

# The Graphical Format of TTCN-3 in the Context of MSC and UML

Ina Schieferdecker and Jens Grabowski

<sup>1</sup>FOKUS Berlin, Germany, [schieferdecker@fokus.fhg.de](mailto:schieferdecker@fokus.fhg.de)

<sup>2</sup>MU Lübeck, Germany, [grabowski@itm.mu-luebeck.de](mailto:grabowski@itm.mu-luebeck.de)

**Abstract.** Graphical system design techniques like Message Sequence Chart (MSC) and Unified Modelling Language (UML) are gaining more and more acceptance because they ease the development, understanding, and maintenance of software systems. In the testing area no accepted graphical test specification and implementation techniques exist. To overcome this shortcoming, a graphical presentation format for the Testing and Test Control Notation (GFT) has been defined. GFT supports the graphical design, implementation, visualization, documentation and tracing of test behaviour. GFT is based on MSC and extends it with test specific concepts like verdicts and defaults. GFT is also the basis for the definition of a UML testing profile to enable the integrated system and test development with UML models. This paper discusses GFT and its relation to MSC and to the UML testing profile.

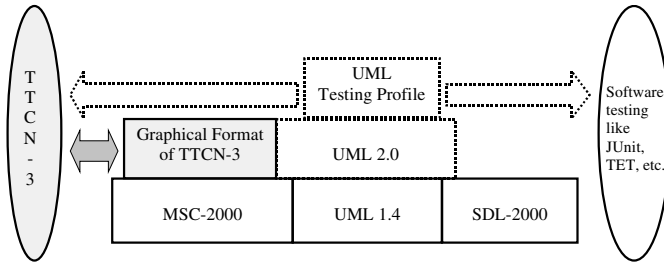
## 1 Motivation

It is well known that test development accounts for a significant portion of the overall effort required for the development of software systems. Consequently, research on testing has focused on how the development and validation of test suites can be made easier, faster and cheaper. To this end, test suite specifications have been hard to develop, to read and to understand. An alternative is the graphical specification of test cases.

This paper discusses two approaches for the graphical presentation and development of test cases, which have been developed on the basis of the Testing and Test Control Notation (TTCN-3 [1]): the Graphical Presentation Format for TTCN-3 (GFT [4]) and the UML 2.0 Testing Profile (U2TP [6]). TTCN-3 is the only standardised language for the specification and implementation of test cases. TTCN-3 has been developed to support black-box testing on the basis of behavioural interface specifications of a system under test (SUT). Black-box testing means that the SUT is considered to be a black-box, i.e., the internal structure of the SUT is not known. Stimuli are sent to the SUT and the responses from the SUT are observed and compared with the expected responses that are prescribed by the specification of the SUT [19], [20], [21], [22]. If expected and observed responses differ, then a fault has been discovered. A pair of stimuli and expected responses is termed a test purpose. A test case is an implementation of a test purpose. Implementation means that in addition to the pure stimuli-response-behaviour of the SUT, properties of the test system also have to be considered, e.g., distribution and co-ordination of test

components. TTCN-3 allows an easy and efficient description of complex distributed test behaviour. It is a modular language and has a similar look and feel to a typical programming language. However, in addition to the typical programming constructs, it contains all the important features necessary to specify test procedures and their control. There has been a requirement from the TTCN-3 users group to provide a graphical presentation formats for TTCN-3 test specifications. Consequently, the Tabular Presentation Format for TTCN-3 (TFT) [3] for the visualization of data and the Graphical Presentation Format for TTCN-3 (GFT) for the visualization of behaviour have been defined. Both formats are independent from each other and allow developing complete TTCN-3 specifications within each of the formats. This paper concentrates on GFT. While TTCN-3 copes solely with test specification and test implementation, the objective of the UML is to support the complete software development process including testing. For example, in [13] and [14] the use of UML to support test development has been investigated to encourage the parallel development of a test suite together with a system specification. The suggested test development activities are straightforward: the identification of the independent system component is followed by a definition of the test configuration, test case structure and test cases. An UML based test tool chain has been proposed by the AGEDIS project [15]. The approach is characterized by the goal to reuse and adopt existing system validation and test code generation tools. However, the support for the resulting test specifications in UML is in terms of well-defined test specification means rather limited. Therefore, the work on a UML 2.0 Testing Profile has been initiated. Both, GFT and U2TP allow the graphical development of test cases and support the graphical visualization and documentation of test cases as well as tracing of test case executions. The possibility to clearly discriminate between different parts of a test description and between different language constructs is one of the main points that are strongly in favour of using graphical test specifications. Contrary to that, the textual test descriptions appear to be quite homogenous, making it more difficult to get an idea about the essentials. Within graphical test specifications, language constructs can be graphically differentiated in order to increase the readability.

GFT and U2TP differ in several aspects. The main difference is that GFT is a presentation format for TTCN-3 having the full expressiveness of TTCN-3, while U2TP is the testing profile of UML being an integral part of UML following the UML concepts. There is no one-to-one mapping between TTCN-3 and U2TP. GFT can be used in “pure” test contexts as well as in combination with other system development approaches. U2TP can be used in integrated system and test development processes based on UML. Hence, GFT and U2TP serve different purposes and are targeted at different user domains. The relationship between GFT, MSC and U2TP is depicted in Fig.1. GFT is based on MSC and supports a one-to-one, bi-directional mapping to TTCN-3. U2TP will be based on UML 2.0. Both, UML 2.0 and U2TP are currently under development, so that the current status can be described only. U2TP will use not only UML 2.0 interaction diagrams (being based on selected MSC concepts), but will use also e.g. UML 2.0 state machines (being based on selected SDL concepts) and class and deployment diagrams from “pure” UML. There will be a mapping to TTCN-3 but not vice versa. Other mappings will support established software testing approaches like JUnit or TET.

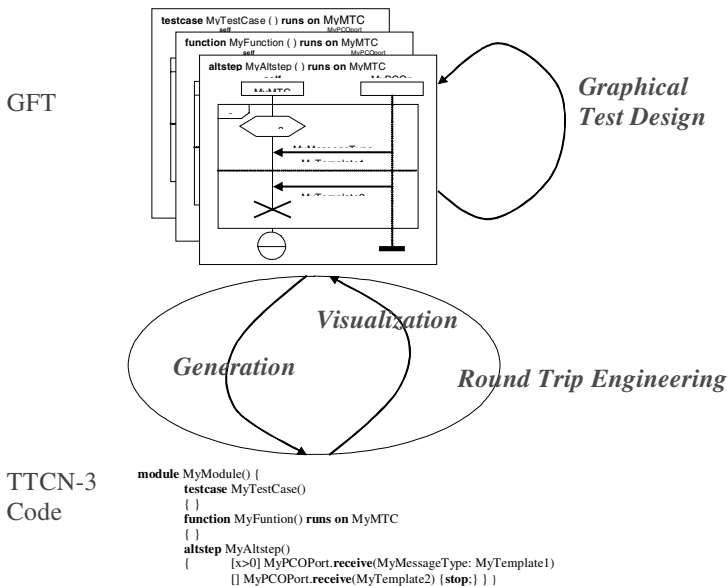


**Fig. 1.** Relationship between GFT, MSC and U2TP

The paper explores these relationships in more detail. Section 1 introduces GFT, Section 2 describes the relation of MSC and GFT and Section 3 discusses U2TP and its relation to GFT. A summary concludes the paper.

