TTCN-3 Test Case Generation from Message Sequence Charts

Michael Ebner

Telematics Group, Institute for Informatics,
Georg-August-Universität Göttingen, Germany,
ebner@cs.uni-goettingen.de

Abstract Scenario-based testing, manual as automatic, is applicable for black-box and specific white-box testing for communication protocols and distributed systems. UML models provide scenario descriptions by sequence diagrams respectively MSCs. Thus, the combination of TTCN-3, as test description language, and UML by MSC to specify and automatically generate test cases has to be considered.
The work in this paper describes a translation of MSC elements to TTCN-3 statements in order to use MSC to define test cases and test case execution order in TTCN-3.

Keywords: Testing and Test Control Notation, TTCN-3, MSC, UML, sequence diagram, test case generation

1 Introduction

The complexity of modern telecommunication systems has increased significantly and the necessity for thorough and systematic testing is undisputed. For instance, conformance and functional testing is widely used in the telecommunication area. However, testing is an expensive and time-consuming task. Before concrete tests can be carried out on a system, much effort has to be spent on specifying what and how to test and on obtaining the test descriptions in a format that is accepted by the test equipment.

Testing has to be integrated into the development process. Therefore, to reduce testing effort tests should be generated from system specification which is named *Computer Aided Test Generation* (CATG). Manual test generation is error-prone wherefore test generation must be automated to be effective and repeatable. However, test generation based on state space exploration is helpful but generates frequently inefficient tests, lacks to cover specific parts, or may not be possible because of an incomplete or missing specification. Thus, *scenario-based, manual test specification* is interesting where the test designer can focus on specific elements in the *System Under Test* (SUT) and requires no test specific knowledge like the used test language.

In the telecommunication area, the *Tree and Tabular Combined Notation* (TTCN) is used as a standardised test description language. For example, *Tree and Tabular Combined Notation (version 2)* (TTCN-2) [1] has been applied successfully to functional testing of communication protocols for years.

Unified Modeling Language (UML) models, defined by the *Object Management Group* (OMG) [2], are an important source for test development and using UML from a test perspective has to be considered. Additionally, there is ongoing work on a UML *Testing Profile* (UTP) which can be mapped to *Testing and Test Control Notation (version 3)* (TTCN-3).

Scenario-based testing, manual as automatic, is applicable for black-box and specific white-box testing for communication protocols and distributed systems like CORBA-based systems, for instance. In particular, manual scenario-based testing is used by test designers to test specific parts of the SUT which are interesting or not covered by a automatic test generation process.

UML provides sequence diagrams which are very similar to *Message Sequence Charts* (MSCs). Thus, they are used to specify test scenarios. Furthermore, main concepts of MSC have been introduced in *Unified Modeling Language 2.0* (UML 2.0) which leads to the convergence of sequence diagrams to MSCs [3,4].

Using TTCN as test description language is quite natural because of its successful application in the telecommunication area by using automatic test generation, and the possible usage of TTCN-3 with UML via the UML *Testing Profile* (UTP). In addition, it was successfully applied for testing *Common Object Request Broker Architecture* (CORBA)-based systems.

The remainder of this paper is structured as follows. Firstly, some basic concepts of TTCN-3 and MSC are introduced. Secondly, the translation of MSC to TTCN-3 to generate test cases gets explained. Finally, some concluding remarks are given.

2 Test and Test Control Notation-3

The *Testing and Test Control Notation (version 3)* (TTCN-3) is the third part of the *Conformance Testing Methodology and Framework* (CTMF) standard for the specification of test suites for conformance testing [5].

TTCN is designed for functional, black-box testing and to describe *Abstract Test Suites* (ATS) which are independent of a concrete test platform. Therefore, special interfaces defined by TTCN between a *System Under Test* (SUT) and the ATS are required to make a test suite executable. First, there has to be an *Abstract Test System Interface* (ATSI) which defines the sight of the ATS upon the SUT. The access points between ATS and SUT are called *Point of Control and Observation* (PCO). Secondly, there is a *Real Test System Interface* required which maps the ATSI to the SUT.

Test configuration in TTCN is done by the *Main Test Component* (MTC) which controls all other test components called *Parallel Test Components* (PTCS). PTCs can be dynamically created whereas the MTC is created automatically at each test case execution. Test components in TTCN-3 communicate with each other via ports (in TTCN-2 via *Communication Points* (CP)), which are modelled as infinite FIFO queues to store incoming calls. Communication between test components and the test system is also done via ports (in TTCN-2 via PCOs).

TTCN-3 improves concepts of TTCN-2 and introduces new concepts to be a test description language for reactive system tests over a variety of communication platforms such as CORBA-based platforms. An important feature of TTCN-3 is the enhanced communication concept which now supports procedure-based communication to provide synchronous communication, as well as the message-based communication which is asynchronous. In addition, a test execution control part, a module and grouping concept and new data types, are introduced to provide a better control and grouping mechanism.

