# A Message Sequence Chart-Profile for Graphical Test Specification, Development and Tracing – Graphical Presentation Format for TTCN-3

Paul Baker[1], Jens Grabowski[2], Ekkart Rudolph[3], Ina Schieferdecker[4]

[1]Motorola (UK) Research Laboratory, Motorola Labs, Basingstoke, Hampshire, ENGLAND, Email: Paul.Baker@motorola.com

[2]Universität zu Lübeck, Institut für Telematik, Ratzeburger Allee 160, D-23538 Lübeck, GERMANY, Email: jens@itm.mu-luebeck.de

[3]Technische Universität München, Institut für Informatik, D-80290 München, GERMANY, Email: rudolphe@informatik.tu-muenchen.de

[4]GMD FOKUS, CC TIP, Kaiserin-Augusta-Allee 31, D-10589 Berlin, GERMANY, Email: schieferdecker@fokus.gmd.de

### *Abstract*

Recently, international standard bodies approved the third evolution of the TTCN test specification language (TTCN-3) as a requirement to modernise and widen its application beyond pure OSI conformance testing. Even though TTCN-3 makes the description of complex distributed test behaviour much easier there is still a requirement from the user community to provide a visualisation means for test specification, development and tracing. Message Sequence Charts (MSC) appeared to be a particularly attractive candidate as a graphical means for visualising TTCN-3 test cases. Therefore, in addition to the pure textual core language, TTCN-3 also defines a MSC profile called the Graphical Presentation Format for TTCN (GPF). This paper describes some of the main concepts of GPF, its use and application to real test suite examples.

## 1 Introduction

In general, test development still accounts for a significant portion of the effort required during the development of software systems; sometimes accounting for as much as 50% of the overall effort. Consequently, much research has focused on how the development and validation of test suites can be made easier and quicker. A first step towards systematic test development has been done with test case specifications allowing the ratification of test suites at industrial consortia, the unambiguous implementation of test suites and the comparison of test results. In this case, we refer to the usage and evolution of the standardised test specification language TTCN, as an independent and formalised means of describing functional test specifications. TTCN has been specifically developed to support black-box testing on the basis of behavioural interface specifications of the System Under Test (SUT). Black-box testing means that expected outcomes, as prescribed by the specification of the SUT, are compared with observed outcomes as produced by executions of the SUT [1, 2, 14, 18]. If expected and observed outcomes differ, then a fault has been discovered.

However, practise has demonstrated [3, 10] that users gain further benefits (e.g. improved validation) when using graphical languages during test specification and execution. In this case, the Message Sequence Chart (MSC) language [22] appeared to be a particularly attractive candidate as a graphical means for visualising test specifications and test traces. Where, MSC is comparable to UML Sequence Diagrams (SDs) [15, 21], but in addition to the pure graphical representation provided by SDs, MSC also has a textual representation and a well-defined semantics. The purpose of MSCs is to show how sequences of messages are interchanged between system components, or processes within a system component and their environment. Therefore, MSCs are used throughout the system development cycle: (1) for capturing requirements, (2) in combination with other description languages during system specification, (3) as a basis for implementation, and (4) for the description of black-box tests in the testing phase.

Consequently, the third edition of the TTCN language (TTCN-3) [5] not only defines a textual representation, but it also allows the definition of other presentation formats. At present, two presentation formats are defined: a tabular presentation format (TPF) [6] that resembles the tabular format of previous TTCN editions, and an MSC-based graphical presentation format (GPF) [4, 7]. In order to have an adequate means for representing test cases graphically, within GPF certain test-specific extensions to MSC, such as port instances and test verdicts etc. are necessary. This also allows for the visual tracing of test case executions.

The rest of the paper is organized in the following manner: An introduction to TTCN-3 and MSC is provided in Section 2. The details of GPF are presented in Section 3. An example based on the Dynamic Host Configuration Protocol (DHCP) can be found in Section 4. Finally, an outlook to future work is given in Section 5.

# 2 An overall view of TTCN-3 and MSC

This section provides an introduction to TTCN-3 and MSC. Both languages form the basis for GPF.

## *2.1 TTCN-3*

TTCN-3 is a language to define test procedures to be used for black-box testing of distributed systems with well defined interfaces. As shown in Figure 1[1], TTCN-3 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 campaigns.

The principle building block of the TTCN-3 core language is the *module*. A module is a self-contained and complete specification, i.e. it can be parsed and compiled as a separate entity. A module consists of a *module definitions part* (lines 2 – 36 in Figure 1), and an (optional) *module control part* (lines 37 – 44). The module definitions part specifies the top-level definitions of the module. These definitions may be used elsewhere in the module, including the control part or be imported from other modules. The module control part describes the execution order (possibly repetition) of the actual *test cases*. Test cases are defined or imported from another module in the module definitions part and then executed in the module control part.

A test case is defined in form of stimuli and expected responses, i.e., stimuli are given to the SUT, the reactions are observed and compared with the expected ones. On the basis of this comparison, the subsequent test behaviour is determined, a *test verdict* of either **pass**, **inconclusive** or **fail** is assigned or the test case ends.

A TTCN-3 test case (e.g. lines 14 – 36 in Figure 1) is executed by one or more *test components*. Each test case has a *Main Test Component* (MTC), which automatically is created and started when the test case is invoked. A test case ends when the MTC terminates. The behaviour of the MTC is defined in the test case body (lines 15 – 36). Further test components may be created by the MTC or other already running test components. Each test component has an associated *component type* (lines 4 – 13) that defines *variables*, *constants*, *timers* and *ports* local to each instance of that type. The type of the MTC is referenced in the test case header (line 14).

The interfaces of a test component are defined by means of *ports*. Basically, a port is a FIFO queue to be used for communication purposes. A port may support either *asynchronous communication* by means of asynchronous message exchange or *synchronous communication* in form of remote procedure calls. The information exchanged at a port has to be specified in an associated *port type definition*.

TTCN-3 provides a variety of *communication operations* and *port control operations*. Communication operations are: **send** for the sending of messages (e.g. line 18 in Figure 1), **receive** for the reception of message (lines 21 – 22), **trigger** to filter a message from a message stream, **call** for the invocation of remote procedures, **getcall** to accept procedure calls from remote, **reply** to send replies for accepted calls, **getreply** to receive replies, **raise** to send exceptions, **catch** to handle the reception of exceptions and **check** to examine the top element of a port. Ports can be started, stopped and cleared by means of the port control operations **start**, **stop** and **clear**.