## 2 Overall View of GFT

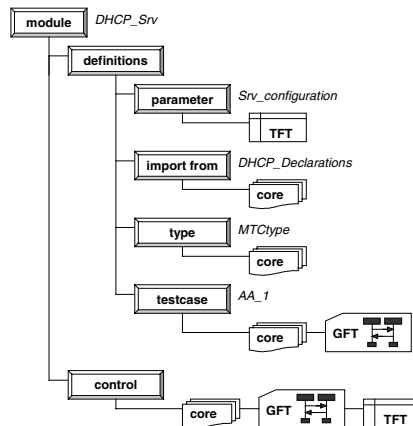
GFT is based on MSC [5] and supports the graphical presentation and definition of TTCN-3 test specifications. MSC is a trace language for the specification and description of the communication behaviour of system components and their environment by means of message interchange.



**Fig. 2.** Use of GFT

GFT is a test specific extension of a subset of MSC to widen the applicability of MSC in the testing process. In order to have adequate means for presenting TTCN-3 test

specifications with MSC, GFT adds test specific concepts of TTCN-3 such as port instances and test verdicts. The majority of extensions are textual extensions only, e.g., GFT diagrams include TTCN-3 keywords and statements. Graphical extensions are defined to emphasize test specific aspects in GFT diagrams. Where possible, GFT is defined like MSC, so that established MSC tools can be used for the graphical definition of TTCN-3 test specifications in terms of GFT. GFT provides a test system (test components) centered view, i.e., GFT diagrams describe how the test components send stimuli to the SUT, and observe expected responses from the SUT, as well as reacting to unexpected responses [17][16]. GFT concentrates on the local view on test component behaviours using basic concepts of MSC. This local view enables in particular a one-to-one mapping between GFT and TTCN-3 core such that TTCN-3 test specifications can be designed within GFT. A combined use of generation and visualization leads to round trip engineering for TTCN-3 (Fig.2): TTCN-3 code can be generated from GFT and visualized by GFT in an iterative way. GFT represents graphically the behaviour definitions of TTCN-3 like test cases or functions. It does not provide graphics for data aspects like, e.g., the declaration of types or templates. By allowing the usage of TTCN-3 code within GFT diagrams, it is possible to develop complete TTCN-3 modules with GFT. GFT also defines no graphical representation for the structure of a TTCN-3 module. The TTCN-3 module structure may be provided in form of an organizer view (Fig.3) or an MSC document-like presentation. An advanced tool may even support different presentations of the same object, e.g., the organizer view in Fig.3 indicates that the behaviour of the control part is available in TTCN-3 code, as TFT table and as GFT diagram.



**Fig. 3.** Organizer view of a TTCN-3 module

## 2.1 GFT Diagrams

GFT diagrams represent behaviour definitions of a TTCN-3 module. Such behaviour definitions are the control part of a module, test cases, functions and altsteps. Consequently, GFT distinguishes between control diagrams, test case diagrams, function diagrams and altstep diagrams. A GFT control diagram (Fig.4) provides a

graphical presentation of the control part of a TTCN-3 module. The heading of a control diagram consists of the keyword **control** followed by the module name. Initial variable declarations may also be found in the diagram header. A GFT control diagram only includes one instance (also called control instance) with the instance name **control** without any type information. The behaviour of the module control part is specified along the control instance.

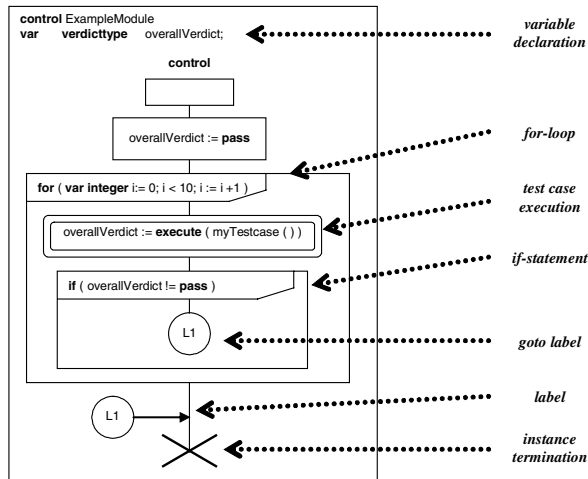


Fig. 4. GFT control diagram

GFT presents test case definitions in terms of a test case diagrams (Fig.5). The header of a test case diagram is the complete signature of the test case in TTCN-3 syntax, i.e., the keyword **testcase** followed by the test case name, formal parameter specification, and (optional) **runs on**- and **system**-clauses. Initial variable declarations may also be found in the diagram header. A GFT test case diagram includes one component instance, also called MTC instance, that represents the main test component (MTC) and one port instance for each port owned by the MTC. In order to distinguish the two kinds of instances graphically, port instances have a dashed instance line. Apart from the FIFO order defined by the TTCN-3 semantics for connections, no further event order is defined for port instances. For the MTC instance, the standard event ordering for an MSC instance is assumed. The name of the MTC instance is **mtc** and the (optional) associated type should be identical to the component type referred to in the runs on clause of the test case signature. The names of the port instances refer to the port names defined in the component type definition of the MTC and the (optional) type information has to be consistent with the component type definition of the MTC. Functions and altsteps are used to define and structure the behaviour of the test components and the behaviour of the module control part. In GFT, they are visualized in form of function and altstep diagrams. Both kinds of diagrams are very similar. Their heading is the interface of the presented definition and the diagram body includes one instance with the name **self** that either represents a test component or the module control. The body of a function or altstep diagram that defines or structures the behaviour of a test component (Fig.6) is identical to the body of a test case diagram.