3 Message Sequence Chart

Message Sequence Chart (MSC) is a graphical specification language standardised by ITU – *Telecommunications Standardisation Sector* (ITU-T) as Recommendation Z.120 [6]. It is a scenario language to describe communication behaviour between system entities and their environment. Sequence charts like UML sequence diagrams [2] and *Message Sequence Charts* (MSCs) are used to specify and describe communication behaviour by message interchange including procedure calls between several distributed entities in a temporal order. MSCs are well used for test purpose and test case specification. Furthermore, there is ongoing work which introduces MSC concepts into UML 2.0 which leads to the convergence of sequence diagrams to MSCs [3,4].

Three types of charts are provided by MSC. Namely, (basic) MSC to describe concrete events in a temporal ordering, *High-Level Message Sequence Chart* (HMSC) to illustrate how to combine MSCs, and MSC documents which serve as a summary of belonging charts.

MSC documents are used to provide an associated collection of MSCs (set of traces) and define all kinds of instances used in the MSCs.

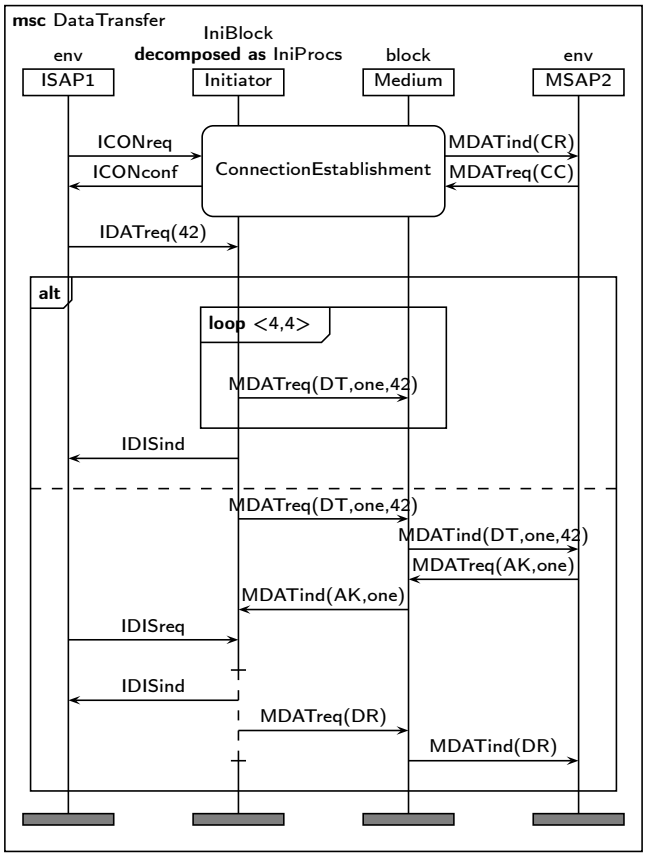
Basic MSCs are used to specify and describe the communication flow between system entities where the concept of *instances* and *messages* is used (see Figure 1). Communication with the *environment* can be described and usage of *gates* to compose MSCs is supported. *Actions* are used to describe internal behaviour of entities and *conditions* are used to restrict number of traces. *Timers* are available to express time limits for execution. Instances can be dynamically created and terminated.

Temporal ordering of exchanged messages is defined by a total temporal ordering on instance axis and a partial order between instances where only the order is defined but no concrete timing. Apart from message interchange, MSC supports synchronous message exchange by means of calls and replies. Calls are like remote method invocations where the result is returned by the reply.

Beside basic MSCs the structural concepts *coregions*, MSC *references*, *instance decomposition*, and *inline expressions* are supported.

Total ordering on instance axes can be suspended by *coregions* where ordering gets changed to allow any event order.

Inline operators are used for easier definition of event structures. The operators **alt**, **par**, **seq**, **opt**, **exc**, and **loop** are available to define alternative,



(b) MSC Expressions, Coregions, and Gates

Figure 1. Basic MSCs for the Inres protocol

parallel, and sequential composition, optional regions, exceptions, and iterations. Alternative composition is used to define alternative execution traces of an MSC whereby only one trace gets executed. The choice between different traces has to be done after executing the common part of the possible traces.

Apart from MSC documents and basic MSCs there are *High-Level Message Sequence Charts* (HMSCs) available as another structuring type. HMSCs are directed graphs which describe how to combine a set of MSCs. Thus, HMSCs provide a higher description and structuring level by abstracting from concrete message exchanges.

