to show the user web graphics about all those processes which exist during testing. They present test cases and their execution results as a product of visualization. We should also be able to distinguish between different types of components existing during test process. Generally there exists always a *system* as a component being tested and additional components like: *mtc* - Main Test Component, *ptc* - Parallel Test Component and *control*. Moreover, different operations implicated by TTCN-3 code must be presented (i.e. send, call, create, stop etc). These must be shown in the Graphical User Interface by means of graphical symbols. Those symbols are derived from TTCN-3 and GFT Notations. Realization concepts are presented in section 4 of this paper.

Table 1 presents several graphical symbols, used within GFT diagrams, and comments their typical usage. All GFT symbols together are sufficient to illustrate test campaigns execution. Moreover, they specify test characteristic features. That is why GFT is the main driving force during designing of test log traces.


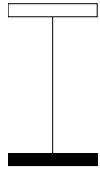




GFT Element	Symbol	Description
Port instance symbol		Used to represent port instances
Component instance symbol		Used to represent test components and the control instance
Start timer symbol		Used for TTCN-3 start timer operation, to be attached to a component symbol
Timeout timer symbol		Used for TTCN-3 timeout operation, to be attached to a component symbol
Stop timer symbol		Used for TTCN-3 stop timer operation, to be attached to a component symbol
Message symbol		Used for TTCN-3 send, call, reply, raise, receive, getcall, getreply, catch, trigger and check statement, to be attached to a component symbol and a port symbol

Table 1 Selected GFT symbols [3]

Table 2 introduces more complicated GFT graphics being determined by combination of single symbols. New more complex diagrams consist of several single graphics are gaining more meaning in the context of different activities appearing during test. GFT diagrams show the test campaign and are used as a graphical presentation of a test specification.

There are general drawing rules for the representation of some specific TTCN-3 syntax elements (semicolons, comments) defined. The way, how to display the execution of GFT diagrams and the graphical symbols associated with TTCN-3 language elements, is described in standards . Some of the rules concerning this kind of mapping can be found in tables 2 and 3, too.

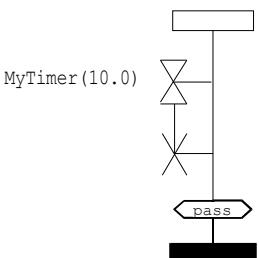
GFT	TTCN-3
 <p>MyTimer (10.0)</p>	<pre>MyTimer.start (10.0); MyTimer.stop; setverdict (pass);</pre>

Table 2 Connected start and stop timer symbols and Set local verdict

For the start/stop timer operations, the start/stop timer symbols shall be attached to the component instance. A timer name and an optional duration value (within parentheses) may be associated. The symbols for a start timer and a stop timer operation may be connected with a vertical line. In this case, the timer identifier needs only be specified next to the start timer symbol .

The verdict set operation setverdict is represented in GFT with a condition symbol within which the values pass, fail, inconc or none are denoted .

Referring to log traces concept, we were trying to use such general drawing rules also for the transformation of specific traditional textual log traces into appropriate graphics. We have adapted a tool presented and described in the last part of this paper.

4. Test Execution Logging Interface

The realisation of all mentioned issues requires a Test Execution Logging Interface which is presented in this paper.

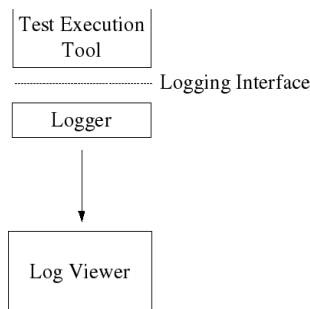


Figure 2 Logging Interface

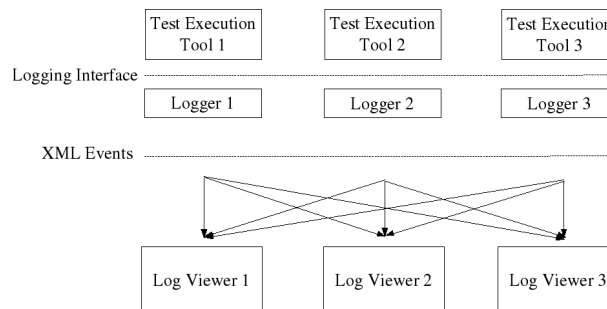


Figure 3 Universal XML based Log Event Format

The task of the *logging interface* is to define a generic API provided by the test execution tool (see Figure 2). The *Logger* is the client of this interface and handles the events produced during test execution. The information contained in the events is visualized through the *Log Viewer*. We should remark that the *Logger* implementation depends both on the *Log Viewer* technology and on the test tool technology.