The operations **receive**, **trigger**, **getcall**, **getreply**, **catch** and **check** are specified together with the expected values. They are executed only, if the received test data, e.g. message value, call record of a remote procedure or value of an exception, matches the expected values. TTCN-3 offers the *template* mechanism for the specification of test data. A template may be defined in the module definitions part for being used by reference in other places of a module or in-line, i.e. at the place of its usage. In Figure 1 references to the templates `DICOVER_s1` and `OFFER_r1` (line 18 and line 21) imported from module `DHCP_declarations` (line 3) are used. A template defines a single value or a whole

[1] A detailed description of the test behaviour specified in Figure 1 will follow in Section 4.

range of values of a specific type. For the specification of value ranges *matching mechanisms* are provided which are comparable to regular expressions. Templates defining a single value can also be used for the description of values to be sent by means of the operations **send**, **call**, **reply** and **raise**.

```
1    module DHCP_Srv (boolean Srv_configuration){
2        // --- Module definitions part
3        import all from DHCP_declarations;   // import from another module
4        type component MTCtype {             // component type definition
5            port   DHCP_LT    LTPCO1;
6            var    OCTET_4    cltid1;
7            var    SrvID_Opt  srvidopt1;
8            var    boolean    blvalue;
9            var    OCTET_4    yip1;
10           var    OCTET_4    lease1;
11           var    OFFER_rxed offer_message;
12           timer  Tshort := Tshort_value;
13       }   // end MTCtype
14       testcase AA_1 () runs on MTCType {       // test case definition
15           activate (Default_1);                // activation of a default
16           Pre_1 ();                            // function call
17           cltid1 := select_xid ();
18           LTPCO1.send (DICOVER_s1(cltid1, BROADCAST_Flag, HAddr1));
19                                                // sending a message
20           Tshort.start;                        // start of timer Tshort
21           LTPCO1.receive (OFFER_r1(cltid1, BROADCAST_Flag, Haddr1))
22               -> value rxed_offer_message;     // reception of a message
23           yip1 := rxed_offer_message.yiaddr;
24           srvidopt1 := rxed_offer_message.opts.lease.time;
25           lease1 := rxed_offer_message.opts.lease.time;
26           Tshort.stop;                         // stop of timer Tshort
27           if ((yip1!=NullAddr)and(srvidopt1.addr!=NullAddr)and(lease1!=ZeroTimer)) {
28               verdict.set (pass);              // setting of the test verdict
29               Post_1 (cltid1, HAddr1, yip1, srvidopt1);
30           }
31           else {
32               verdict.set (fail);
33           }
34           deactivate (Default_1);              // deactivation of a default
35           stop;                                // termination of the MTC
36       }   // end testcase AA_1
37       control {  // begin module control part
38           if (Srv_configuration) {
39               execute(AA_1);                   // execution of test case AA_1
40           }
41           else {
42               execute(AA_2);                   // execution of the imported testcase AA_2
43           }
44       }   // end module control part
45   }   // end module DHCP_Srv */
```

Figure 1 **TTCN-3 module**

TTCN-3 uses *defaults* to handle unexpected behavior of the SUT. A default is defined in form of a *named alternative* and can be activated and deactivated in a test case. A named alternative is a special TTCN-3 macro mechanism. In line 15 of Figure 1 the named alternative `Default_1` is activated as default and in line 34 it is deactivated. `Default_1` is imported from `DHCP_declarations`.

The test configuration of test case `AA_1` in Figure 1 is not distributed. The MTC port `LTPCO1` of type `DHCP_LT` (cf. line 5) is implicitly created when the test case is invoked and afterwards is used for the asynchronous communication with the SUT.

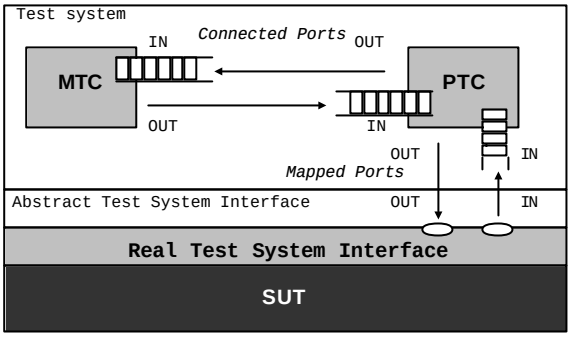



Figure 1 **TTCN-3 module**

In case of a distributed test configuration, a *test system interface* in form of a component type definition has to be referenced in the test case header. The test system interface defines the interface for the SUT. When a test case using a distributed test configuration is invoked, the MTC is created implicitly together with its ports and may create further test components by means of a **create** operation. The ports of all newly created components are also instantiated implicitly and have to be connected (**connect** operation) to the ports of running test components or mapped (**map** operation) to the test system interface. A **connect** operation instantiates a connection between two test components and allows their communication. A **map** operation makes the port of a test component visible to the test system interface, i.e. allows the communication between the test component owning the port and the SUT. After the creation of a component and the connection and mapping of its ports, a test component is started by means of a **start** operation. The *component behavior* is defined by a function reference in the **start** operation.

Figure 2 shows a distributed test configuration with two test components: an MTC and a *parallel test component* (PTC). Communication among the test components is performed via connected ports and communication with the SUT is done via mapped ports. Figure 2 also shows the distinction between *test system*, *abstract test system interface*, *real test system interface* and *SUT*. Test system and abstract test system interface are defined in the scope of TTCN-3. The real test system interface connecting the TTCN-3 runtime system and the SUT has to be provided by the test equipment.

In addition to the typical features of programming languages, further TTCN-3 language constructs allow the specification and handling of timer (e.g. line 20 and line 26 in Figure 1), the description of interleaved test behaviour, the use of external constants and functions, the import of data types and data values from other languages, e.g. ASN.1, C++ or IDL, and to provide encoding information for test data. A complete and detailed description of TTCN-3 can be found in [5], [8] or [11].

## *2.2 Message Sequence Charts*

The Message Sequence Chart (MSC) language is a graphical means for describing the behaviour of distributed reactive systems in form of traces. Syntax and semantics of MSC are defined by the International Telecommunications Union – Telecommunications Standards Sector (ITU-T) in Recommendation Z.120 [22]. The MSC language comprises two sorts of diagrams: MSC diagrams and High Level-MSC (HMSC) diagrams.

MSCs [2] describe the interaction of entities of a distributed system (Figure 3). The entities are called *instances* and are represented by vertical lines with an instance head to which an *instance name* and (optionally) an *instance type* are associated.The basic model of interaction in MSC is that of asynchronous communication by means of message passing between instances or between instances and the environment. *Messages* are represented by arrows. The sending of a message is described by the arrow origin and the consumption of a message is described by the arrow head. *Message name* and (optionally) *message parameters* are specified near the corresponding message arrow. The *environment* of an MSC is represented by the frame around the diagram area. Communication with the environment is described by message arrows starting or ending at the environment frame. In addition to asynchronous communication, synchronous communication can be modeled in form of method invocation by means of a message representing the invocation that ends at the beginning of a method symbol and a corresponding reply message (represented by an arrow with a dotted tail) that starts at the end of the *method symbol*. Between a call and the corresponding reply an instance may be blocked. This can be specified by means of an *suspension region*.

[2] In the following the abbreviation MSC refers to the MSC language or an MSC diagram. In case of ambiguities we will use the terms MSC language and MSC diagram instead.