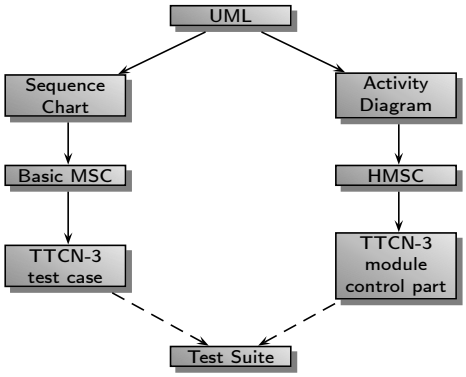


Figure 2. Basic MSC to TTCN-3 translation concept

4 MSC to TTCN-3 Translation

CATG based on state space exploration generates frequently inefficient test cases. Therefore, it is desirable to use, for instance, graphical test purposes, for CATG which are also more suitable for a test designer. Test case generation and specification using MSCs were formerly done for TTCN-2 by using *Message Sequence Chart-1996* (MSC-96) and is provided, for instance, with AUTOLINK in the TAU tool set from TELELOGIC [7,8,9]. Hence, a translation from MSC to TTCN-3 gets defined to allow definition of graphical test purposes for TTCN-3, too.

The focus is on timed order of message exchanges and test suite details are hidden. This distinguishes this approach in contrary to UTP where a test suite is represented in more detail. Thus, specification of scenario-based test cases gets simplified. Furthermore, usage of a given specification by MSCs or UML diagrams which can be converted into MSCs is possible. The *Message Sequence Chart-2000* (MSC-2000) is used as substitute for UML sequence charts as stated before. UML activity diagrams can be converted to HMSC which is used to define test execution order (see Figure 2). Thus, MSC is used to generate TTCN-3 test cases and control parts.

MSC is also used for real-time testing and MSC concepts have been used to develop TTCN-3 *Graphical Presentation Format* (GFT) and UTP. The introduction of new concepts for real-time testing with TTCN-3 and MSC is discussed in [10,11]. The GFT in context of MSC and UML is discussed in [12]. The deployment of UTP is detailed in [13] where a UML-based specification of test descriptions is explained.

The translation starts with a description of the used approach to motivate the following parts which are structured similar to the MSC specification document [6] to provide easy access to the translation of each MSC element. Therefore, translation of the MSC documents and comments, basic MSCs, structural concepts, and HMSCs are explained.

4.1 Approach

Contrary to the TTCN-3 presentation format GFT [14] and UML test specification via UTP [13] which are inspired by MSC the approach here uses MSC to manually specify test purposes and cases. However, the approach is also not using the full MSC semantics because some restrictions and adaptations respectively have to be done to permit test case specification via MSC. Thus, MSC semantics is overwritten by an own MSC to TTCN semantics which is as near as possible to the MSC semantics without introducing new graphical elements. Hence, seamless use by test designer familiar with MSC is supported.

Requirements There are some requirements which are desirable to provide good support for TTCN-3 with MSC and to widen usability and acceptance:

Structure It should be feasible to control the TTCN-3 test case generation process to get a desired test suite structure. For instance, the MSC document structure can be used.

Aliases Use of aliases is important to write easily test cases via MSCs wherefore usage of TTCN-3 templates has to be supported.

Statements Support insertion of TTCN-3 statements (program code) in MSC, for instance, insertion of **setverdict(pass)** at the end of a MSC diagram.

Defaults Usage of predefined TTCN-3 defaults must be possible.

Concurrency Support concurrent and non-concurrent TTCN-3 wherefore non-concurrent TTCN-3 should be a special case of concurrent TTCN-3.

Basic Structure In order to use MSC for generating TTCN-3 the test architecture and communication has to be mapped first. Static information like types and data has to be given by other sources like by a *Interface Definition Language* (IDL) mapping [15]. Test architecture and communication is based on components and ports whereby ports are the element which connects both together. Furthermore, usage of components would be too abstract for scenario-based testing. Therefore, MSC instances are mapped to ports (see Figure 3). At least the static *Main Test Component* (MTC) and SUT ports have to be used to specify test cases. Dynamic *Parallel Test Components* (PTCs) can be represented by instance creation. In order to distinguish ports by components the instance head provides the component name in addition. Hereby, an own test case for each component can be generated in case of concurrent TTCN. In case of non-concurrent TTCN usage of PTCs is forbidden and component name MTC is optional and only test case(s) for the MTC are generated.

In case where a MSC is ambiguous in sense of a TTCN-3 test case several test cases will be generated to comply the ambiguities. Nevertheless, an ambiguous test case specification in sense of this approach cannot be seen as a test purpose in its proper meaning because traces have not to be completed. Ambiguous test cases can appear by using sending messages in front of alternative inline expressions and must appear by using HMSCs.

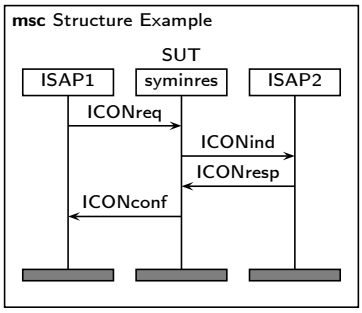


Figure 3. Basic instance structure to specify test cases via MSC

Restrictions and Adaptations As stated before, some restrictions and adaptations are made to use MSC in conjunction with TTCN-3:

Configuration file A configuration file has to provide the used *data types*, *templates* (constraints), and configuration information like *port* definitions and their types and components (for create, connect, visibility) which can be provided by a *Specification and Description Language* (SDL) or IDL specification, for instance.
The configuration file has to be a correct TTCN-3 module. A reference to the configuration file can be given via a text comment in the MSC because the generated test case will be inside a TTCN-3 module, too.

