ETSI TR 102 840 V1.1.1 (2009-09)

Technical Report

Methods for Testing and Specifications (MTS); Model driven testing in standardization

Reference

DTR/MTS-00106-ModDrivTesting

Keywords

conformance, interoperability, methodology, testing

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

Introduction

Carried out both within STFs and the Center for Testing and Interoperability (CTI), *test specification and test creation* for standardized systems is an important part of ETSI's mission today. In order to make the best tools and methods available to the Members, the technical committee on *Methods for Testing and Specification* (MTS) is continuously aiming, among others, to provide *methodologies for the specification of standardized tests including formal definition languages as well as the generation, processing and verification of test suites*, as the MTS chairman has noted. As part of this pursuit, MTS has created a work item to investigate the use of *model driven testing* as a complementary method for test design and test creation as well as to collect requirements on and recommendations related to this methodology, especially pertaining to the standardization context.

The rationale for this work item is that test creation is a resource-intensive and costly process, and that it is in the best interest of the Members to streamline and improve this process if possible. Model driven testing, a methodology and family of technologies for the automatic derivation of test descriptions and test cases from system models, has been taken successfully into use by early movers in industry verticals such as data- and telecommunications infrastructure, and has been reported to have realistic potential for significantly reducing the cost of test design and increasing the quality of test specifications.

ETSI has a long history in using various formal and semi-formal languages for modelling and specifying both systems and test systems, including but not limited to SDL, TTCN-3, and TPLan. Whereas the model driven testing methodology is not tied to any particular modelling notation (such as SDL or UML), nor to any output format (English, MSCs, TPLan or TTCN-3), this history provides a solid foundation for potential deployment of model driven test generation in the future.

The present document describes shortly in clause 4 the present manual process for test specification, and gives in clause 5 an overview on the model driven test generation approach. Clause 6 lists then requirements that stem from the standardization context, but in the form of recommendations, for modelling languages and tool chains used to potentially implement model driven test generation in the ETSI context. Clause 7 provides (early) recommendations for process-level pragmatic methodology that have been derived from industrial experience and adapted to the ETSI context.

The document is not normative and is provided for informational purposes only, with the hope that it provides a useful foundation for discussion, and hopefully in the future also for the definition and adoption of a methodology that will positively add to ETSI's capability to serve its Members in the test specification sphere in the early 21st century.

1 Scope

The present document presents a collection of recommendations for applying system model driven test generation in a standardization context, especially within ETSI.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific.

- For a specific reference, subsequent revisions do not apply.
- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:
  - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;
  - for informative references.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are indispensable for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

Not applicable.

2.2 Informative references

The following referenced documents are not essential to the use of the present document but they assist the user with regard to a particular subject area**.** For non-specific references, the latest version of the referenced document (including any amendments) applies.

[i.1] ETSI EG 202 237: "Methods for Testing and Specification (MTS); Internet Protocol Testing (IPT); Generic approach to interoperability testing".

[i.2] ISO/IEC 9646-1: "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions apply:

Implementation Under Test (IUT): See ISO 9646-1 [i.2].

Implementation Conformance Statement (ICS): statement made by the supplier of an IUT claimed to conform to a given specification, stating which capabilities have been implemented

Implementation eXtra Information for Testing (IXIT): statement made by a supplier of an IUT which contains or references all of the information related to the IUT and its testing environment, which will enable the test laboratory to run an appropriate test suite against the IUT

system model: computer-readable behavioural model that describes the intended external operational characteristics of a system, i.e. how the system being modelled interacts with its environment

System Under Test (SUT): See ISO 9646-1 [i.2].

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

CSCF	Call Session Control Function
CTI	Center for Testing and Interoperability
ICS	Implementation Conformance Statement
I-CSCF	Interrogating-CSCF
IFS	Interoperable Functions Statement
IMS	IP Media Subsystem
IP	Internet Protocol
IUT	Implementation Under Test
IXIT	Implementation eXtra Information for Testing
MTS	Methods for Testing and Specification
P-CSCF	Proxy-CSCF
PDU	Protocol Data Unit
S-CSCF	Serving-CSCF
SDL	Specification and Description Language
SIP	Session Initiation Protocol
SUT	System Under Test
TC	Test Cases
TD	Test Descriptions
TP	Test Purposes
TSS	Test Group Structure
TTCN	Testing and Test Control Notation
UML	Unified Modelling Language
XML	Extensible Markup Language

4 Standardized test specification development at ETSI

Next to producing standardized base specifications for a variety of technologies, ETSI also has a strong reputation in producing standardized test specifications. As specified in EG 202 237 [i.1], these test specifications can serve one of two purposes: they can either help to assess if an implementation conforms to a standard, i.e. conformance test specifications, or they can be used to assess if two or more implementations interoperate properly with each other, i.e. interoperability test specifications.