Logging interface is a crucial part enabling observation of test log traces. It is a bridge between Test Execution Tool and *Logger*. It provides therefore all possible log events. Beside creation, start and termination of test components and ports several other events represent the operations done during the test case.

The most important are the execution (start / stop) of test cases, the timers within statements: start, stop, timeout. Furthermore such logs like:

- verdict information
- start/stop control
- logStatements, which implicates the position of user in the program code (phase)
- name of the protocol and its behavior
- (un)map
- (dis)connect
- start/stop of components
- start/stop of ports
- send/enqueue message
- template matching
- call, reply, getcall, catch, raise of synchronous operations
- interruptions (e.g. ext. function calls)

are of major importance.

Structure of Logging Interface influences additionally the *LogViewer* as the data format included in all considered parts must provide faultless data exchange between them. Moreover, this interface should be created by an advanced programmer due to its required flexibility, usability and possible universal application. This is the reason why we discussed also an alternative solution of that problem.

In a more generalized view, as shown in Figure 3, we consider a universal XML based layer which introduces unified data formats for the events consumed by the log viewers. The central notion behind using the XML event format is to prescribe a common prescriptive event in a consistent format so that tooling can be developed to support these goals. A common event format for log events is useful to allow the production of the same event types by different *Loggers*. Obviously, the *Log Viewer* must be capable to understand the XML event format. The big advantage of this approach is the independence between test execution tools and log viewer tools. The log events produced by an execution tool can be visualized with any log viewer tool which supports the XML event format.

We defined the XML event format with the help of XML Schema . The specification of the XML event format is basically the mapping one to one of the events defined in the Logging Interface and their representation in XML. All defined events follow a common format:

- the *tag name* represents the name of the event which is mapped from the corresponding operation in the Logging Interface.
- *global attributes*. There are several attributes which are common to all events. They are: *timestamp* which indicates when the event is produced, *source location* which indicates the file which contains the specification of the executed test, *line* in the source which points to the place where the event is produced, the *name*, the *id* and the *name* of the component which produces the event, and an *additional message* which completes the explanation of the event.
- *event content* which is a sequence of tags describing up to the event: components, ports, timers, testcases etc.

To get an insight of the technical details we provide as example the log event definition of the “execute” operation in TTCN-3. For “execute” event we define in the Logging Interface the tcExecute(...) operation. The signature below presents the IDL specification of the tcExecute. It has as parameters the common parameters: am (additional message), ts (time stamp), src (the file source of the test specification), line (the line code of the operation which creates the log event) and c (the component which creates the event) but has also three other specific parameters: tcId (the identifier of the test case to be executed), pars (the list of parameters required to start the test case) and dur (the maximal duration the test case is allowed to run).

```
void tcExecute(in string am, in long long ts, in string src, in long line, in TriComponentId
c, in TciTestCaseId tcId, in TriParameterList pars, in float dur);
```

This signature is mapped in XML Schema to the tc-execute tag. The content of tc-execute event is defined by its type TCExecute. TCExecute is an extension of the Event tag definition which contains the common parameters: am, ts, src, line, and component given by c-name, c-id, c-type.

```
<xsd:element name="tc-execute" type="Events:TCExecute"/>
<xsd:complexType name="Event" mixed="true">
  <xsd:sequence>
    <xsd:element name="am" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="ts" type="SimpleTypes:TimeStamp" use="required"/>
  <xsd:attribute name="src" type="SimpleTypes:SourceLocation" use="optional"/>
  <xsd:attribute name="line" type="xsd:integer" use="optional"/>
  <xsd:attribute name="c-name" type="SimpleTypes:QualifiedName" use="required"/>
  <xsd:attribute name="c-id" type="SimpleTypes:Id" use="required"/>
  <xsd:attribute name="c-type" type="SimpleTypes:QualifiedName" use="required"/>
</xsd:complexType>
```