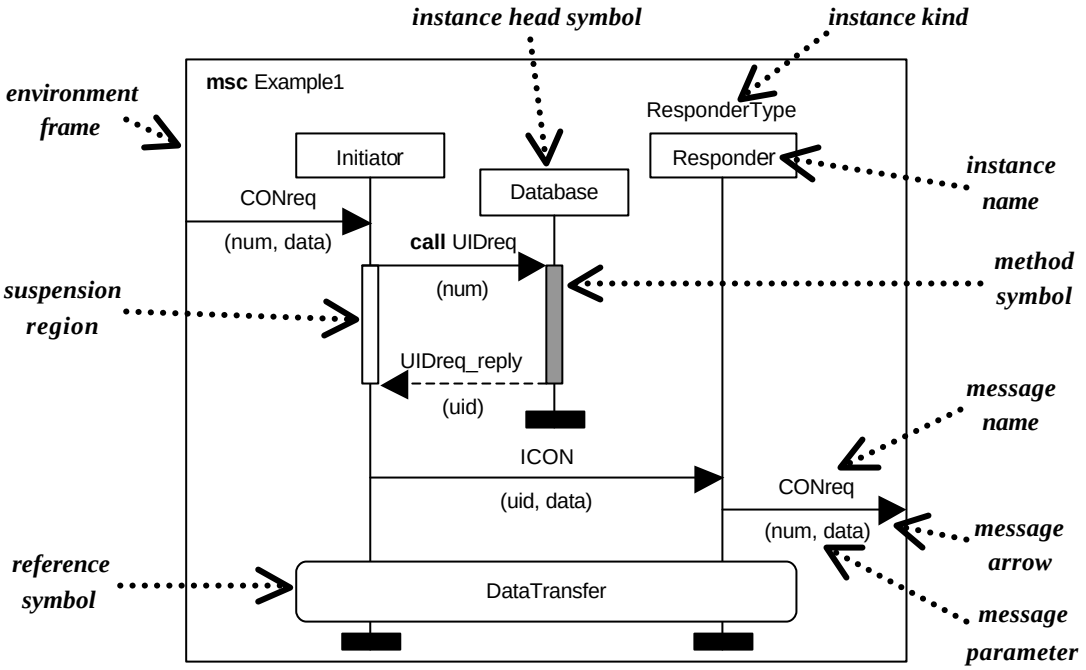



**Figure 3** MSC `Example1`

As shown in Figure 4, further instance behavior can be described with other MSC language constructs for *action*, *timer set*, *timer reset*, *timeout*, *instance create*, *instance stop* and *coregion*. A coregion specifies that all events in the coregion can happen in arbitrary order. The *instance end symbol* only is a graphical means to describe the end of an instance behavior definition in the diagram and does not specify the termination of an instance.

Outside of a coregion, all events along an instance axis are ordered in time from top to bottom. Events on different instance axes are in general not ordered. The only order for events on different instances is implied via messages, i.e., a message must be sent before it is consumed.

MSC *conditions* (Figure 4) are the means to describe system states. Conditions may describe local states of one instance, non-local states of several instances or global states of all instances in an MSC. In addition to state representation, conditions may also be used as *guards* containing Boolean expressions.

Composition of event structures may be defined inside an MSC by means of *inline expressions*. The operators of inline expressions are **alt**, **loop**, **par**, **exc** and **opt**. The **alt** operator defines alternative MSC sections, i.e. only one of them will be executed, the **loop** operator defines the iteration of an MSC section, the **par** operator describes the parallel execution of MSC sections, the **exc** operator allow to specify exceptional MSC behavior and the **opt** operator indicates optional MSC sections. An example for an inline expression with an **alt** operator can be found in Section 4.

MSC *references* provide the possibility to structure an MSC into several MSCs. The meaning of an MSC reference (Figure 3) is defined by another MSC with the name of the reference inscription.

HMSCs provide the possibility to compose MSCs and HMSCs to complex behavior structures. Language constructs of HMSCs are *start*, *end*, *flow line*, *connector*, *parallel frame*, *reference* and *condition*. An example for an HMSC (without a parallel frame) is shown in Figure 5[3].

An HMSC diagram can be read like a flow graph. The detailed behavior is referenced by means of MSC references and global states reached during the execution of the trace are described by means of conditions. The behavior description begins with the start symbol and by following the flow lines the behavior is constructed from the meaning of the visited MSC references. The parallel frame allows the parallel composition of MSCs. Connectors are only graphical means for branching and joining of flow lines. End symbols denote the end of a trace.

[3] The keyword **execute** in the MSC reference is not part of the standard MSC syntax. It is a GPF extension to denote the execution of test cases. The figure will be re-used for the DHCP example in Section 4.

MSC also supports data descriptions. The MSC approach for including data was not to define a special MSC data language, but instead to provide an interface by which the data syntax can be checked and the MSC semantics can be evaluated. This allows the use of arbitrary data languages within MSC.

MSCs and HMSCs can be collected in an *MSC document* (Figure 6). The document mechanism is also used for GPF and, therefore, will be explained in Section 3.1.