In the development of both conformance and interoperability test specifications, ETSI has traditionally followed a stepwise approach based on the methodology defined in ISO 9646-1 [i.2] and resulting in a number of different test specification deliverables. Figure 1 illustrates this approach. These steps can be understood as different levels of abstraction that bridge the large intellectual gap between a base specification and the final executable test suite. They not only form an essential framework for test specification but also enable a common understanding of the complete test specification for different target audiences, e.g. standardization experts, technology experts, managers, and test engineers.

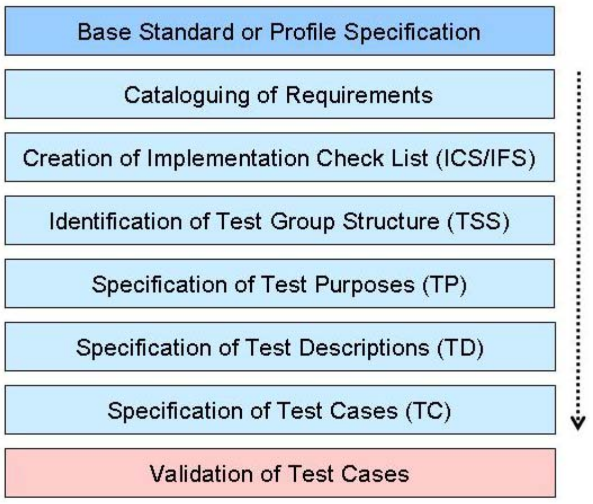
Base Standard or Profile Specification
Cataloguing of Requirements
Creation of Implementation Check List (ICS/IFS)
Identification of Test Group Structure (TSS)
Specification of Test Purposes (TP)
Specification of Test Descriptions (TD)
Specification of Test Cases (TC)
Validation of Test Cases

Figure 1. ETSI (Test) Specification Development

In the first step, requirements are identified from one or more base specifications. These may be catalogued and published in a Requirements Catalogue. Then the Implementation Conformance or Interoperable Functions Statement (ICS/IFS) is constructed. These both are essentially high level check lists for features and capabilities in a standard. The check lists are filled out by system vendors, according to which features they implement in their products. This information can help to determine if two implementations of the same standard have potential to interoperate.

In the next step one or more test purposes are specified for each testable identified requirement, either in English prose, or in the TPLan notation [TPLan]. A test purpose formulates (an aspect of) a requirement as a set of IUT pre-conditions, stimuli, and responses, and specifies test verdict criteria. The testability of a requirement is affected by the type of testing to be carried out, e.g. requirements related to error management cannot be assessed using interoperability tests because many error conditions cannot be triggered by conforming implementations.

After the definition of test purposes, an informal test description can be specified in English prose, as tables, or as message sequences covering usually one, but sometimes multiple test purposes. Test descriptions extend test purposes by providing more detailed information of preambles and postambles. Test descriptions are by definition not executable.

The preambles and postambles in test descriptions are not conceptually the focus of testing, even though they in practice depend on identified requirements in the same way as test bodies do. Therefore, conventionally preambles tend to be specified to invoke behaviour that is most likely to be correctly implemented. In addition, preambles tend to be reused across test descriptions as much as possible.

In the next step, executable test cases are potentially specified, usually in TTCN-3, which is a platform and SUT independent test specification language [CORE]. Test cases tend to be specified using multiple concurrently executing test components for stimulating and observing different logical interfaces of the IUT, i.e. one test component per one abstract test interface. Another key concept in the specification of test cases is the use of IXIT, which means testing parameters, such as parameterized message fields, that can be specified at test execution time. IXIT is frequently used for example to control TTCN-3 test execution based on the ICS or IFS.