The TCExecute has three specific parameters: tc-id defined by its type Types:TestCase, pars defined as Types:ParameterList and dur of type Types:Duration. The last two parameters pars and dur have the flag minOccurs="0" which means that they can be omitted.

```
<xsd:complexType name="TCExecute">
  <xsd:complexContent mixed="true">
    <xsd:extension base="Events:Event">
      <xsd:sequence>
        <xsd:element name="tc-id" type="Types:TestCase"/>
        <xsd:element name="pars" type="Types:ParameterList" minOccurs="0"/>
        <xsd:element name="dur" type="Types:Duration" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

An instance of a logging event, written according to the XML Schema presented, is presented below:

```
<tc-execute ts="1097247828" src="SIP_ProtocolTest.ttcn3" line="2044"
  c-name="c" c-id="component5" c-type="SIP_TestComponent">
  <am>start execution of a test case</am>
  <tc-id>SIP_CC_TE_CE_V_013</tc-id>
  <dur>5</dur>
</tc-execute>
```

The Logging Interface and the XML event format summarize 63 events which may occur during the execution of a test. It covers all TTCN-3 operations and notifies also the change of variables, verdicts or scopes.

5. Implementation

The visualisation tool example being presented in this paper bases on some GFT concepts. The focus of this tool is the graphical visualization of activities happening (on-line) or happened (off-line) during the execution of

TTCN-3 tests using TTthree and μ TTman [7]. The TTthree logging interface has been adapted to the TraceViewer tool [5] for TTCN-3 purposes. A Scalar Vector Graphics – SVG [6] plugin for Internet Explorer was used to display the graphical symbols.

SVG is a language for describing two-dimensional graphics and graphical applications in XML. It has two parts: an XML-based file format and a programming API for graphical applications. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled and transformed into previously rendered objects. Text can be in any XML namespace suitable to the application, which enhances searchability and accessibility of the SVG graphics. The feature set includes nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility. SVG drawings can be dynamic and interactive. The Document Object Model (DOM) for SVG, which includes the full XML DOM, allows for straightforward and efficient vector graphics animation via scripting. A rich set of event handlers such as onmouseover and onclick can be assigned to any SVG graphical object. Because of its compatibility and leveraging of other Web standards, features like scripting can be done on SVG elements and other XML elements from different namespaces simultaneously within the same Web page. This kind of graphic is used in many business areas including Web graphics, animation, user interfaces, graphics interchange, print and hardcopy output, mobile applications and high-quality design [9].