Synchronisation At the border of an inline expressions synchronisation is always assumed to prevent cases of *interleave* and to permit loop abortion in loop expressions. MSC references are synchronised per component.
Synchronise conditions and local synchronise rules will be used and hence, no global synchronisation and global references are available.

Message type In an inline expression the first message has to be type limited to make sense in TTCN-3. Therefore, only *receive messages* and *receive statements* respectively are permitted as first message in an alternative, optional, and exceptional inline expression.

4.2 MSC Documents and Comments

MSC documents are used to structure test cases in a TTCN-3 module wherefore the MSC document name is used as module name and the referenced MSCs are converted to appropriate test cases inside the module. The document *relation* is used to bind a configuration file to the MSCs where static informations are provided.

The three comment types *note*, *comment*, and *text* are used to insert TTCN-3 statements and comments. Notes occur only in the textual syntax and therefore, may be used in comment types *comment* or *text* to insert TTCN-3 comments. Comments are associated with a symbol or comment type *text* why they can be used to insert TTCN-3 statements at special positions. Text is used for global

TTCN-3 comments and statements which have to be set at the beginning of the generated TTCN test case. For instance, defaults can be activated in *text* comments.

4.3 Basic Message Sequence Charts

The conversion concepts for basic MSC are given below. Instances, messages, and control flow is discussed first and the ordering and synchronisation of events gets explained afterwards. At the end the conversion of environment and gates, actions, timers, and dynamic instances is detailed (see Figure 4).

Each chart contains a chart name which gets the test case name which may include used parameters.

Instances, Messages, and Control Flow Instances and communication by message exchange and procedure invocation are the basic parts of MSCs. According to the approach mentioned before instances are used as ports and *Point of Control and Observations* (PCOs) respectively where the corrsponding component is mentioned, too. Instance *name* is used as port name and instance *kind* without denominator is used as corresponding component name.

The instance definition itself provides no further information but attached elements are considered to belong to the port or component respectively. Hence, attached messages and procedure invocations belong to the given port and all others to the component why all elements are put into the same component test case(s). The used port type like **message**, **procedure**, or **mixed** may be given in the instance name but is only necessary if semantics checks will be done without further available configuration information or because of consistency for test designer. The concrete port definitions are given by the separate static information in the configuration file as also done for the component definition.

If no architecture information beside ports and SUT is given a heuristic can be used or only a **mtc** is assumed. It is indicated which ports belong to the SUT and message interchange where no SUT instance is involved indicates use of a PTC as far as the MTC sends no message to itself. For instance, in case of concurrent TTCN synchronisation between components is done via synchronisation coordination messages. Furthermore, test cases are focussed on communication with the SUT why SUT instances are mostly involved in message interchange. However, usage of such heuristics is up to tool vendors because they have no influence to the translation itself.

A MSC message and procedure consists of a relation between an output and input event from and to an environment or instance. These can be matched to send and receive operations in TTCN-3. Output message events can only be matched to **send** and output procedure events to **call**, **reply**, or **raise** operations whereas input message events can be matched to **receive**, **trigger**, and **check** and input procedure events to **getcall**, **getreply**, and **catch** operations. It is possible to provide a TTCN-3 statement attached to an event by a *comment* to force its conversion to a specific communication operation. The default

conversion converts to **send** and **receive** respectively for messages and **call** and **reply** for procedures if a statement is missing. Value assignments and optional parameters can also be provided in comments to a communication event. One-to-many connections are not supported until now by MSC and therefore, are not considered here.

Procedure calls in TTCN-3 have a blocking characteristic to state whether test case execution is blocked or not until the call has returned by a response or exception. Thus, MSC asynchronous and synchronizing calls are mapped to TTCN-3 non-blocking and blocking calls.

The message and procedure names represent either the used type (message type or procedure) or template (message or signature template). In case of a template without parameters no parameter list is given. Otherwise, the parameter list is used to provide the necessary parameters. TTCN-3 templates and their arguments provide easier usage of communciation operations because there are no further information necessary. Inline templates for matching receiving events can be used, too. Thus, test cases get more understandable and maintainable.

Ordering and Synchronisation To specify and implement test case generation from MSC the specification of the underlying event ordering is fundamental. Using MSC event ordering implies generation of test cases for each possible trace through a chart. However, intention of using sequence diagrams by test designer directly is control of generated test cases and thus, a limited MSC event ordering gets used. In the following only non-concurrent TTCN is mentioned to keep explanation simpler. Concurrent TTCN differs mainly in generation of test cases for each component which can be easily adopted by considering involved instances only [16].