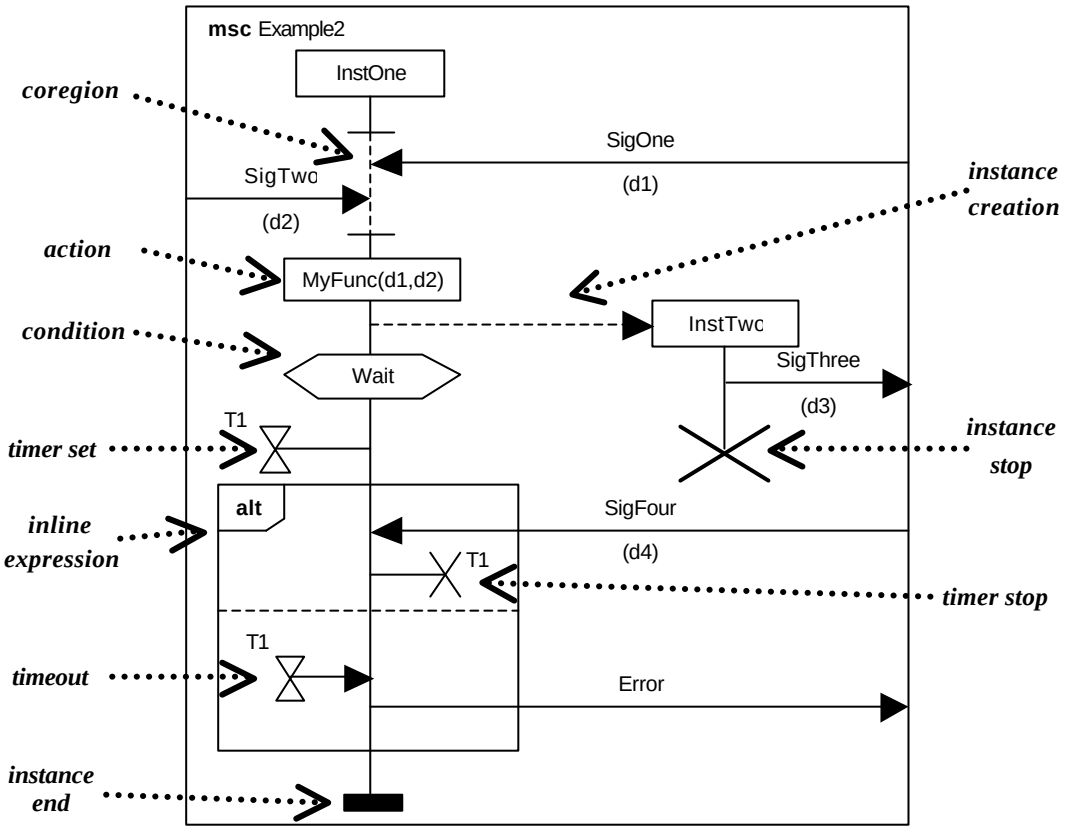



**Figure 4** MSC `Example2`

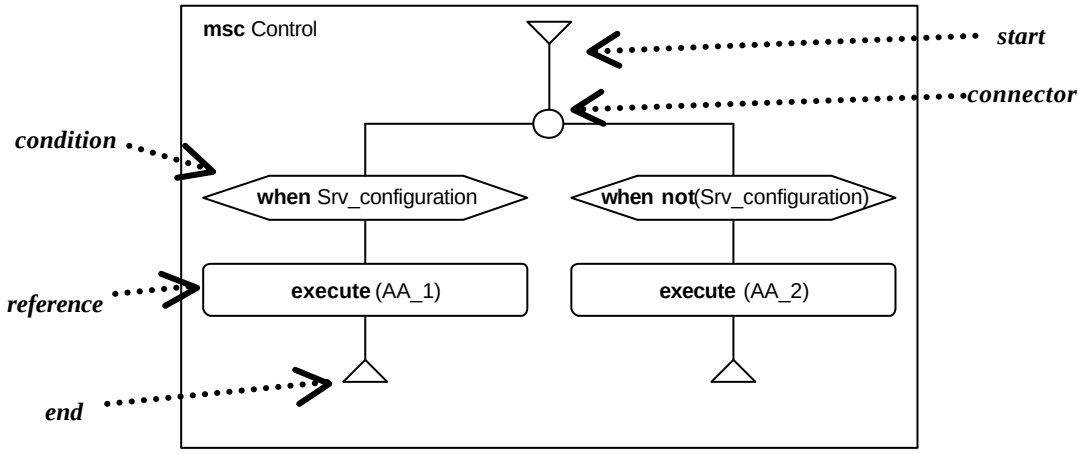



**Figure 5** HMSC `Control`

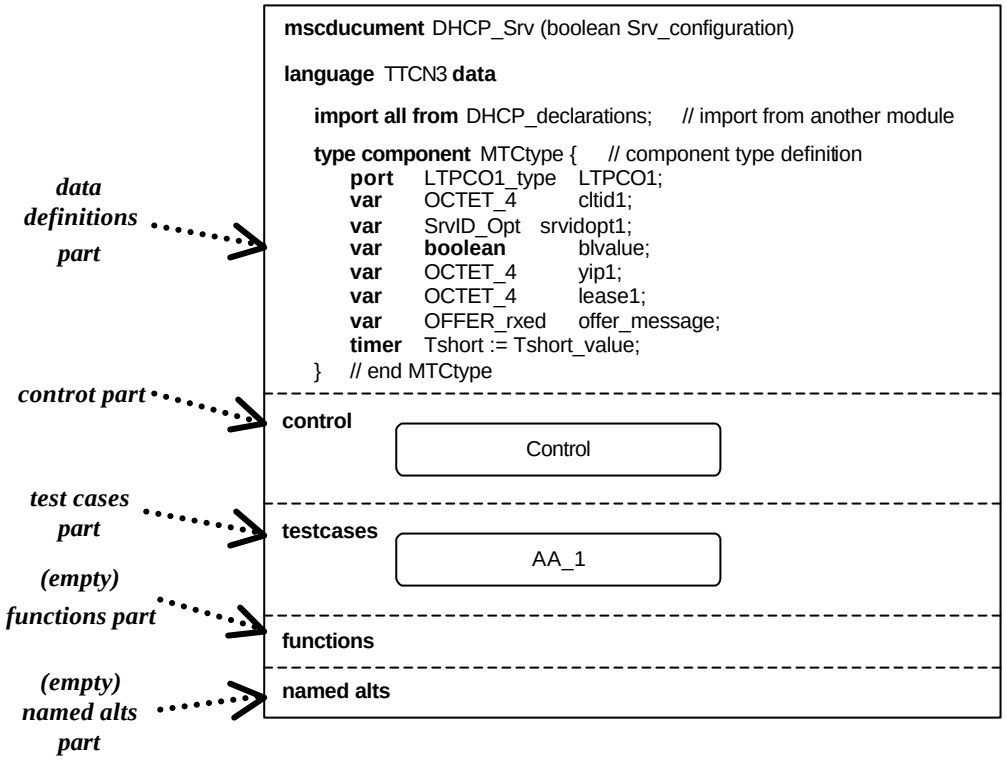



**Figure 6** MSC document `DHCP_Srv`

# 3 The Graphical Presentation Format for TTCN-3

GPF is designed to describe TTCN-3 test behavior. Therefore GPF diagrams represent module control, functions, test cases and named alternatives as the means for TTCN-3 test behavior descriptions. All GPF diagrams of a TTCN-3 module are collected in a document. This section provides an overview of the GPF extensions to MSC in order to obtain a sufficient graphical presentation of TTCN-3 test behavior.

## *3.1 MSC documents and TTCN-3 modules*

In TTCN-3 all data and behavior definitions are collected in a module. A TTCN-3 module also has a (optional) module control part that defines the order and conditions under which the test cases shall be executed. In GPF, the counterpart of a TTCN-3 module is the MSC document.

Figure 6 provides an example for a MSC document with the name `DHCP_Srv`. Dashed lines structure the document into a *data definitions part*, a *control part*, a *test cases part*, a *functions part* and a *named alts part*.

Data and type definitions can be found in the data definitions part. MSC diagrams representing module control, test cases, functions and named alternatives are referenced by means of references in the control, test cases, functions and named alts part. Similar to TTCN-3 modules MSC documents may also be parameterized.

## *3.2 Data*

TTCN-3 data types and values are brought into GPF using the mechanism to parameterize MSC with arbitrary data languages. Data is incorporated into GPF in a number of places, such as document parameters, control variables, verdicts, component and port instances, message values, timers, action boxes, and references. Data is used in two distinguishable ways either statically, such as in the parameterization of an MSC diagram, or dynamically, such as in the acquisition of a value through a message receipt. All declarations, values and type definitions are specified using the TTCN-3 core notation that is placed, or referenced, within the data definitions part of the MSC document.

In GPF both, MSC documents and diagrams may be parameterized. An MSC reference must provide the actual parameters whose scope is the diagram body. Parameters are treated as constants, i.e. they cannot be modified dynamically.

Variables are owned by single instances representing test components. This means that only the instance owning a variable can define and change its value through the use of bindings. A binding can be considered to be a special form of an assignment. It consists of an expression part and a pattern part that are connected by a bind symbol. The bind symbol has left and right form both of which are equivalent, but which permit more natural reading of a binding associated with a message. Figure 7 illustrates a simple message exchange in which the variable `x`, owned by instance `TC1`, is bounded to an expression involving the variable `y`, owned by instance `TC2`.