## 2.2 GFT Presentations of TTCN-3 Statements and Operations

GFT provides graphics for the most significant TTCN-3 statements and operations. Some TTCN-3 constructs have no graphical representation, e.g., declarations, calls of functions without graphical presentation, or attributes associated to a GFT diagram. They have to be presented in action boxes and text symbols. TTCN-3 data types and values are brought into GFT using the mechanism to parameterise MSC with arbitrary data languages. Data is incorporated into GFT in a number of places, such as diagram parameters, control variables, verdicts, component and port instances, message values, timers, action boxes, or references. Data is used in two distinguishable ways either statically, such as in the parameterisation of a GFT diagram, or dynamically, such as in the acquisition of a value by a message receipt. All declarations, values, type definitions and data manipulations are specified using the TTCN-3 notation.

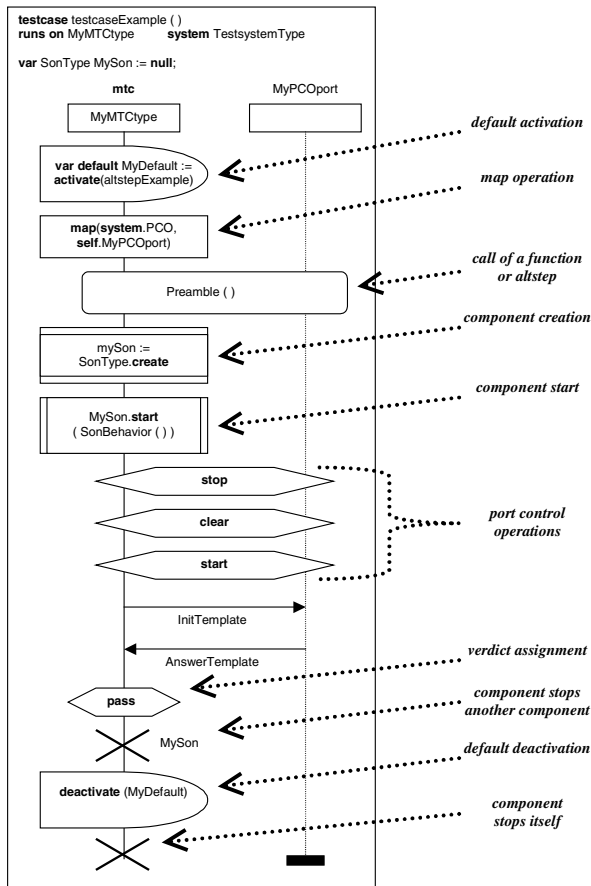


Fig. 5. GFT test case diagram

The GFT representations of the TTCN-3 configuration operations **create**, **map**, **connect**, **start** component, **stop** component, **start** port, **clear** port and **stop** port are

shown in Fig.5. The creation of a test component is described by a create symbol and the start of its execution is represented by a start symbol. The termination of a test component is described by means of two symbols: The MSC stop symbol is used if a test component stops itself<sup>1</sup> and a special component stop symbol is used, if a test component stops another test component. The component reference associated to a component stop symbol addresses the component to be stopped. Mapping and connecting ports have to be done by using the TTCN-3 map and connect operations within action boxes. Special **start**, **clear** and **stop** conditions are used to represent the TTCN-3 operations for controlling ports, i.e., start port, clear port and stop port. Within GFT, message-based communication is described by means of messages (Fig.6). Along GFT instance axes representing test components, a TTCN-3 **send** operation is represented by the origin of a message arrow and a TTCN-3 **receive** operation is described by an arrowhead. On top of the message arrow the message type may be given, and below the message arrow a TTCN-3 template has to be provided. A template may be defined in the definitions part of a TTCN-3 module and is then referenced in the GFT message, or is provided in terms of an in-line definition.

TTCN-3 supports procedure-based communications, where a test component can either play the role of a calling party or the role of a called party. For the calling party, TTCN-3 provides the **call** operation to call a remote procedure, the **getreply** operation to handle replies from remote, and the **catch** operation to catch exceptions triggered by the remote side. In GFT, these operations are modelled by means of messages (Fig.6). The name placed above the message arrow refers to the name of the remote procedure, and the operations are specified as prefixes to the message name. Values sent and received by the different operations are given in TTCN-3 syntax below the message arrow. TTCN-3 distinguishes between non-blocking and blocking procedure-based communication. In GFT, blocking **call** operation is described by using inline expressions that group all messages belonging to the call and suspension regions. Dashed timer symbols are used to describe the (optional) time guard of a blocking call. For the specification of non-blocking calls, blocking areas and inline expressions are omitted. An example for a blocking call operation is shown in Fig.6. The test component **self** calls *MyProc*. The call is guarded with a duration of *20ms*. After the call, the test component either receives a reply, catches an exception or the guarding timer expires. For the called party, TTCN-3 provides the **getcall** operation for accepting a call from remote, the **reply** operation to reply to an accepted call and the **raise** operation to raise an exception if required by the testing situation. In GFT, these operations are again represented by messages with the operation names as prefixes to the message names. In addition, an activation region may be used to indicate the flow of control that belongs to the handling of the accepted call. In Fig.6, the test component accepts the call of procedure *MyProc* by means of a **getcall** operation. Depending on the evaluation of the Boolean expressions in the guarding conditions, the test component replies to the call, or raises an exception. GFT uses the MSC timer symbols and provides a one to one mapping for the TTCN-3 timer operations **start**, **stop** and **timeout** to the corresponding MSC symbols set, reset and timeout. The TTCN-3 timer operations running and read have no graphical

---

<sup>1</sup> As shown in 0, the MSC stop symbol is also used to represent the stopping of the module control.

counterpart in GFT. They have to be specified in action boxes or in guarded conditions.