Finally, each test description or test case has to be validated (usually outside of ETSI) to ensure that it is correctly specified, e.g. by executing a test description at an interoperability event or by running a test case from a conformance test tool against a number of different implementations of a given standard. So far transitions between the different steps, e.g. the specification of test descriptions from or for test purposes, have been always performed manually at ETSI.

NOTE: Some of the steps described above may be omitted in the development of test specifications , e.g. requirements are not always explicitly catalogued, interoperability test specifications are usually only refined to the level of test descriptions, conformance test specifications skip usually the specification of test descriptions, etc.

5 System model driven test generation

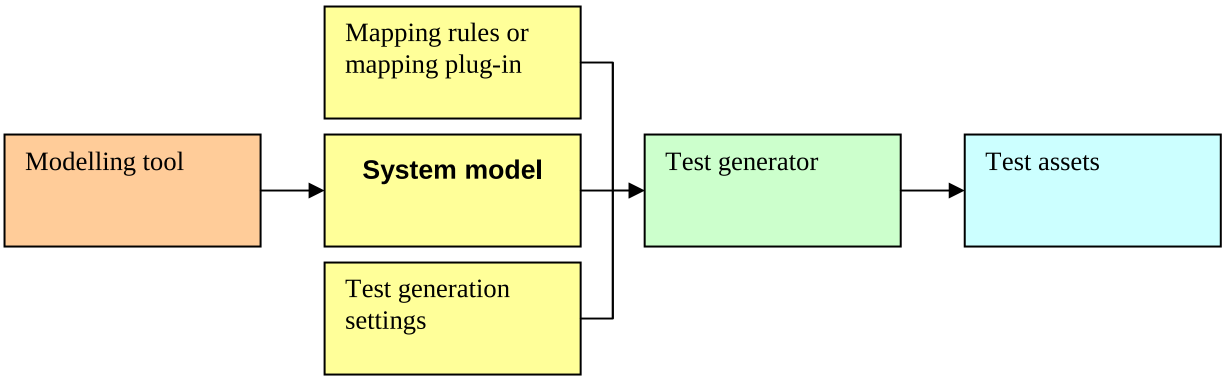

Figure 2: System model driven test generation process

For many companies and institutions, test case design is a resource-consuming and error-prone task. One approach to reducing the resourcing burden and increasing the quality of the test case design process is system model driven test generation (see figure 2).

In this approach, people responsible for test case design provide a model of the behaviour of the IUT at the abstract test system interface. This model describes the intended behaviour of the IUT (which includes also error management features) at a chosen level of abstraction. A test generator derives from the model a set of test cases or test descriptions, which are sequences of logical messages, to (inbound) and from (outbound) the test interface. The behaviour specified in a protocol standard is usually quite straightforward to model (although not necessarily easy) and the automatic generation of conformance tests from such models is well understood. If a method of modelling behaviour at the user or application level can be identified then the same concept can be also applied in an interoperability testing context where there may be more than one IUT being driven by the derived tests.

These test cases can be rendered into test assets, e.g. test cases in TTCN-3, MSCs or textual test descriptions. Depending on the tooling, the mapping of the logical test cases into "physical" assets is configured by mapping rules, mapping plug-ins, or some other similar approaches.

The definition of what constitutes a "good" test suite to be derived from a given system model is rather open-ended as the system model itself typically defines a large space of potential tests. Therefore a system model driven test generation capability is usually augmented by a capability to modify the test selection rules and heuristics.

NOTE 1: To warrant the acceptance of model driven testing in the standardization community such heuristics should be defined in such a way that they generate results similar to manual test design methods applied in this domain.

The system model driven approach provides a system-centric view as well as an automated approach to test generation and selection. It is appropriate in contexts where specification and validation are driven by articulated functional requirements.

NOTE 2: In general, modelling the intended behaviour of systems provides also additional benefits such as potential capability to verify the behaviour or analyze it statically, but these aspects are beyond the scope of the present document.

5.1 Modelling

For the purposes of system model driven test generation, a model of an IUT is a:

1) computer-readable;
2) open;
3) abstracted; and
4) partial description of the intended behaviour of the IUT at the test interface.

The model will be computer-readable in order to apply an automated test generation approach.

The model will be open in the sense that it describes a system that operates inside an environment that is not specified. This is crucial for test generation, as it is the operating environment that is being synthesized in the form of test cases.