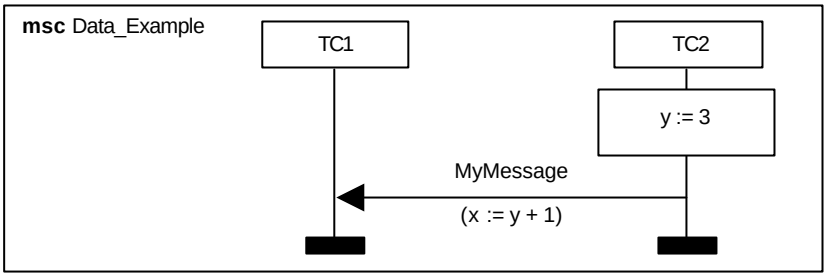



**Figure 7** Variable binding

## 3.3 Configuration

In TTCN-3, test configurations are related to component and port type definitions, and behavior descriptions, i.e., test cases, functions and named alternatives. Where, the component and port type definitions are provided by the user within the data definitions part of the MSC document. For test cases, functions and named alternatives corresponding MSC diagrams may be provided.

Instances within the MSC diagrams either represent test components or ports (Figure 8). In order to distinguish the two kinds of instances graphically, different graphical symbols can be used. Where port instances may be represented explicitly as particular 'environmental' instances by using dashed instance axes, or using the **port** keyword. Apart from the FIFO order defined by the TTCN-3 semantics for connections, no further event order is defined for port instances. For test components the standard event ordering for an MSC instance is assumed.

The GPF representation of the TTCN-3 configuration operations: **create**, **map**, **start component**, **stop component**, **start port**, **clear port** and **stop port** are shown in Figure 8.

The creation of a test component is represented by MSC create symbols, i.e. a dashed arrow pointing to the header of the newly created instance. Mapping (and connection) ports has to be done by using the TTCN-3 **map** (and **connect**) operations within action boxes. GPF represents the start of the execution of a test component by using a dashed start arrow. Special *start*, clear and stop messages are used to represent the TTCN-3 operations for controlling ports, i.e. **start port**, **clear port** and **stop port**. The termination of a test component (TTCN-3 **stop** operation) is described by means of the MSC stop symbol.

The MSC stop symbol in combination with a **return** keyword and an (optional) return value underneath is also used to represent the TTCN-3 **return** statement. A **return** statement describes the end of a function and the return of control and an optional return value to the calling entity, i.e. module control, test case or function.

## 3.4 Asynchronous communication

Within GPF, asynchronous communication is described by means of messages (Figure 9). Along MSC 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 message template has to be provided. Templates are the TTCN-3 mechanism to specify message values. A template may be defined in the data definitions part of a MSC document and is then referenced in the message, or is provided in form of an in-line definition. The message type is optional if the template is referenced, i.e. the message type is provided in the template definition.

## 3.5 Synchronous communication

TTCN-3 supports remote procedure calls as a synchronous communication mechanism. Where, a test component can either play the role of a calling party or the role of a called party.