Event ordering in MSC is defined by

- total ordering on instance axis,
- partial ordering between instances,
- a general ordering mechanism,
- and coregions.

Total ordering on instance axis is also used because event order on a port has to be ordered totally by default. If another behaviour is wished the ordering has to be modified explicitly by using corresponding elements. Partial ordering between instances can enable several permitted traces of a chart which prevents exact test case design because of unwished traces. Thus, partial ordering is limited by *synchronisation points*, send events are executed as early as possible, and graphical order is used to decide about used trace if necessary. Graphical order means ordering is given by the order from left to right and top to bottom in the MSC until the end or a synchronisation point.

Synchronisation points are used to provide a well defined point where all send and receive events have been executed wherefore event execution before is forced and no event after can be moved before a synchronisation point. Due to the limitation of the partial ordering between instances and the preference of the

graphical order the general ordering mechanism is not necessary but can be used for additional, more precise, and explicit order information. Coregions are used to override explicitly any ordering by all permutations of the involved events which is done by the **interleave** statement in TTCN-3. If the above mentioned rules for ordering can be widen by automatic detection of *interleave* cases like reception of two consecutive events at the SUT an **interleave** statement shall be used for it.

Synchronisation can be forced by using conditions and is assumed around all inline expressions and actions. MSC references are synchronised per component. Since synchronise conditions and local synchronise rules like described before are used, no global synchronisation and global references are available.

Miscellaneous Translation of environment, actions, conditions, timers, and dynamic instances gets described.

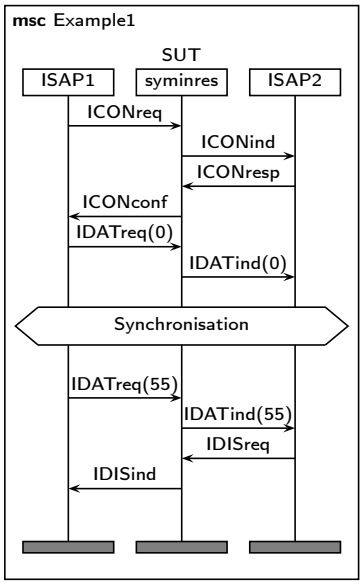
Environment and Gates Support of external message exchange can be designed similar to the normal message exchange by handling the environment like another instance. Gates require no special translation rules because they are only used for better organisation of charts.

Actions An action describes an internal activity of an instance and hence, it can be used to insert comments and TTCN-3 statements directly into test cases depending on an instance.

Conditions Conditions are only used for synchronisation which requires coordination messages if concurrent TTCN is used.

Timers and Time Constraints Timers can be started and stopped according their position in the MSC and only plotted and global timers are considered. Timer time out can be catched by using the default behaviour statement of TTCN where the test case verdict can be set to **fail** and a log message can be written. There is no stop after the verdict to allow ending in a defined state.

Instance Creation and Termination Dynamic instances are mapped like static instances but are created and started only during test case execution and not at the begin of a test case like it is done for MTC and SUT. However, dynamic instances make only sense if a new component is used. Information for connecting and translation of ports to each other can be taken from exchanged messaged. Instance termination is mapped to the **stop** component operation but can only be inserted if no other port of the component is in use.



(a) *MSC*

```
testcase Example1() runs on syminres {
    ISAP1.send( ICONreq )
    ISAP2.receive( ICONind )
    ISAP2.send( ICONresp )
    ISAP1.receive( ICONconf )
    ISAP1.send( IDATreq ( 0 ) )
    ISAP2.receive( IDATind ( 0 ) )
    ISAP1.send( IDATreq ( 55 ) )
    ISAP2.receive( IDATind ( 55 ) )
    ISAP2.send( IDISreq )
    ISAP1.receive( IDISind )
}
```

(b) *TTCN-3*

Figure 4. MSC to TTCN-3 Base Translation

4.4 Structural Concepts

Conversion of coregions, references, instance decomposition, and inline expressions, which are all summarised as structural concepts in MSC, are detailed now.

Coregions Coregions are used to override explicitly the total ordering of an instance axis by unordered events. In sense of a test case description best translation is done by using all permutations of the involved events which is done by the **interleave** statement in TTCN-3, as well. This was mentioned earlier in 4.3.

Usage of coregions is limited on message exchange among components and between SUT and a component. Overlapping of coregions on different instance axes for the same component has to be prevented. The first event must be always a receive event. In case of non-concurrent TTCN coregions are only used on SUT axes.

MSC References MSC references provide usage to import other charts which easies decomposition and reuse. However, test cases cannot be decomposed into other test cases and therefore, references are converted to function calls. Nevertheless, test case decomposition is done by HMSCs which is described in subsection 4.5. To allow correct behaviour of function calls by references they have

to be synchronised per test component. Several function calls can be provided in one reference by usage of the **seq** operator for references.