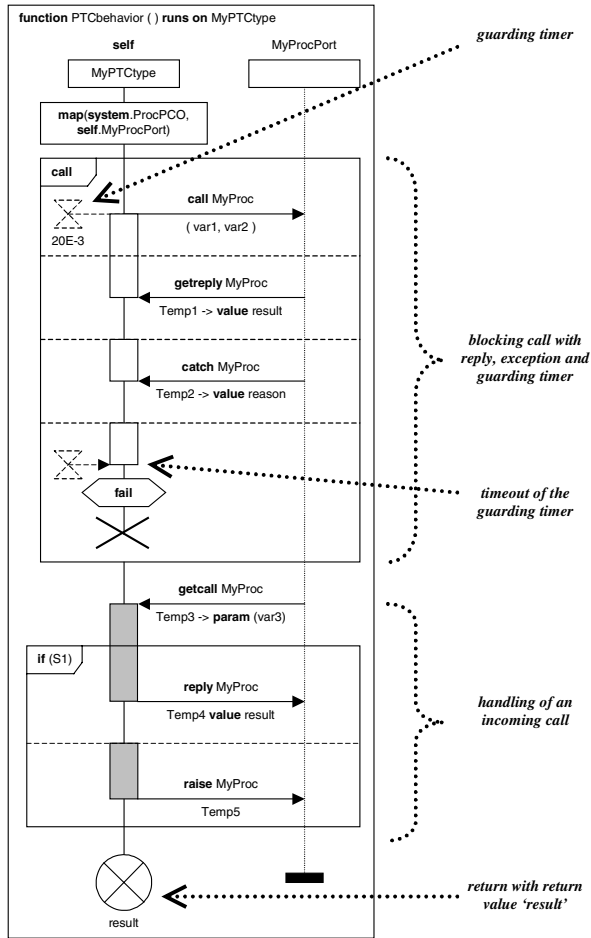


Fig. 6. GFT function diagram

In TTCN-3, the verdict of a test component is handled as a special object whose value can only be accessed by the operations **setverdict** (for setting the verdict value) and **getverdict** (for retrieving the actual verdict value). To emphasize the importance of verdicts, GFT uses conditions, containing the verdict value as special keywords for setting component verdicts. An example for a verdict set operation is described in Fig.5. A **pass** verdict is assigned after the reception of an *AnswerTemplate* message. For retrieving the verdict value, the **getverdict** operation can be used within an action box. In TTCN-3, a special default mechanism is used to handle unexpected or exceptional behaviour of the SUT during the test run. The mechanism is based on altsteps that define the default behaviour of a test component or the module control. An altstep can be activated as default behaviour and may afterwards be deactivated



using activate and deactivate statements. GFT provides a default symbol to emphasize the activation and deactivation of defaults. An example for the activation and deactivation of default *altstepExample* is given in Fig.5.

### 2.3 Alternative, Cyclic, and Interleaved Behaviour

TTCN-3 provides several possibilities to describe alternative, cyclic and interleaved behaviour. In GFT, all possibilities are represented using MSC in-line expressions with new operators that correspond to the TTCN-3 statements. The TTCN-3 statements for describing alternative behaviour are if-else and alt. Examples of the GFT representation of alternative behaviour are provided in Fig.4 and Fig.6. GFT represents the Boolean guards of the alternatives in an alt statement by using guarding conditions. The matching conditions of the receiving operations are described in the templates of the messages, calls, replies and exceptions to be received. For the presentation of an else branch within an alt statement, a guarding condition with the special keyword else has to be used. Cyclic behaviour refers to the TTCN-3 loop statements for, while and do-while. In GFT all loop statements are represented by means of inline expressions with the corresponding loop operator, i.e., GFT introduces the operators for, while and do-while for inline expressions. The different exit criteria for the loops are part of the loop operator. An example for the GFT representation of a **for** loop can be found in Fig.4. TTCN-3 also provides a possibility to describe interleaved behaviour by means of the interleave statement. Therefore, GFT provides an interleave operator for in-line expressions to represent interleaved behaviour in an appropriate way.

### 2.4 Invocation of Test Cases, Functions, and Defaults

Altsteps and functions can be called in other GFT diagrams by using the MSC reference symbol. The reference symbol should cover all component, control and port instances known by the called function or altstep. Fig.5 shows the call of a *Preamble*, which may either be provided in form of a function or an altstep. GFT provides an **execute** symbol to emphasize the execution of test cases. The control diagram in Fig.4 includes one execute symbol. It specifies the execution of the test case *myTestcase* within a loop. The TTCN-3 statements **return**, **repeat**, **goto** and **label**<sup>2</sup> allow specifying the transfer of control. GFT provides special symbols to represent these statements. The return symbol with an optional associated return value underneath is used to describe the termination of a function graphically (Fig.6). It specifies the return of control and an optional return value to the calling entity, i.e., module control, test case, altstep or function. The repeat symbol is used to force a new snapshot and the re-evaluation of an alt statement. It may be used within an alt statement or for describing the termination of an altstep. The latter case causes a new snapshot and the re-evaluation of the alt statement in which the altstep has been invoked.


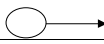




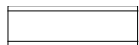



---

<sup>2</sup> The symbols **goto** and **label** allow to specify the transfer of control within a GFT diagram. The example in 0 describes the transfer of control to label *L1*, if value of variable *overallVerdict* is not equal to **pass**.

### 3 GFT and MSC

The MSC language is a graphical means for describing the behaviour of distributed reactive systems in form of traces. A main advantage of the MSC language is its clear graphical layout, which immediately gives an intuitive understanding of the described behaviour. Using MSC as a presentation format for TTCN-3 considerably improves the readability of test cases and makes them more understandable<sup>3</sup>. Although GFT uses most of the graphical MSC symbols, the inscriptions of some MSC symbols have been adapted to the needs of testing and, in addition, some new symbols have been defined in order to emphasize test specific aspects. New GFT symbols have been defined e.g. for the representation of port instances and the creation of test components. The new symbols are summarized in Table 1.

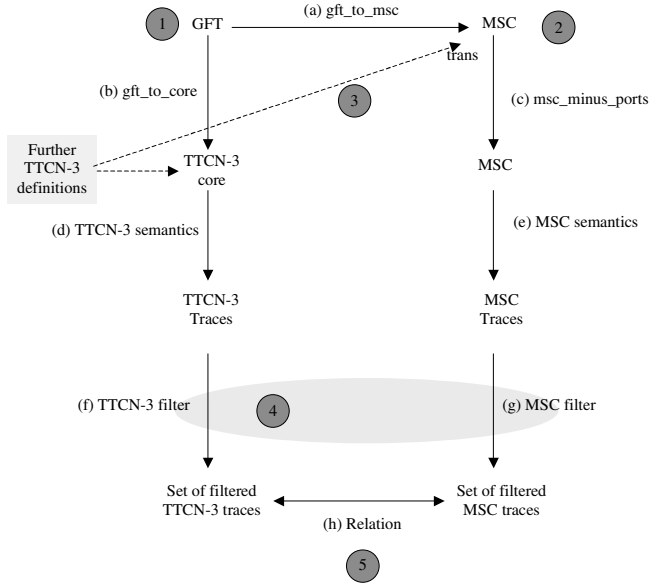
**Table 1.** Special GFT Symbols, extending MSC

GFT Element	Symbol	Description
Port instance symbol		Used to represent port instances
Labelling symbol		Used for TTCN-3 labelling and goto, to be attached to a component symbol
Goto symbol		Used for TTCN-3 labelling and goto, to be attached to a component symbol
Default symbol		Used for TTCN-3 activate and deactivate statement, to be attached to a component symbol
Return symbol		Used for TTCN-3 return statement, to be attached to a component symbol
Repeat symbol		Used for TTCN-3 repeat statement, to be attached to a component symbol
Create symbol		Used for TTCN-3 create statement, to be attached to a component symbol
Start implicit timer symbol		Used for TTCN-3 implicit timer start in blocking call, to be within a call inline expression and attached to a component symbol
Timeout implicit timer symbol		Used for TTCN-3 timeout exception in blocking call, to be within a call inline expression and attached to a component symbol
Execute symbol		Used for TTCN-3 execute test case statement, to be attached to a component instance symbol

In addition, specific inline expressions for **for**, **while**, **do-while** and **call** statements have been added. MSC has been extended in order to enable the definition of features specific to TTCN-3 like port instances and defaults as well as to improve the readability and to highlight test specific aspects like verdicts or test case execution. Besides the extended graphical means, GFT is based on the TTCN-3 semantics.