The model will always be abstracted, but the abstraction level may vary. In general, the model driven test generation approach can be applied to systems on very detailed specification level (e.g. functional testing mechatronics controllers) as well as on very abstracted specification level (e.g. interoperability requirements). From the standardization perspective, a high level of abstraction can usually be assumed.

The model is also typically partial in the sense that only some aspects of the functionality of the IUT are described. For example, an IUT could have both standardized and proprietary interfaces, and a model could describe the behaviour at the standardized interfaces only.

From this perspective, the essential aspect of being a model is not being graphical (even though usually models are expressed at least partially as diagrams), but that it models the real system under test, and this from a viewpoint that is useful for test case derivation.

5.2 Test case construction

Test cases are constructed by the model driven test generator basically by finding a set of external behaviours such that:

1) the behaviours in the set are possible to obtain from the model by simulating it (correctness criterion); and
2) they cover different aspects of the system sufficiently well as specified by the user (comprehensiveness criterion). The external behaviours (inbound and outbound messages) are then presented as test cases. The correctness criterion is typically fixed, whereas the comprehensiveness criterion depends on the testing goals. For example, the user might dictate that there will be at least one test case for every functional requirement references from the model (assuming a system for tracing to functional requirements from models), and a test set that does not cover all the requirements would then be considered not comprehensive.

Coverage is a quality measure of the overall test specification that describes the degree to which the model of a system has been covered by the generated test cases and therefore comprehensiveness criterion can be defined based on coverage of embedded requirements and model elements. A test case covers a specific coverage goal if the execution of the test against the model itself leads to the meeting the specified goal. In addition to requirement coverage, model driven coverage criteria include for example control flow, data flow, and condition coverage. The present document concentrates primarily on the requirement coverage.

5.3 Test asset generation

Test assets are generated from the logical test cases derived by the model driven test generator. The technologies used to render the test cases vary. Typically the output can be configured using mapping rules or plug-ins, or there is a generic intermediate output format such as XML that can be then post-processed to obtain the test assets. Typically generated test assets include English text, message sequence charts, tables, and executable tests in languages like TTCN, TCL or Java.

5.4 Requirements traceability

An important aspect of any test specification process is requirements traceability, i.e. the possibility to trace individual test cases to the requirements they (partially) cover. This is particularly important in the context of model driven test generation as test design is automated and there is no human-written explanation of the intent of individual test cases.

6 Recommendations on languages and tools

This clause enlists recommendations on a modelling language, a modelling tool supporting it, and a test generation tool that are together used in standardization context.

In this context, the modelling language is the formal notation and the accompanying facilities for describing intended system behaviour. The modelling tool is the tool or tool chain used to create, modify and manage system models. The test generation tool is the tool or tool chain used to derive tests from system models employing the system model driven test generation paradigm.

6.1 Modelling language support

6.1.1 Standardized languages

It is recommended that the modelling language should be standardized by an international standards body as a whole or in part.

RATIONALE: Using a standardized language might improve the odds of continuing tools support in the future.

6.1.2 Widely used languages

It is recommended that the modelling language or its parts should be widely recognized and known and used actively today in the relevant industries.

RATIONALE: De facto industrial standard languages with large user communities tend to be more long-living, and it is easier to find people mastering them.

6.1.3 Model exchange

It is recommended that models created in the modelling language should be exchangeable between interoperating modelling tools.

RATIONALE: The possibility of model exchange increases the odds of having multiple interoperable tool implementations for the modelling language.

6.1.4 Availability of multiple tools

It is recommended that there should be more than one modelling tool and more than one test generation tool that supports the modelling language, or that there should be tools that support alternative modelling languages into which models expressed in the modelling language can be translated automatically or semi-automatically.

RATIONALE: The availability of alternative tools and tool chains increases the possibility for continuous tool support, and also typically drives down direct and indirect tool chain ownership and use costs.

6.2 Modelling language features

6.2.1 Expressivity

The modelling language should be capable of expressing the behaviour of communicating systems on a level of abstraction that matches the level of abstraction in standards that ETSI works with.

Secondarily, it is recommended that the modelling language supports modular and compositional constructs that enable creation of reusable models and the reuse of them.