The *sequential* MSC reference expression **seq** is used to call several functions in sequential order. The *alternative* and *optional* expression **alt** and **opt** is used to generate a test case for each possible alternative. *Loop* expressions are converted according done for MSC inline expressions in HMSCs (see subsection 4.5).

Instance Decomposition Instance decomposition is used to replace an instance axis by a detailed chart. The instance axis can be seen as the environment from the detailed chart. Therefore, instance decomposition has only to be resolved during conversion wherefore no further special conversion concept is required.

Inline Expressions In order to structure charts inline expressions are provided which allow alternatives, loops, and parallel execution.

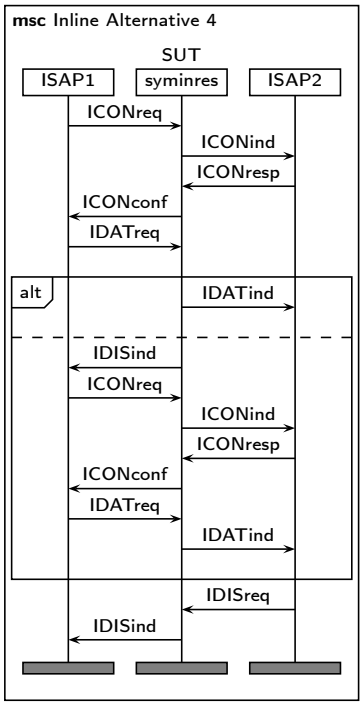
An appropriate translation requires synchronisation before and after each inline expression to prevent cases of *interleave* which cannot be solved easily. In addition, handling concurrent TTCN-3 gets better. From MSC point of view events before and after an inline expression may influence the expression behaviour but for test case design this behaviour is not wished because it makes test case design more complicated by loosing control about the number of generated test cases. To provide a good translation to TTCN-3 alternative and loop statements synchronisation is required, as well. Especially, good support of *loop* expressions by using TTCN loop statements is also wished.

The inline expressions *alternative*, *option*, *exception*, and *loop* are detailed below. Inline expression *parallel* will not be mentioned because there is no appropriate translation available.

Alternative The *alternative* expression is directly mapped to the alternative behaviour statement of TTCN-3. Therefore, the first element of each alternative part has to be a reception statement according the specification of the **alt** statement. For each alternative of an alternative expression containing a send event as first event another test case containing this alternative is generated. In worst case an own test case gets generated for each alternative. Of course, all events in alternatives are restricted by the **alt** statement semantic (see Figure 5).

Optional The *optional* expression can be seen as a special case of the alternative expression with an additional *empty* alternative part. The empty alternative part gets realised by an **else** guarded empty alternative part.

Exception The *exception* expression is a special case of the *alternative* expression where the last alternative part is the remainder of the MSC.



(a) *MSC*

```
testcase  Inline_Alternative4 () runs on syminres {
    ISAP1.send( ICONreq )
    ISAP2.receive ( ICONind )
    ISAP2.send( ICONresp )
    ISAP1.receive ( ICONconf )
    ISAP2.send( IDISreq )
    ISAP1.send( IDATreq )
    alt {
        [] ISAP2.receive ( IDATind )
           {
           }

        [] ISAP1.receive ( IDISind )
           {
                ISAP1.send( ICONreq )
                ISAP2.receive ( ICONind )
                ISAP2.send( ICONresp )
                ISAP1.receive ( ICONconf )
                ISAP1.send( IDATreq )
                ISAP2.receive ( IDATind )
           }

    }

    ISAP2.send( IDISreq )
    ISAP1.receive ( IDISind )
}
```

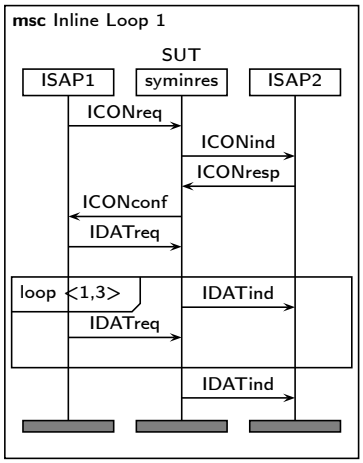
(b) *TTCN-3*

Figure 5. MSC to TTCN-3 Alternative Translation

Loop The *loop* expression can be converted onto the **for** and **while** loop statements of TTCN-3 whereby infinite loops are best mapped to **while** loops and finite loops to **for** loops. In contrary to the alternative expression there is no special first event necessary. Instead, loop operations require an abortion criteria if lower and upper boundary are *not* equal. Equal boundaries impose exact number of loop passes where no abortion criteria is required. The abortion criteria from MSC point of view is the next receive event after the loop expression. Therefore, a receive message has to be given after each loop expression. The receive message gets checked inside the loop and concrete message consumption is done after loop execution (see Figure 6).