<sup>3</sup> It should be pointed out that GFT is not intended as a standalone language, but as a presentation and visualization means for TTCN-3 test behaviour only.

However, the semantic relation between GFT and MSC can be defined along the following mappings:



**Fig. 7.** Semantic relation between GFT and MSC

The GFT and MSC diagrams (1 and 2 in Fig.7) refer to a set of diagrams describing the behaviour of one test component or module control only. The TTCN-3 information needed in MSC (3 in Fig.7) is related to the data managed and manipulated by the specified test component or module control. The filtering operations (4 in Fig.7) on the traces will be different: on the MSC side only non-TTCN-3 events are filtered away, while on the TTCN-3 side the filtering is related to the events described in the GFT diagram and to one component only. This means, if several components use the same function (represented as a GFT diagram), the traces produced by each component have to be investigated separately. In result of these mappings, a simple ‘subset’ relation between the filtered TTCN-3 traces and the filtered MSC traces holds: the set of traces on the TTCN-3 side might be bigger due to the specifics of TTCN-3 default mechanism and component termination. The relationship between GFT and MSC is summarized in Table 2.

## 4 GFT and U2TP

The UML 2.0 Testing Profile (U2TP) defines a way to specify test procedures unambiguously within UML. It is dedicated to enable the integrated system and test development by using wherever possible information from the system model within the test model. Specific objectives are to base the UML Testing Profile on existing black-box test technologies, in particular on TTCN-3, and to enable the abstract test specification for established software testing methods.

**Table 2.** Summary of GFT and MSC relation

Objective	GFT	MSC
	TTCN-3 test behaviour specification, tracing and documentation	General purpose trace language for system specification, design, simulation, testing, and documentation
Differences in Concepts	Control, test case, function and altstep diagrams	Basic and high-level MSCs, MSC documents
	Local, test component view only	Local and global view
	Communication between test component and port instances only	Communication between instances and environment (gates)
	TTCN-3 data	Open data interface
	Variables belonging to test component instances and to GFT diagrams	Variables belonging to instances only
	Specific actions for component handling	Instance creation and termination
	Specific actions for default handling	---
	Specific constraints for verdict handling	---
	Alt, if, for, while, do-while, call inline expression	Alt, opt, exc, loop inline expression
Semantics	TTCN-3 traces	MSC traces
	Comparable	

#### 4.1 An Overview on U2TP

So far, UML technology has focused primarily on the definition of system structure and behaviour and provides limited means for describing test procedures. However, with the approach towards system engineering according to model-driven architectures with automated code generation, the need for solid conformance testing, certification and branding has increased. Therefore, a proposal for a UML Testing Profile (U2TP) has been initiated by FOKUS. It solicits proposals for the following:

- A U2TP based upon the UML metamodel,
- that enables the specification of tests for structural (static) and behavioural (dynamic) aspects of computational UML models, and
- that is capable of inter-operation with existing test technologies for black-box testing.

IBM, Ericsson, FOKUS, Motorola, Rational, Softeam, Telelogic and the University of Lübeck formed a consortium to develop this U2TP, which will be developed in three stages:

1. The concept space together with the terminology.
2. The meta-model for the concept space aligned with the UML 2.0 meta-model.
3. The syntactic elements for the stereotypes of the UML testing profile.

The work will be based on recent developments in testing such as the results from TTCN-3 [2] and COTE [10]. The initial intend to base the U2TP just on GFT could not be taken directly as additional requirements from software testing together with the alignment with UML required additions and generalizations to the concepts of TTCN-3. However, the U2TP will be defined such that a mapping onto TTCN-3 is possible in order to enable the reuse of existing TTCN-3 infrastructures. Major generalizations are

- The separation of test behaviour and test evaluation by introducing a new test component: the arbiter to evaluate the system observations and to calculate the verdict. This enables the easy reuse of test behaviour for other testing kinds without changing the test behaviour but just the arbiter. This concept is comparable to the evaluate function of TSSL [11].
- The integration of the concepts test control, test group and test case into just one concept of a test case, which can be decomposed into several lower level test cases. This enables the easy reuse of test case definitions in various hierarchies. A test suite is then just a top-level test case. This concept is comparable to the test object concept of TSSL.
- The support of data partitions not only for observations, but also for stimuli. This allows describing test cases logically without having the need to define the stimulus data completely but as a set or range of values. This concept is comparable to the concept of Test Data Definitions (TDD) of ADL [12].

Furthermore, some additions will widen the practical usability of the U2TP:

- An initial test configuration will be used to describe the initial setup of the test components and the connectivity to the SUT and between each other
- Component and deployment diagrams will be used to enable the definition of software components realizing a test suite and their requirements regarding test execution on certain nodes in a network

The different background of the U2TP members has lead to an intensive discussion on the basic set of terms. A number of topics allow alternative views. We will address these issues in an overview on the actual terminology. It has been agreed to distinguish three major groups of terms:

- Test architecture, i.e. the elements and their relationship which are used in a test,
- Test data, i.e. the structures and meaning of values to be processed in a test, and
- Test behaviour, which address the observations and activities during a test.

The *test architecture sub package* covers the concepts for specifying test components, the interfaces of and connections between test components and to the system under test. Test components are active entities within the test system, which perform the test behaviour defined in a test case (the Test Behaviour sub package) by using test data as defined in the test data sub package. The *test architecture* is a set of related classes and/or components for which *test contexts* with test configurations and test cases may be specified. The *test configuration* is the collection of test components and connections between the test components and to the SUT. The test configuration defines both (1) test components and connections when a test case is started (the initial test configuration) and (2) the maximal number of test components and connections during the test execution. The initial test configuration is the precondition for a test case to be executed. A test component is an active object within a test system performing the test scenario. A test component has a set of interfaces via which it may communicate with other test components or with the SUT via its connections when the interfaces are connected. In addition to test components, *utility parts* can be used to denote helper and miscellaneous parts for testing.