Such advantages like DOM for SVG or XML-based file format were used in our tool. They provide gaining appropriate structure of the logs data content as well as easy use of GFT symbols on the Web page. Graphical visualization of log traces was enriched in additional navigation bars, dynamic popup windows or colors change opportunities. In the tool there was also fluent parameters exchange observation and a possibility to hide some information, which may be too detailed for the analysis level. One could notice that some GFT symbols in the tool implementation were insignificantly simplified. However, they are still clear and should be seen as the original ones.

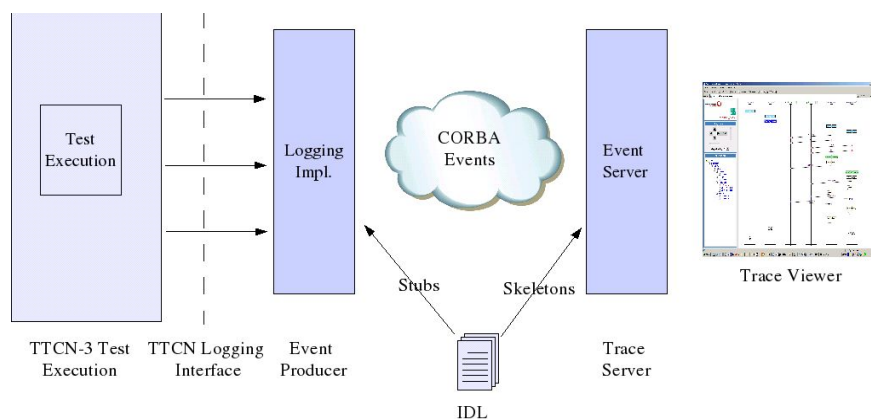


Figure 4 Implementation Architecture

The architecture of the visualisation tool is based on the open source TraceViewer and TraceServer [5], however events are produced on the TTthree side and tests are executed in μ TTman tool. TTCN-3 events in the form of appropriate test-logs are generated and sent to the TraceServer by mean of CORBA connection [2] (see Figure 4). The progress of the processes can be observed continuously. A special attention is put onto test specific concepts like test components, verdicts, defaults, etc. Logging Interface includes not only configuration requirements, but also all possible events (here: logs) implied by TTCN-3 actions (e.g. map request). Each test log corresponds with appropriate process occurring during test execution.

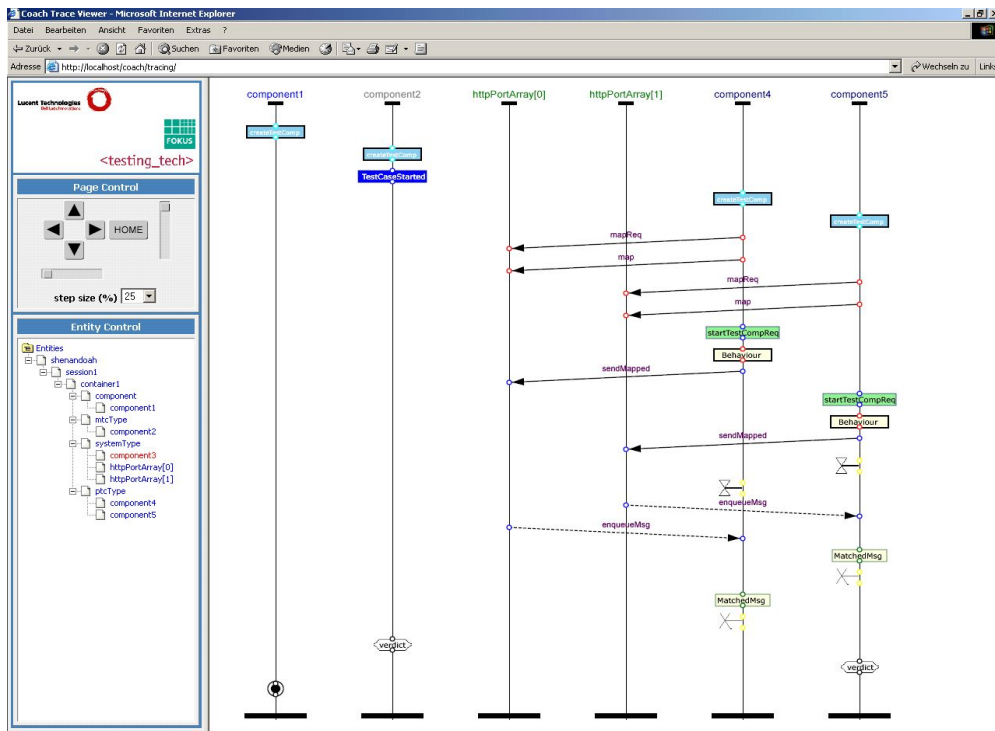


Figure 5 Screen shot from the TraceViewer

The screen shot given in Figure 5 is gathered from a real life example that is used to illustrate the Trace Viewer features in combination with TTthree. E.g. every test component and each interaction point (port) of the System under Test is represented by an instance line in the sequence diagram. It shows the execution traces of the test campaign. Selected activities observed during the execution of a TTCN-3 test suite are sent to the event server and represented in the visualization tool, e.g. creation of components and ports, sending of the message, start or termination of timers.

In particular, each test activity has its own graphical symbol in the tool. Components and their ports are described at the top of the life line. Actions being executed on the selected instances are visualised by rectangles while actions between components are presented in the form of arrows. Direction of communication is determined by continuous (from the source component) and dashed (to the source component (e.g. reply, receive)) lines. Timers and verdicts are created according to their GFT symbols specification. Termination of a component is marked with a verdict value. Test case ends when two circles (big – unfilled containing small – filled one) appear on the control component.

6. Conclusion

This paper presents concepts on the graphical visualisation of test execution traces as well as describes test execution logging. The visualisation can be done on-line during execution or off-line after the execution. The conceptual architecture and the implementation architecture of a graphical visualisation tool for TTCN-3 execution environments are discussed. The chosen graphical symbols and the test execution logging interface is considered and explained.