4.5 High-Level Message Sequence Charts

HMSC are thought to describe possible combinations of MSCs. As mentioned earlier UML activity diagrams are comparable with HMSCs. If they are seen as a kind of test suite decomposition they can be used to specify test case execution order. Therefore, HMSCs are used to describe test case execution in the module control part of TTCN-3.



(a) *MSC*

```
testcase Inline_Loop1() runs on syminres {
    ISAP1.send( ICONreq )
    ISAP2.receive ( ICONind )
    ISAP2.send( ICONresp )
    ISAP1.receive ( ICONconf )
    ISAP1.send( IDATreq )
    var integer iterator__1 := 0;
    for ( iterator__1 := 0; iterator__1 < 3;
          iterator__1 := iterator__1 + 1 ) {
        alt {
            [] ISAP2.receive ( IDATind )
               {
                   ISAP1.send( IDATreq )
               }

            [iterator__1 >= 1] ISAP2.receive( IDATind )
                               {
                                   goto iterator__1_label
                               }
        }
    }
    label iterator__1_label;
}
```

(b) *TTCN-3*

Figure 6. MSC to TTCN-3 Loop Translation 1

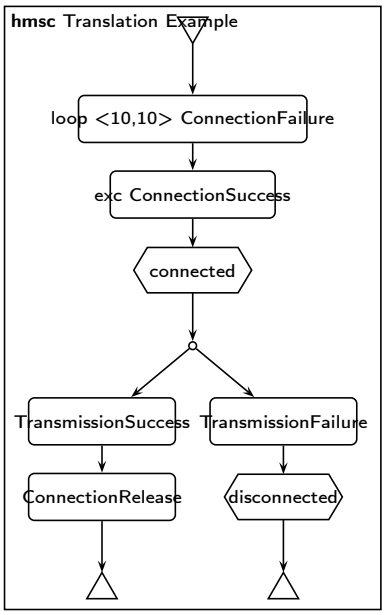
MSC references including reference expression in HMSCs are converted to corresponding test case execution calls and conditions and parallel frames are ignored. Each execution trace of a HMSC gets collected in an own function and all functions are called in the control part. No backward loops are allowed and there are only reference expression loops with same finite lower and upper boundary allowed because data concept is not introduced into UML and thus, not considered here. Alternative and optional expressions lead to several traces only. Exception expressions are used to stop further trace execution if the test case fails. An example is given in Figure 7.

5 Conclusions

The work presented in this paper describes a translation of MSC elements to TTCN-3 statements in order to use UML sequence charts to define test cases and test case execution order in TTCN-3.

Usage of comments for introducing direct TTCN-3 statements is used especially for TTCN-3 *defaults*. One-to-many connections are not supported until now by MSC and therefore, are not considered. Apart from translation chart elements the ordering semantics of events had to be defined which differs from the original semantics of MSC to be adequate for test specification. For instance, default synchronisation points were introduced for references and inline expressions.

Procedural communication is represented by flow control concept and mapped to **call** and **getreply** statements. However, alternative handling of reception events by usage of alternative statement in a blocked call or a **getreply** statement is not supported until now. Data and time concepts of MSC-2000 have not



(a) *HMSC*

```
module {

  function testExecution1 () {
    verdicttype verdict ;
    integer i ;

    for ( i:=10; i>0; i:=i-1)
      execute ConnectionFailure ();

    verdict = execute ConnectionSuccess();
    if ( verdict == false )
      return ;

    execute TransmissionSuccess ();
    execute ConnectionRelease();
  }

  function testExecution2 () {
    verdicttype verdict ;
    integer i ;

    for ( i:=10; i>0; i:=i-1)
      execute ConnectionFailure ();

    verdict = execute ConnectionSuccess();
    if ( verdict == false )
      return ;

    execute TransmissionFailure ();
  }

  control {
    testExecution1 ();
    testExecution2 ();
  }
}
```

(b) *TTCN-3*

Figure 7. MSC to TTCN-3 Translation of HMSCs

been considered because they are not used in UML 2.0, until now. Nevertheless, in context of *TIMED*TTCN-3 it has been shown that is possible to translate real-time information contained in MSC to *TIMED*TTCN-3 [10,11]. Data support was not considered but can be easily used to enhance expressiveness and integration into TTCN-3.

A prototype was implemented to demonstrate feasibility. A first version was demonstrated on the TTCN-3 launching event in October 2000 at *European Telecommunications Standards Institute* (ETSI). The used examples are converted with it. The prototype shares code with the IDL to TTCN-3 converter prototype also developed by the author.

In further work the prototype should support HMSC and concurrent TTCN which have been explained but not implemented until now. A discussion of concurrent TTCN-2 and MSC can be found in [8]. Usage of TTCN-3 as data language has also to be considered. There is a prototype enhancement available which supports the *time* concept as described by *TIMED*TTCN-3 [11]. Industrial interest on the prototype has been shown.