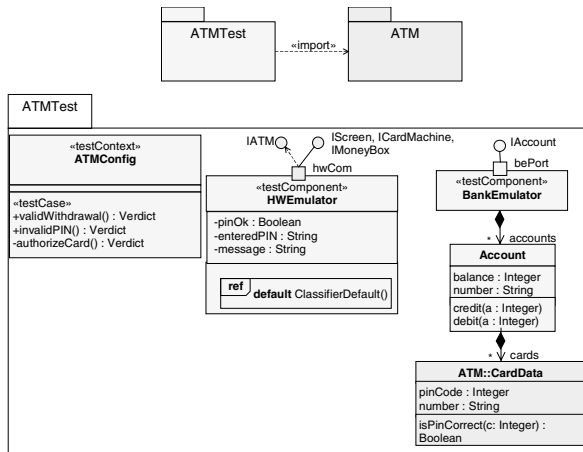


Fig. 8. An example test architecture

An *arbiter* is a specific test component to evaluate test results and to assign the overall verdict of a test case. There is a default arbiter for functional, conformance testing, which generates pass, fail, inconc, and error as verdict, where these verdicts are ordered as pass < inconc < fail < error. The *system under test* (SUT) is characterised by the set of interfaces via which a real SUT can be controlled and observed during testing (by means of communication). An SUT can be on different abstraction levels: a complete system, a subsystem thereof, a single component, object or even a class. An example is given for a bank automaton - an ATM. The bank automaton offers various interfaces, in particular, a port to the bank network and interfaces to the user to insert and withdraw a bankcard as well as to take the money. The test objective is to check that debit account is possible provided that enough funds are available. An example test configuration for a bank automaton is shown in Fig.8. The test case is defined for *ATM* being the SUT. The test package *ATMTest* imports the definition from the *ATM* SUT and defines the test context *ATMConfig* as well as the classes for the test components *HWEmulator* and *BankEmulator*. The test context is used to define the test cases *validWithdrawal()*, *invalidPIN()* and *authorizeCard()* and the test configuration, which is the internal structure of test context. It uses two test components: a bank emulator *be* and a hardware emulator *hw*. A utility part *current* is used to represent the bankcard used during the tests. The test components are connected with the SUT via interfaces.

The *test data sub package* covers the concepts for *data sent* to the SUT and *received* from the SUT. Mechanisms in order to change and compare test data are used to enable precise and succinct test specifications. Data can be concrete (i.e. a specific value) or abstract (i.e. a logically described set of values) Logical partitions are used to define such value sets within test parameters. Coding rules are part of the test specification and denote the encoding and decoding of test data. With the help of coding rules, the interfaces of the SUT can be bound to certain encodings such as for CORBA GIOP/IOP, IDL, ASN.1 PER or XML. In the ATM example, different messages to and from the SUT are used. They are declared in the class diagram of the ATM

```

enterPin(in PIN:string {length = 4})
messageDisplay(in Message:string)
enterAmount(in Amount:integer)
deliverMoney(in Amount:integer)

```

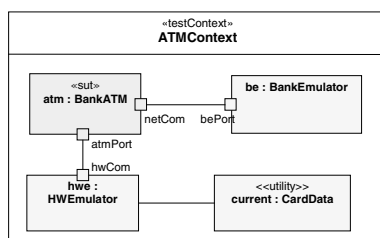
In the test behaviour, concrete data is used for example like

```

enterPin(validPIN())
messageDisplay("EnterPIN")
enterAmount(x:=SufficientFunds())
deliverMoney(x)

```

Here, *validPIN* is a characterization for a PIN to be valid and provides a valid PIN for test execution. *"EnterPIN"* as well as *"DebitAccount"* are concrete values. Also, *SufficientFunds()* is a logical characterization. The selected value is bound to *x*, so that it can be reused in the subsequent test behaviour, i.e. in order to check that the correct amount of money is delivered. Although the basic concepts for test data are defined, the details of their realization within U2TP are open – in particular as UML provides limited means for data modelling only. For example, further investigations are required for exception handling, pattern matching and value binding.



**Fig. 9.** An example test configuration

The *test behaviour sub package* defines as the top-level notion a test context and a related collection of tests. In contrast, in [1] a 'test suite' comprises a set of 'test cases'. However by considering the term test case as a set of cooperating test components allows e.g. to combine two test cases into one test case, which makes use of the two original test cases. From this understanding the latter view (i.e. without the dedicated "top level" test suite) is more flexible and useful in the context of e.g. integration testing or test deployment and therefore used by the U2TP. A *test case* is a specification of one case to test the system, including what to test with which input, result, and under which conditions. It uses a concrete technical specification of how the SUT should be tested - the *test behaviour*. A test case is the implementation of a *test objective* for a particular test configuration, which is defined by the test behaviour. A test case uses an arbiter to evaluate the outcome of its test behaviour. A test objective is a general description of what should be tested. The test behaviour is the specification of behaviour performed on a given test configuration, i.e. sequences, alternatives, loops and defaults of stimuli to and observations from the SUT. Test behaviours can be defined by any behavioural diagram of UML 2.0, i.e. as interaction diagrams or state machines. There can be a designated main test behaviour for a given test configuration. By invocation, test cases can make use of other test behaviours. TTCN-3 Verdicts such as **pass** and **fail** are predefined as the outcome of a test case. In addition, user defined verdicts and arbiters can be used to denote specific test outcomes for example for performance tests. Every test component handles a local

verdict. When a test component terminates, its verdict is reported to the arbiter for calculation of the overall verdict for the test case. A validation action is an action to evaluate the status of the execution of a test scenario by assessing the SUT observations and/or additional characteristics/parameters of the SUT. A validation action is performed by a test component and sets the local verdict of that component. U2TP uses implicit validation actions when matching the observed responses from the SUT to the expected ones. A match leads implicitly to **pass** while a mismatch leads to the current series of defaults. Defaults can be defined on four levels: individually for events in interaction diagrams or for states in state machines, for test component parts of test configurations, for test components of a specific class or for all test components in a test system, i.e. the *metadefault()*. These defaults are evaluated in sequence – from the event default up to the metadefault. The metadefault defines the deferral of any event that cannot be matched to a subsequent evaluation. During the execution of a test case a test trace is generated. It is the log of a test case execution containing logs for each action performed during that test case execution as well as containing the test result of that test case execution. A log action can be used to log additional information in the test trace. Parts of the test behaviour for an example test case are shown below. A sequence diagram is used here to denote the information exchange between the test components and the SUT.