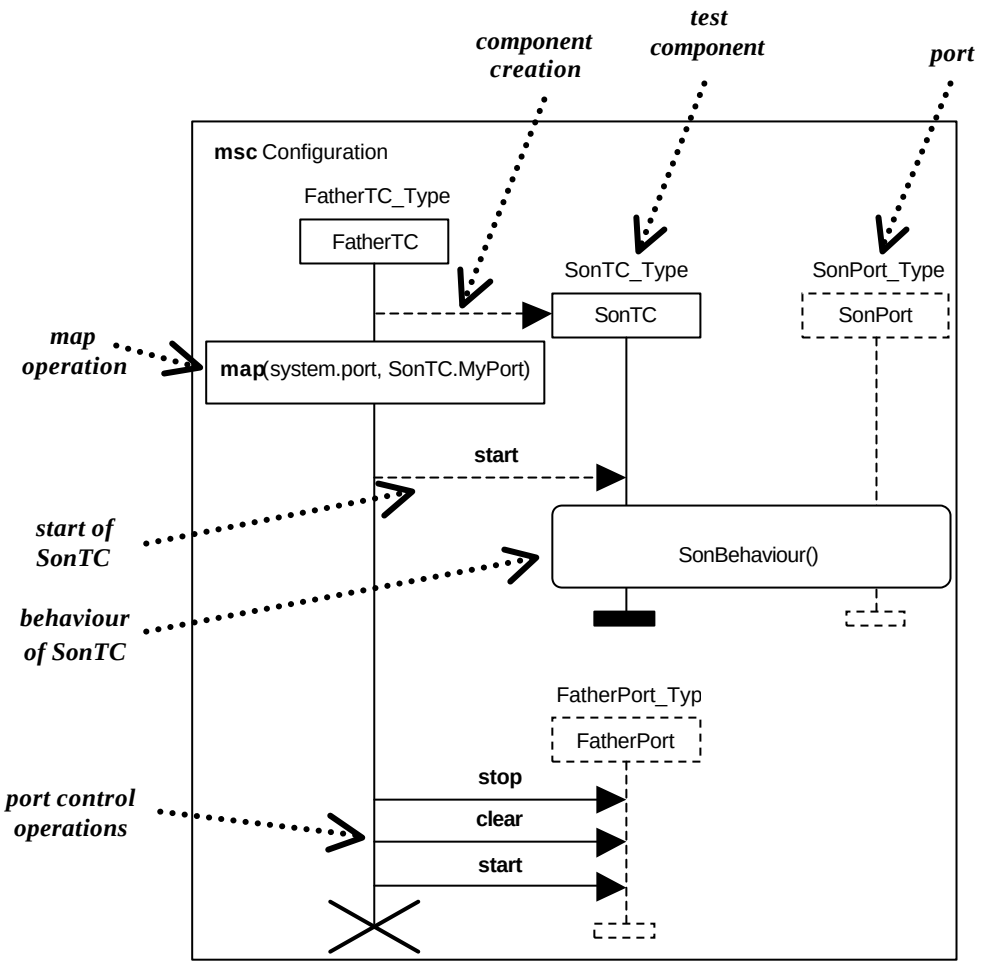



**Figure 8** GPF description of TTCN-3 configuration operations

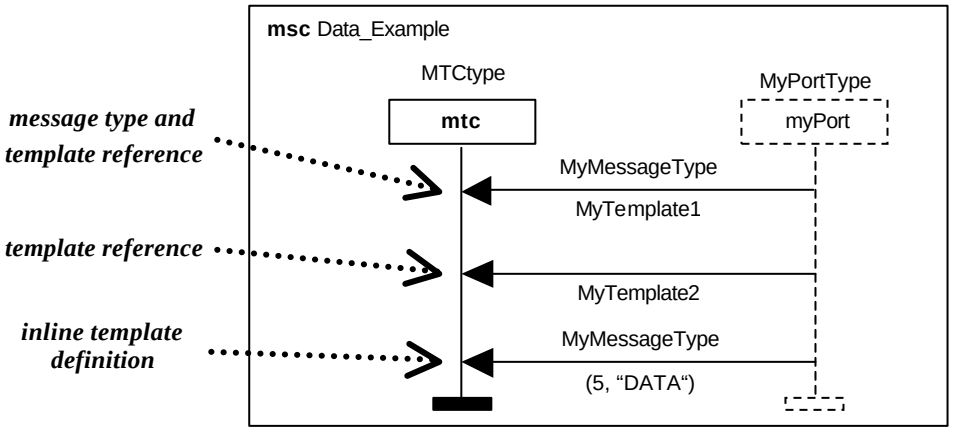
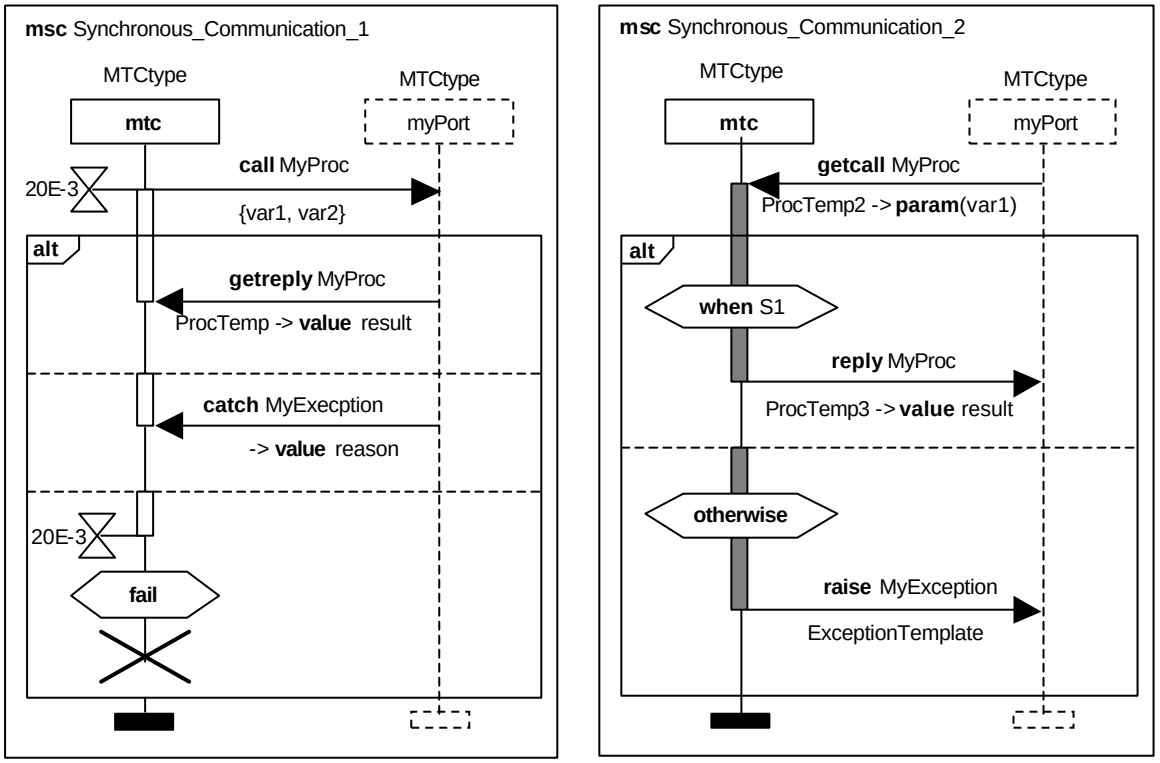



**Figure 9** Usage of messages and templates

For the calling party, TTCN-3 provides the **call** operation for calling a remote procedure, the **getreply** operation for handling the reply from remote, and the **catch** operation to catch exceptions raised by the remote side. The call operation may be guarded with a duration in order to handle situations where the remote side neither replies nor raises an exception. In GPF the operations are modelled by means of messages. 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.

In Figure 9a the main test component `mtc` plays the role of a calling party. A timer guards the call of the remote procedure `MyProc`. After the call, `mtc` either receives a reply, catches an exception or the guarding timer expires. The suspension region indicates the blocking of the test component during the call. TTCN-3 also supports non-blocking calls. For the specification of non-blocking calls, the blocking area is omitted.




(a) Blocking call with reply, exception and timeout

(b) Handling of an incomming call

**Figure 9** Synchronous coummunication

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 GPF 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 Figure 9b the test component `mtc` plays the role of a called party. 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 `mtc` replies to the call, or raises an exception.

## *3.6 Timer*

The semantics for timer is identical in TTCN-3 and MSC. Therefore, GPF provides a one to one mapping for the TTCN-3 timer operations *set*, *reset* and *timeout* to the corresponding MSC symbols (Figure 4). The TTCN-3 timer operations *running* and *read* have no graphical counterpart in MSC. They have to be specified in action boxes or in guarded conditions (running operation only) within a GPF presentation.

### 3.7 TTCN-3 statements without graphical representation

In GPF, MSC action boxes are used for TTCN-3 statements having no graphical representation, e.g. bindings, operation calls or function calls.

### 3.8 Alternative, cyclic and interleaved behavior

TTCN-3 provides several possibilities to describe alternative, cyclic and interleaved behavior. In GPF, all possibilities are represented using MSC in-line expressions.

The TTCN-3 constructs for describing alternative behavior are the if-else and the alt statement. In GPF, both are described by means of in-line expressions with an *alt* operator. For describing the branching conditions of a TTCN-3 **if-else** statement GPF uses guarding conditions. The special keyword **otherwise** has been introduced for conditions to emphasize the else branch (e.g. Figure 9b). The branching conditions of the TTCN-3 alt statement are represented in GPF by using again guarding conditions and the conditions described in the template of the message to be received.

Cyclic behavior refers to the TTCN-3 loop statements for, while and do-while. In GPF all loop statements are represented by means of inline expressions with a loop operator. The different exit criteria for the loops have to be described by appropriate guarding conditions.

TTCN-3 also provides a possibility to describe interleaved behavior by means of the **interleave** statement. In GPF, a new interleave operator for in-line expressions has been introduced to represent interleaved behavior in an appropriate way.

### 3.9 Test verdicts

In TTCN-3, the verdict of a test component is handled as a special object whose value can only be accessed by the operations *set* (for setting the verdict value) and *get* (for retrieving the actual verdict value). To emphasise the importance of verdicts, GPF uses conditions, containing the verdict value as special keywords for setting component verdicts. The verdict value may either be **none**, **pass**, **inconc**, or **fail**. An example for a verdict set operation can be found in Figure 9a. A **fail** verdict is assigned after the timeout. For retrieving the verdict value, the **get** operation can be used within an action box.

### 3.10 Default behavior

In TTCN-3, default behavior is used to handle unexpected or exceptional behavior of the SUT during the test run. The TTCN-3 default is a macro mechanism, which introduces additional alternatives to receiving events. Macros are defined in the form of named alternatives, and the macro expansion is driven by **activate** and **deactivate** statements.

In GPF, named alternatives may be represented in form of MSC diagrams and new graphical symbols for describing the activation and deactivation of defaults have been introduced. The activated and deactivated defaults, i.e. MSC diagrams, have to be referenced in activate and deactivate symbols. An example for the activation and deactivation of default `Default_1` can be found in Figure 10.