It is recommended that the modelling language should support the expression of time properties and timers, as these are important in the communications sphere.

It is recommended that the modelling language should allow the specification of systems with multiple interfaces and to naming of those interfaces.

The modelling language should allow the definition of complex message types in a type system compatible with or comparable to the TTCN-3 type system, to the extent possible.

RATIONALE: The TTCN-3 type system has been designed based on good understanding of how communicating systems are designed and tested, so it forms a good reference point. Also, an aligned or comparable type system should make it easy to map the computer-generated test cases into TTCN-3.

It is also recommended that the modelling language should support the description of systems in terms of multiple concurrently executing components in order to allow for compositional construction of system models.

EXAMPLE: Constructing a core IMS signalling model from models for P-CSCF, I-CSCF, S-CSCF and other similar IMS node models, or creating models with multiple SUTs in the context of interoperability testing.

6.2.2 Modelling of underspecified features

It is recommended that the modelling language supports the modelling of underspecified features, such as one or more fields within a complex message structure whose value or values in certain conditions are not defined, and so that the information about under specification can be propagated to the generated tests.

RATIONALE: Underspecified features or message fields appear commonly in the standardization context.

6.2.3 Modelling of optional and conditional features

It is recommended that the modelling language should support the modelling of optional or conditional features, so that the optionality information can be propagated to the generated tests.

RATIONALE: Optional and conditional features (features depending on optional features) appear often in standards, and they need to be accounted for in test descriptions that target multiple implementations that differ in the optional and conditional features.

6.2.4 Expressing test configurations

It is recommended that the modelling language supports the expression of the existence of multiple test configurations in an efficient manner, allowing the user to generate tests for different test configurations from a single model or a family of models. For example, in the IMS context it should be possible to reflect situations involving interworking, roaming, non-existing users, in/excluded IMS application servers and in/excluded DNS ENUM translation servers.

RATIONALE: If the modelling language forces the user to hard-code a singular test configuration, only part of the required test cases can be derived from any single model, as in practice usually multiple test configurations are required for testing a (complex) standardized system.

6.2.5 Requirement annotations

It is recommended that the modelling language should provide a facility for linking behaviours described in the model to requirement identifiers such as clause and paragraph references in standardization documents, so that it is possible to trace back to requirements from generated tests.

RATIONALE: Requirements traceability information is crucial for understanding the tests, and for enabling the efficient transfer of feedback information from test execution and domain experts to people maintaining the system models. Requirements traceability can be also used to organize and order test cases, and to guide test execution.

If the annotation facility links requirements to individual syntactic elements of a model, it should be possible to associate requirements at least with model transitions and states (in state-based models) and conditions. It is also recommended that requirements should be possible to be associated with individual message fields and other data flow objects as appropriate.

It is recommended also that optional and requirement dependencies can be represented.

It is also recommended that there should be a facility for grouping requirements and defining categorical hierarchies of them.

RATIONALE: Requirement groups and hierarchies provide an efficient vocabulary for test case management.

6.3 General tool requirements

6.3.1 General availability

It is recommended that the modelling tool and the test generation tool should both be generally available.

RATIONALE: Generally available tools can be accessed relatively easily by ETSI members as necessary.

6.3.2 Platform support

It is recommended that the modelling tool and the test generation tool should both be available on multiple operating system platforms, or be platform independent.

RATIONALE: Some ETSI members have active open source initiatives and have large deployments of non-Windows based systems. A Windows-only tool chain would restrict the availability of the tool chain to ETSI members in an unduly fashion.

6.3.3 Model import and export

It is recommended that the modelling tool should allow for importing models from other modelling tools or modelling formats, and should allow for exporting models in interchangeable formats to facilitate model exchange.

RATIONALE: Model interchange capability reduces the risk of tool chain lock-in and makes it easier for ETSI members to benefit from the models and to develop then further.

6.4 Test case generation, selection and organization

6.4.1 Requirements traceability

It is recommended that the test generation tool should understand requirement annotations that are present in a model, and to use the annotations to provide requirements traceability information in the generated test cases. The test generation tool should be able to list for every generated test case a set of requirements that potentially affect the logic of the test case, and which are thus covered by the test case, and to map the requirements to individual test steps and test message parts to the extent possible.