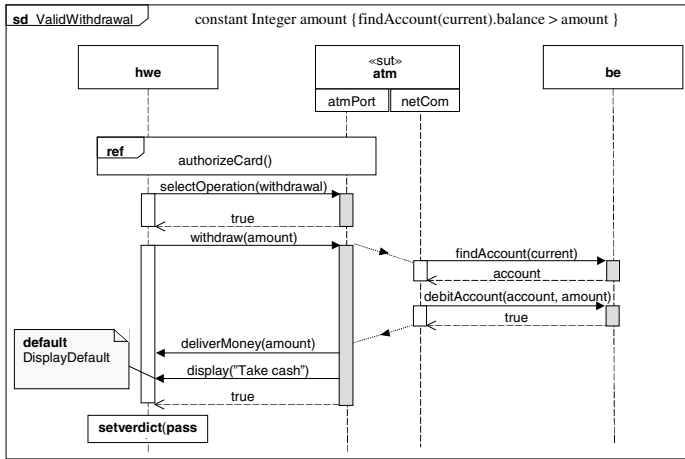


Fig. 10. An example test case

This test case depicts the test for a valid withdrawal of money: after authorization of the bankcard (by referencing to the *authorizeCard* sequence diagram) the *withdrawal* operation is selected and an *amount* requested, which is smaller than the balance on the card. This is defined by a logical partition with a constraint on amount (see top right of the figure). The SUT then interacts with the bank emulator *be* to debit the account and delivers the money afterwards. An event specific default *DisplayDefault* is used for the display event in order to handle different display messages specifically. Finally, the verdict is set to **pass**. Implicit assumptions ease the definition of the test



behaviour. For example, a timer<sup>4</sup> is implicitly started whenever a response from the SUT is awaited. If the timeout occurs without any reaction from the SUT, a **fail** verdict is assigned – this is handled by the metadefault. Furthermore, a pass is assigned whenever the received response matches the expected one.

## 4.2 The Relationship between GFT and U2TP

MSC (being the basis of GFT) forms a central constituent of UML 2.0 and is employed for the formalisation of Interaction Diagrams (being both sequence diagrams and activity diagrams). Since there is no accepted test notation in UML yet, this was an ideal opportunity to bring TTCN-3 in form of GFT to the attention of the UML world. In this context, a graphical format is of particular importance since UML is exclusively based on graphical modelling techniques. In fact, GFT is the archetype for U2TP. U2TP uses several concepts being developed in GFT: sequence diagrams are for example considered to be the first choice to define test behaviour, but not the only one as state machines can also be used. Specific symbols are defined for defaults and verdict handling.

Still, GFT and U2TP differ in several respects: U2TP is based on the object oriented paradigm of UML, where behaviours are bound to objects only, while GFT is based on the TTCN-3 concept of functions and binding of functions to test components. U2TP uses additional diagrams to define e.g. the test architecture, test configuration and test deployment. Test behaviour can be defined as interaction diagrams but also as state machines. While GFT supports dynamic configurations in terms of kind and number of test components and the connectivity to the SUT and between test components<sup>5</sup>, U2TP uses static configurations where only the number of test components may vary but not the structure of the connections between test components. In addition, U2TP has only one FIFO queue per test component, while GFT uses a FIFO queue per test component port. The verdict handling in GFT is bound to the well-established verdict handling of conformance testing, while U2TP uses in addition the ability of user-defined verdicts and the arbitration of verdicts, i.e. the definition of algorithms of when and how verdicts are determined. Additional validation actions can be used to calculate local verdicts of test components by the use of external information from the test execution context. Another difference is that of default handling for unexpected or irrelevant behaviour from the SUT: GFT uses function-based defaults which can be dynamically activated and deactivated during test execution, while U2TP uses structural defaults, which are bound to the structure of a test system – from test component level down to event/state level – leading to a defaults hierarchy and less dynamic default handling. Although the data part is not yet on a solid basis it is clear that U2TP will support UML data only, i.e. primitive types and classes, while GFT supports all types available in TTCN-3: basic types, user-defined structured types (record, record of, set, set of, enumerated, union), and anytype. In addition, any imported data like ASN.1 or IDL is supported. Overall, GFT

---

<sup>4</sup> Please note that UML 2.0 currently does not define a timer concept, so that U2TP will add timers, timeouts, and the starting, stopping and reading of timers.

<sup>5</sup> In TTCN-3 and hence in GFT, ports can even be connected, reconnected, started, stopped and cleared during test execution, which leads to dynamic test configurations in terms of connectivity between test components and to the SUT.

covers all TTCN-3 features while U2TP is targeted at UML with restricted means for TTCN-3. A mapping from U2TP to TTCN-3 will be possible but not the other way around. The relationship between GFT and U2TP is summarized in Table 3.

**Table 3.** Summary of GFT and U2TP relation

	<b>GFT</b>	<b>U2TP</b>
<b>Objective</b>	Graphical presentation format of TTCN-3 with a one-to-one mapping to TTCN-3 code	Integrated system and test development in UML
<b>Basis</b>	TTCN-3 and MSC	UML 2.0 package, class, interaction and state machine diagrams
<b>Differences in Concepts</b>	Function-based without inheritance	Object oriented
	Dynamic test configurations	Static test configurations
	Test components with several FIFO queues per port and specific queue handling operations	Test components with single FIFO queue only, limited queue handling only (e.g. defer)
	Fixed verdict semantics	Verdict validation and arbitration
	Function-based default behaviours	Structure-based default behaviours
	TTCN-3 type system and other imported data languages	UML class diagrams and other profiled data
<b>Expressiveness</b>	Full expressiveness of TTCN-3	Limited TTCN-3 expressiveness

## 5 Summary and Outlook

MSC and dialects thereof are used throughout the engineering process of test development: for the specification of test purposes to define the specific objective of a test, via the specification of test cases to define the concrete test behaviour, up to the visualization of test executions – despite the fact that no standardized graphical test notation for test purposes, test behaviour and test traces exist yet. Therefore, several test specific, proprietary extensions to MSC have been proposed and are used to make it applicable in a testing context. Within the area of conformance testing, MSC is already well established for the specification of test purposes, and as such for the automatic generation of TTCN test cases [8][9]. Beyond that, MSCs have been proposed for a selected visualisation of TTCN descriptions by means of simulation techniques [7]. Although MSC has been used for limited test specification in the past, the latest version of the language now contains constructs that make the comprehensive MSC specification of test suites feasible. Such language constructs include MSC composition, object oriented modelling, as well as data. These enabled the definition of GFT. Furthermore, MSC and dialects in general are often used by commercial test devices and test platforms to represent test traces, for example, MSCs are used by Agilent BSTS or NetTest InterWatch. In addition, proprietary extensions towards test behaviour definition are used for example by Tektronix G20.