### 3.11 Function calls and execution of test cases

In TTCN-3, function calls and the execution of test cases are made by reference, i.e., the names of functions and test cases are used to refer to the corresponding definitions. Where, in TTCN-3 the **execute** keyword is used to denote the execution of a test case. In GPF, the same mechanism is used: MSC references are used to refer to MSC diagrams representing TTCN-3 test cases and functions. Where, the **execute** keyword is placed within the MSC reference symbol to denote the execution of a test case. Examples for a TTCN-3 **execute** statement and a function call are shown in line 39 and line 16 of Figure 1. The corresponding GPF descriptions can be found in Figure 5 and Figure 10.

## 4 A GPF example

This section presents a GPF example based on the Dynamic Host Configuration Protocol (DHCP) [12, 13]. DHCP is an Internet protocol for automating the configuration of computers that use TCP/IP. DHCP can be used to e.g. assign IP addresses automatically and to deliver TCP/IP stack configuration parameters such as the subnet mask and default router.

Figure 6 presents the MSC document `DHCP_Srv` that visualizes the TTCN-3 module shown in Figure 1. In the data definitions part definitions, test cases, functions and named alternatives are imported from document

DHCP_declarations and the test component type MTCtype is defined. An HMSC Control and the MSC AA_1 are referenced in the control and test cases part of the MSC document.

HMSC Control is shown in Figure 5. It defines the execution order of the test cases. In our case, the value of the document parameter Srv_configuration determines whether test case AA_1 or test case AA_2 is executed. AA_2 is not referenced in document DHCP_Srv (Figure 6). It is imported from DHCP_declarations.

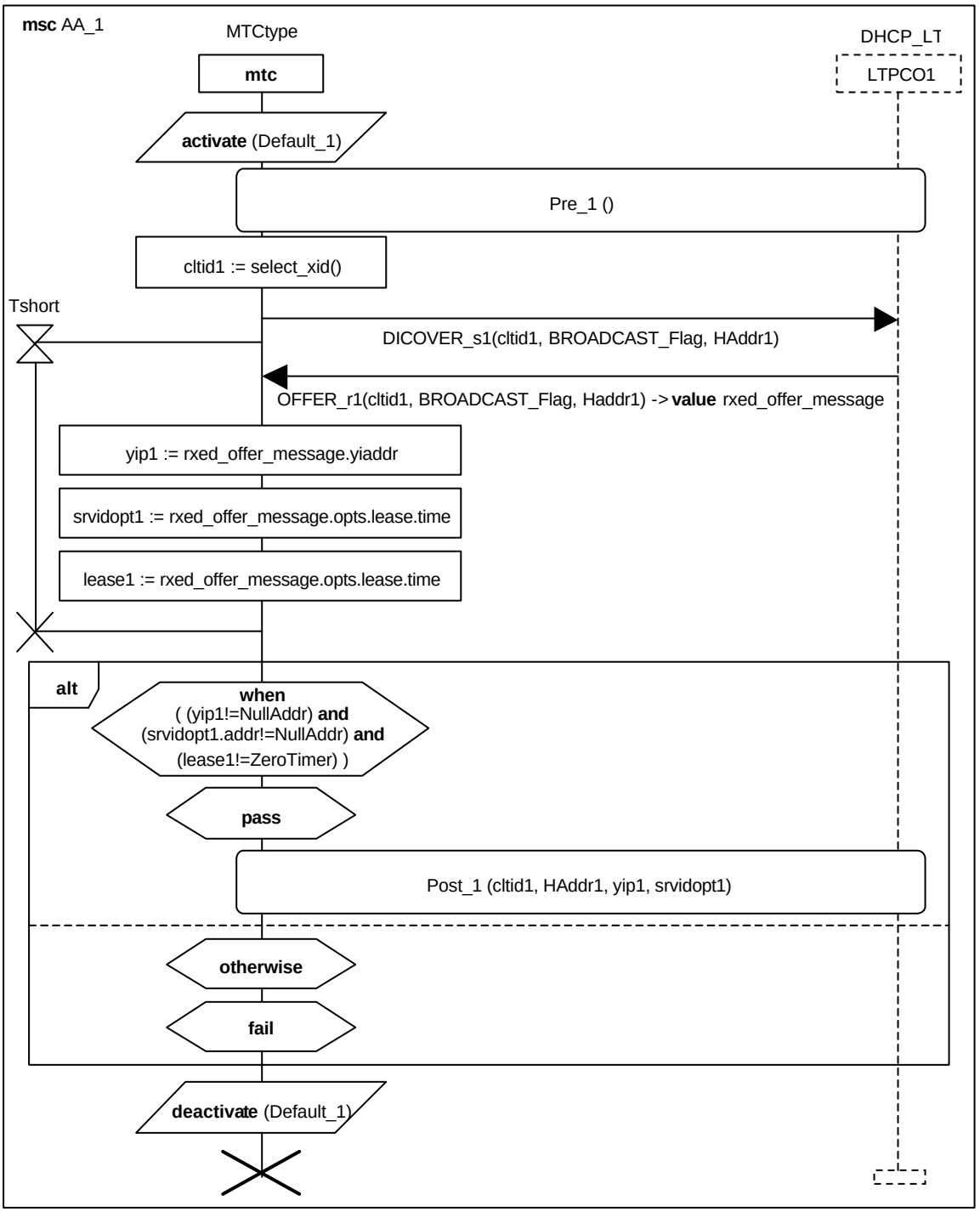



**Figure 10** A GPF description of test case AA_1

The MSC describing test case AA_1 is shown in Figure 10. It only contains the main test component MTC. MTC activates the default behavior by referencing the named alternative Default_1 and afterwards executes its preamble by calling the function Pre_1. Both, Default_1 and Pre_1 are not referenced in DHCP_Srv (Figure 6), i.e., they are imported from DHCP_declarations.

After performing Pre_1, the variable cltid1 is initialized, a DISCOVER message is sent over port LTPCO1 and the timer Tshort is started. Then, an OFFER message as response to the DICOVER message is awaited at port LTPCO1 and stored in variable rxed_offer_message. The following three action boxes describe the extraction of information from variable rxed_offer_message, i.e., from the received OFFER message.

If the OFFER message is not received within the run-time of timer Tshort, the timer will timeout. The timeout will be treated by the activated default behavior Default_1.

In Figure 10 the other case is shown. The OFFER is received in time and the timer is cancelled. Subsequent to that, an alternative is defined for the comparison of the received data values with the expected ones. If the received data values are as expected, a pass verdict is assigned and the postamble Post_1 is invoked. Post_1 is also imported from DHCP_declarations.

If the received data is not as expected, represented by the *otherwise* condition, the test verdict fail is assigned. Finally, the default Default_1 is deactivated and the test case terminates.

The close relationship of TTCN3 and GPF can be seen by comparing the TTCN-3 core language description of module DHCP_Srv (Figure 1) and the corresponding graphical presentation (Figure 5, 6 and 10). All TTCN-3 declarations, i.e., the import statement and the component type definition, can be found in the data definitions part of the MSC document. The TTCN-3 module control part and the test case definition AA_1 are graphically represented by MSC diagrams. Within the MSCs, all TTCN-3 constructs have straightforward and intuitive graphical representation.