References

1. ISO/IEC: Information Technology — Open Systems Interconnection — Conformance Testing Methodology and Framework — Part 3: The Tree and Tabular

Combined Notation (second edition). International Standard 9646-3, ISO, International Organisation for Standardisation and IEC, International Electrotechnical Commission (1998)
2. OMG: Unified Modeling Language Specification. OMG Formal Document formal/03-03-01, OMG, Object Management Group (2003) Version 1.5.
3. OMG: Unified Modeling Language 2.0: Superstructure. OMG Final Adopted Specification ptc/03-08-02, OMG, Object Management Group (2003)
4. Jeckle, M., Rupp, C., Hahn, J., Zengler, B., Queins, S.: UML 2 glasklar. 1 edn. Hanser (2004)
5. ETSI: Methods for Testing and Specification (MTS) — The Testing and Test Control Notation version 3 — Part 1: TTCN-3 Core Language. European Standard ETSI ES 201 873-1 v2.2.1, ETSI, European Telecommunications Standards Institute, Sophia-Antipolis, France (2002)
6. ITU-T: Recommendation: Message Sequence Chart (MSC). International Standard Z.120 (11/99) with Corrigendum 1, ITU-T, International Telecommunication Union — Telecommunication Standardisation Sector SG 10 (2001)
7. Koch, B., Grabowski, J., Hogrefe, D., Schmitt, M.: Autolink – A Tool for Automatic Test Generation from SDL Specifications. In: IEEE International Workshop on Industrial Strength Formal Specification Techniques (WIFT'98), Boca Raton, Florida (1998)
8. Koch, B.: Test-purpose-based Test Generation for Distributed Test Architectures. doctoral thesis, Medizinische Universität zu Lübeck, Germany (2001)
9. Schmitt, M.: Automatic Test Generation Based on Formal Specifications — Practical Procedures for Efficient State Space Exploration and Improved Representation of Test Cases. doctoral thesis, Georg-August-Universität Göttingen, Germany (2003)
10. Dai, Z., Grabowski, J., Neukirchen, H.: *TIMED*TTCN-3 – A Real-Time Extension for TTCN-3. In Schieferdecker, I., König, H., Wolisz, A., eds.: Proceedings of the IFIP TC6/WG6.1 14^{th}International Conference on Testing of Communicating Systems, (TestCom 2002), Berlin, Germany, The International Federation for Information Processing, IFIP, Kluwer Academic Publishers (2002) 407–424
11. Dai, Z., Grabowski, J., Neukirchen, H.: *TIMED*TTCN-3 Based Graphical Real-Time Test Specification. In Hogrefe, D., Wiles, A., eds.: Proceedings of the IFIP TC6/WG6.1 15^{th}International Conference on Testing of Communicating Systems, (TestCom 2003), Sophia-Antipolis, France. Volume 2644 of Lecture Notes in Computer Science, (LNCS)., The International Federation for Information Processing, IFIP, Springer Verlag (2003) 110–127
12. Schieferdecker, I., Grabowski, J.: The Graphical Format of TTCN-3 in the context of MSC and UML. In Sherratt, E., ed.: Proceedings of the 3^{th} International Workshop on SDL and MSC (SAM 2002), Telecommunications and beyond: The Broader Applicability of SDL and MSC, Aberystwyth, UK, June 24.-26., 2002. Revised Papers. Volume 2599 of Lecture Notes in Computer Science, (LNCS)., Springer Verlag (2003) 233–252
13. Schieferdecker, I., Dai, Z., Grabowski, J., Rennoch, A.: The UML 2.0 Testing Profile and its Relation to TTCN-3. In Hogrefe, D., Wiles, A., eds.: Proceedings of the IFIP TC6/WG6.1 15^{th}International Conference on Testing of Communicating Systems, (TestCom 2003), Sophia-Antipolis, France. Volume 2644 of Lecture Notes in Computer Science, (LNCS)., The International Federation for Information Processing, IFIP, Springer Verlag (2003) 79–94

14. ETSI: Methods for Testing and Specification (MTS) — The Testing and Test Control Notation version 3 — Part 3: TTCN-3 Graphical Presentation Format (GFT). Technical Report ETSI TR 101 873-3 v.1.1.2, ETSI, European Telecommunications Standards Institute, Sophia-Antipolis, France (2002)
15. Ebner, M., Yin, A., Li, M.: Definition and Utilisation of OMG IDL to TTCN-3 Mappings. In Schieferdecker, I., König, H., Wolisz, A., eds.: Proceedings of the IFIP TC6/WG6.1 14^{th}International Conference on Testing of Communicating Systems, (TestCom 2002), Berlin, Germany, The International Federation for Information Processing, IFIP, Kluwer Academic Publishers (2002) 443–458
16. Grabowski, J., Koch, B., Schmitt, M., Hogrefe, D.: SDL and MSC Based Test Generation for Distributed Test Architectures. In R, D., v. Bochmann, G., Lahav, Y., eds.: SDL '99 The next Millenium – Proceedings of the Nineth SDL Forum, Montreal, Canada, Elsevier (1999)