Both, GFT and U2TP are aimed at well-defined specification means for testing in order to close the gap between practical needs of graphical test specification and the lack of standardized approaches to do that. GFT is tailored towards the representation and visualization of TTCN-3 test behaviour. The first advantage of GFT is highlighting test specifics with special graphics to ease readability and improve the understanding. The second advantage of GFT in comparison with TTCN-3 refers to the description of the communication behaviours between test components and their ports. Within GFT all ports may be represented by different port instances. Consequently the test events belonging to different ports are clearly separated

visually, in contrast to TTCN-3 core notation, where all events appear in a mixed form.

We described the extensions that are needed to MSC in order to facilitate the adequate representation of TTCN-3. GFT supports a bi-directional mapping which is defined in form of an executable mapping using the functional programming language, Standard ML of New Jersey (SML/NJ)<sup>6</sup>. Firstly, both the TTCN-3 and GFT grammars are represented as separate SML data types on the basis of which a set of mapping functions that map the GFT data types onto the TTCN-3 data types for each GFT object (i.e. test case diagram, test step diagram, control diagram, and function diagram) and the other way around. One purpose behind this activity on an executable mapping has been to aid the validation of the graphical grammar and language concepts of GFT. Future work will reconsider the global view to denote the interaction of test components within test behaviours and the use of enhanced high-level MSCs – the concept of hyperMSCs [16] – for GFT. Commercial tool support for GFT is already available. Existing MSC tools can be used for test specification purposes<sup>7</sup> [8], but the first new test tools supporting the whole GFT definition will be available on the market soon [18].

Furthermore, with the interest in MSC within OMG, as a potential candidate for UML v2.0, there was a further possibility to use GFT as a basis for the UML testing profile. The integration of system and test related information supports independent test laboratories in their work, but also the system engineers to perform the test runs by their own. The status of the basic test concepts and terminology, which have been presented in this paper, could be regarded as a consensus of different test experts working in heterogeneous IT fields like object-oriented systems or telecom protocols. Fundamental elements of test architecture, test data and test behaviour have been collected and applied exemplarily using different UML diagrams. A comparison with the established concepts of the test notation standard TTCN confirms the suitability of the selected definitions in the UML profile for testing. The UML testing profile language elements can be mapped to TTCN-3 but not vice versa. One problem is for example the lack of timers in UML 2.0. At the time of writing this paper the work on the UML profile for testing is still ongoing in OMG and no final decisions have been made. Due to the dependencies on the UML2.0 release it is expected that the final submission of the UML profile for testing will not be available before 2003. Nevertheless the importance of testing with UML has to be elaborated earlier to assist its acceptance.

## References

- [1] ISO/IEC 9646-3 (1998): "Information technology – Open systems interconnection – Conformance testing methodology and framework – Part 3: *The Tree and Tabular combined Notation (TTCN)*"
- [2] ETSI ES201873-1v220 (2002) *TTCN-3: Core Language*.

---

<sup>6</sup> SML/NJ is open source and is freely available from Bell Laboratories, (<http://cm.bell-labs.com/cm/cs/what/smlnj/>).

<sup>7</sup> If MSC editors are used, specific graphical GPF symbols have to be modelled by using keywords, naming conventions or TTCN-3 core notation in other symbols.

- [3] ETSI ES201873–2v220 (2002) *TTCN-3: Tabular Presentation Format*.
- [4] ETSI DES201873–3v220 (2002) *TTCN-3: Graphical Presentation Format*.
- [5] ITU-T Recommendation Z.120 (2000): *Message Sequence Charts (MSC)*.
- [6] OMG: *The Initial Submission to the RFP on the UML Testing Profile*, April 2002.
- [7] J. Grabowski, T. Walter. *Visualisation of TTCN Test Cases by MSCs*, SAM98, Proc. of the 1st Workshop on SDL and MSC, Humboldt University Berlin, 1998.
- [8] P. Baker, C. Jervis, D. King: *An Industrial use of FP: A Tool for Generating Test Scripts from System Specifications*. – Scottish Functional Programming Workshop/Trends in Functional Programming.
- [9] J. Grabowski, D. Hogrefe. *TTCN, SDL- and MSC-based specification and automated test case generation for INAP*. Proc. of the 8th Intern. Conf. on Telecommunication Systems (ICTS'2000) – Modeling and Analysis", Nashville, March 2000.
- [10] C. Jard, S. Pickin: *COTE – Component Testing using the Unified Modelling Language*. – ERCIM News No.48, January 2002.
- [11] T. Vassiliou-Gioles, M. Li, I. Schieferdecker, M. Born, M. Winkler: *Configuration and Execution Support for Distributed Systems*. – IWTCS'99, Budapest, Hungary, Sept. 1999.
- [12] The Open Group: *ADL 2.0 Translation System*, 1998. <http://adl.opengroup.org/>
- [13] M. Born, I. Schieferdecker, M. Li: *Test Framework for Component-Based Systems*, Intern. Workshop on Distributed System Validation and Verification (DSVV'2000), Taipei (Taiwan), April 2000.
- [14] M. Born, I. Schieferdecker, M. Li: *UML Framework for Automated Generation of Component-Based Test Systems*. – Intern. Conf. on Software Engineering Applied to Networking and Parallel/ Distributed Computing (SNPD'00), Reims, France (2000).
- [15] C. Crichton et al.: *Using UML for Automatic Test Generation*: Proc. of the Intern. Conf. On Automated Software Engineering, ASE'2001.
- [16] E. Rudolph, I. Schieferdecker, J. Grabowski: *HyperMSC – a Graphical Representation of TTCN*. Proc. of the 2nd Workshop on SDL and MSC (SAM'2000), Grenoble (France), June, 26 – 28, 2000.
- [17] E. Rudolph, I. Schieferdecker, J. Grabowski: *Development of an MSC/UML Test Format*. FBT'2000 – Formale Beschreibungstechniken für verteilte Systeme (Editors: J. Grabowski, S. Heymer), Shaker Verlag, Aachen, June 2000.
- [18] Testing Technologies. TT Tool Series. <http://www.testingtech.de/products>
- [19] ANSI/IEEE. *Glossary of Software Engineering Terminology*. ANSI/IEEE Std 729–1983, ANSI/IEEE Std 729–1983, 1983.
- [20] B. Beizer. *Software Testing Techniques Second Edition*. Van Nostrand Reinhold New York, 1990.
- [21] G. J. Myers: *The Art of Software Testing*. John Wiley, 1979.
- [22] Sommerville: *Software engineering*. Addison Wesley, 1989