# 5 Using GPF for the visualization of test traces

A test trace describes a concrete execution of a test case during a test campaign. Especially in cases where a test case fails, such a test trace is very helpful for analysing the failure. Test traces can be easily represented in GPF on different levels of abstractions: showing purely the messages exchanged between the test components and the SUT, or showing in addition the function in which a message has occurred and giving timer information, or showing also assignments or even showing further information on the test execution like default activation or start of timers etc. A sample test trace for the example introduced in the previous section is shown in Figure 11.

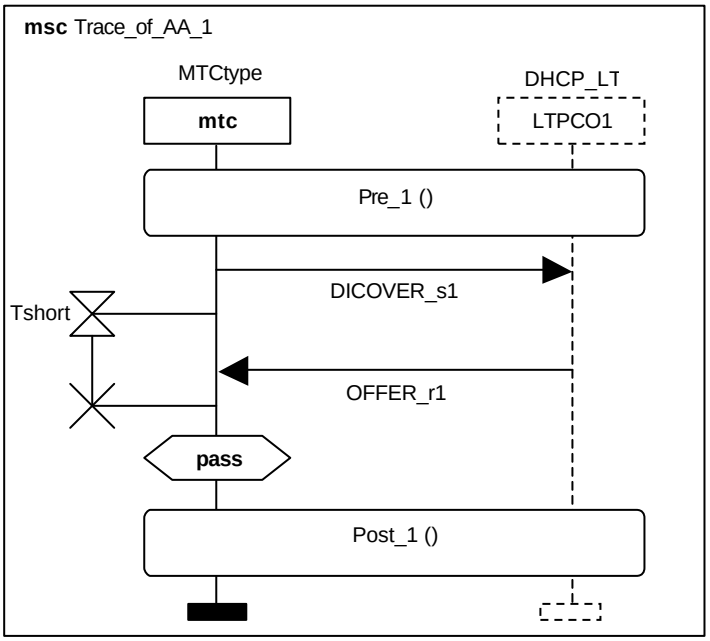



**Figure 11** A GPF trace of test case AA_1 shown in Figure 10

# 6 Outlook

The first complete GPF version has been edited and covers the concepts presented in this paper, i.e. single component definitions and static configurations. Included in this definition is a grammar definition and a mapping to the TTCN-3 core notation. Further work is planned to extend the existing GPF definition in order to handle multiple test components and dynamic configurations. Ongoing, as part of these efforts we are working on the convergence of GPF with MSC identifying how GPF concepts can be effectively represented as valid MSCs. Where this is not possible, extensions to MSC will be proposed.

More advanced studies cover the efficient and user-friendly handling of incomplete GPF specifications, complex behaviour and comprehensive documents [9, 16, 17]. Hybrid representations allowing the handling of TTCN-3 core notation and MSCs in the same document are promising for the handling of incomplete GPF specifications. Hypertext-like representations seem to be an adequate means for the intuitive presentation of complex and comprehensive MSC specifications.

Commercial tool support for GPF is already available. Existing MSC tools can be used for test specification purposes[4] (e.g. [19]), but the first new test tools supporting the whole GPF definition are also on the market (e.g. [20]).

Within the UML standards arena, MSC is being proposed as the sequencing notation for UML version 2.0. Where, the adoption of MSC would naturally lead to the use of a MSC test profile. However, UML still lacks test specification support. Therefore, we have driving a Request For Proposals (RFP) for UML test specification that would allow us to propose GPF as a future UML test specification profile.

# Bibliography


[1] ANSI/IEEE. *Glossary of Software Engineering Terminology*. ANSI/IEEE Std 729-1983, ANSI/IEEE Std. 729-1983, 1983.

[2] B. Beizer. *Software Testing Techniques* (Second Edition). Van Nostrand Reinhold New York, 1990.

[3] P. Baker, C. Jervis, D. King, An Industrial use of FP: *A Tool for Generating Test Scripts from System Specifications*. Trends in Functional Programming, Proceedings of the Scottish Functional Programming Workshop, Glasgow, UK, 1999.

[4] P. Baker, E. Rudolph, I. Schieferdecker, Graphical Test Specification – The Graphical Format of TTCN-3. Proceedings of the 10th SDL-Forum, Copenhagen, June 2001, Lecture Notes in Computer Science, Springer, June 2001.

[5] ETSI ES 201 873-1. *Methods for Testing and Specification; The Tree and Tabular Combined Notation version 3 (TTCN-3); Part 1: TTCN-3 Core Language*. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2001.

[6] ETSI ES 201 873-2. *Methods for Testing and Specification; The Tree and Tabular Combined Notation version 3 (TTCN-3); Part 2: TTCN-3 Tabular Presentation Format (TPF)*. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2001.

[7] ETSI TR 101 873-3. *Methods for Testing and Specification; The Tree and Tabular Combined Notation version 3 (TTCN-3); Part 3: TTCN-3 Graphical Presentation Format (GPF)*. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, 2001.

[8] J. Grabowski. *TTCN-3 - A new Test Specification Language for Black-Box Testing of Distributed Systems*. Proceedings of the 17th International Conference and Exposition on Testing Computer Software (TCS'2000), Theme: Testing Technology vs.Testers' Requirements, Washington D.C., June 2000.

[9] J. Grabowski, P. Graubmann, E. Rudolph. *HyperMSCs with Connectors for Advanced Visual System Modelling and Testing*. Proceedings of the 10th SDL-Forum, Copenhagen, June 2001, Lecture Notes in Computer Science, Springer, June 2001.

[10] J. Grabowski, D. Hogrefe. *TTCN SDL- and MSC-based specification and automated test case generation for INAP*. Proceedings of the "8th International Conference on Telecommunication Systems (ICTS'2000) - Modeling and Analysis", Nashville, March 2000.


[4] 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.

[11] J. Grabowski, A. Wiles, C. Willcock, D.Hogrefe. *On the Design of the new Testing Language TTCN-3*. '13th IFIP International Workshop on Testing Communicating Systems' (Testcom 2000), Ottawa, 29.8.2000-1.9.2000, Kluwer Academic Publishers, August 2000.

[12] IETF rfc 2131. *Dynamic Host Configuration Protocol*. March 1997.

[13] IETF rfc 2132. *DHCP Options and BOOTP Vendor Extensions*. March 1997.

[14] G. J. Myers. *The Art of Software Testing*. John Wiley, 1979.

[15] E. Rudolph, J. Grabowski, P. Graubmann. *Towards a Harmonization of UML-Sequence Diagrams and MSC*. In: 'SDL'99 - The next Millenium' (Editors: R. Dssouli, G. v. Bochmann, Y. Lahav), Elsevier, June 1999.

[16] E. Rudolph, I. Schieferdecker, J. Grabowski. *HyperMSC - a Graphical Representation of TTCN*. Proceedings of the 2nd Workshop of the SDL Forum Society 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] I. Sommerville. *Software Engineering*. Addison Wesley, 1989.

[19] Telelogic AB. TAU product description. http://www.telelogic.se/products

[20] Testing Technologies. *TT Tools product description*. http://www.testingtech.de/products/TTToolSeries.html

[21] J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modelling Language, Reference Manual Version 1.1*. Rational 1997.

[22] ITU-T SG 10. *Message Sequence Chart (MSC)*. Rec. Z.120, Geneva 1999.