As far as prototype implementation is considered, application of special domains for different users would be a part of future work. It means – it is foreseen that people with diverse roles would run the tool using the functions which they need and only those. This would create a possibility to adapt to different requirements during testing process and to see only this information which is obligatory for clear conclusions. The second issue assigned as future work is integration of the TraceViewer in Eclipse. This would give opportunities to work in an environment enabling much more than only log events visualisation what is supported by Eclipse features and other plugins. Another challenge would be to develop a stable data base recording all the events Recording in a database would help for more complex statistical analysis of tests like finding out the average time of test execution, maximal or minimal value of response time. Last, but not least, test logging visualisation is not

restricted to TTCN-3 language. It is derived from GFT indeed, but kind of mapping of other languages to graphical symbols used in the TraceViewer could be also possible if demanded.

Acknowledgment

We would like to thank Axel Rennoch for his comments and technical support during the development of this paper.

References

- [1] ITU-T Recommendation Z.120 (1999): Message Sequence Chart (MSC). ITU Telecommunication Standards Sector, Geneva (Switzerland).
- [2] OMG CORBA v2.4: The Common Object Request Broker: Architecture and Specification, Section 3: Interface Definition Language (IDL). Object Management Group (OMG).
- [3] ETSI European Standard (ES) 201 873-1 V2.2.1 (2003-02 Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language
- [4] ETSI European Standard (ES) and Draft European Standard (DES) 201 873 (2002): The Testing and Test Control Notation version 3 (TTCN-3). Part 1 (ES): TTCN-3 Core Language. Part 3 (ES) Graphical Presentation Format for TTCN-3.
- [5] H. Batteram, W. Hellenthal, W. Romijn, A. Hoffmann, A. Rennoch, A. Vouffo Implementation of an Open Source Toolset for CCM Components and Systems Testing. In R. Groz, R. Hierons, editors, Testing of Communicating Systems, Lecture Notes in Computer Science (LNCS). Springer Verlag, 2004.
- [6] Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation, <http://www.w3.org/TR/SVG/>, 2003
- [7] TTthree and μ TTman, TTCN-3 compiler and execution environment, Testing Technologies, <http://www.testingtech.de/products/TTthree>
- [8] UML Testing Profile, Final Adopted Specification, <http://www.omg.org/cgi-bin/doc?ptc/2003-08-03>
- [9] C. Lilley, D. Jackson About SVG, 2d Graphics in XML, <http://www.w3.org/Graphics/SVG/About>
- [10] <http://www-users.cs.umn.edu/~raghaven/research/mansouri-samani92monitoring.pdf>
- [11] D. P. Helmbold, C. E. McDowell, "Traceviewer: A graphical browser for trace analysis," Tech. rep., Univ. Calif. Santa Cruz, UCSC-CRL-90-59, 1990.
- [12] Y. Howard, S. Gruner, A. M. Gravel, C. Ferreira, J. C. Augusto, Model-Based Trace-Checking
- [13] Eclipse: Open Source Integrated Development Environment, www.eclipse.org
- [14] XML Schema, XML Schema part [1-3] Primer, Structures and Datatypes, <http://www.w3.org/XML/Schema>
- [15] XML, Extensible Markup Language, <http://www.w3.org/XML/>
- [16] C. Anslow, S. Marshall, R. Biddle, J. Noble, and K. Jackson, XML Database Support for Program Trace Visualisation, Australasian Symposium on Information Visualisation, Christchurch, 2004.