6.4.2 Minimal coverage of new requirements per test case

It is recommended that the test generation tool allows for a test construction option where a separate test case is generated for every requirement annotated in the model, and that the test cases otherwise cover as few requirements as possible.

RATIONALE: This helps to keep the test cases as independent of each other as possible and also eases test result analysis and reporting.

6.4.3 Identification of preambles and postambles

It is recommended that the test generation tool should be able to make a distinction within a test case between the preamble, the postamble, and the actual test, following the present ETSI methodology as closely as possible.

RATIONALE: This aligns the automatically generated output with manually designed test cases and test descriptions within the ETSI processes.

6.4.4 Preamble and postamble selection

It is recommended that the test generation tool allows the user to influence preamble and postamble selection based on requirement categories, for example so that the test generation tools avoids covering in preambles requirements pertaining to error management.

RATIONALE: This maximizes to chances to reach actual test bodies during test execution.

6.4.5 Test case dependencies

It is recommended that the test generation tool should be able to produce a report on test case dependencies.

RATIONALE: A test case *a* depends on test case *b* then failing to execute test case *b* implies that test case *a* is also likely to fail. If these dependencies are documented, it speeds up the test execution process in the case of a partially incorrect IUT.

6.4.6 Test case ordering

It is recommended that the test generation tool should be capable of ordering the generated test cases according to the grouping of requirements, e.g. grouping tests related to mandatory, conditional, or optional requirements, tests relating to a specific functionality, or tests related to error handling from other tests.

Additionally, it is recommended that the test generation tool should be capable of ordering the test cases based on their dependency ordering.

RATIONALE: Proper ordering of test cases makes it easier to understand and manage the test collection later in the process. Also, it is in general proper to start to test a system from its basic functionality and to move to advanced and optional functionality only later.

6.4.7 Test cases ending in the starting state

It is recommended that the test generation tool should allow for the construction of test cases so that every test case will, assuming it is successfully carried out, return the SUT in the same state (at least conceptually) where it was prior to the execution of the test case.

In the case of executable tests, the executable tests should bring the system back to the starting state even in the case of failure conditions, to the extent possible.

RATIONALE: This guarantees that test cases can be executed independently of each other, and if so desired in alternative order, or some test cases can be omitted from execution based on external requirements on the test execution process.

6.4.8 Support for testing underspecified behaviour

It is recommended that the test generation tool should be able to generate test cases that correctly allow for multiple or only partially defined output messages or behaviours of the IUT in the case of underspecified features.

RATIONALE: There are cases where in the standardization context the behavior of the system under test has been intentionally left not fully specified. In these cases it is important to be able to accept all behaviors allowed by the standard.

EXAMPLE: The IMS core network specification allows the network to answer to an INVITE message in specific conditions either with an error message or with an informational message, and if no further information about the configuration is available, both behaviours should be allowed in the corresponding test cases. Note that after these two different answers the IMS core network is conceptually in two different states.

6.4.9 Support for generating undefined inputs

It is recommended that the test generation tool should be able to generate tests where some message fields in inbound complex message structures are not defined, reflecting underspecified inputs in the model.

RATIONALE: There are many cases where standards do not fully specify the values of all input fields, and thus no particular values should be assumed in test cases for such fields when non-executable test cases are being constructed.

NOTE: In the case of executable test cases, the values for the underspecified fields can be selected either by the test generation tool, or within the test adaptation layer.

6.4.10 Conformance testing

It is recommended that the test generation tool should be able to generate conformance tests, i.e. tests that drive an SUT from one or multiple interfaces in order to check its conformance to a standard.

6.4.11 Interoperability testing

It is recommended that the test generation tool should be able to generate interoperability tests, i.e. tests that drive more than one SUT from one or multiple interfaces in order to verify that the SUTs can interoperate.

RATIONALE: Model driven generation of interoperability tests can increase the coverage of interoperability testing and hence contribute to the interoperability of systems that ETSI members operate with.

Additionally, it is recommended that the test generation tool should support the generation of interoperability tests with conformance checking, i.e. tests where the messaging between the SUTs is also verified against the system model (passive testing).

RATIONALE: Passive monitoring of message flows between SUTs in interoperability testing provides for a cost-efficient conformance checking without disrupting the interoperability test in any fashion.

NOTE: Although the automatic generation of interoperability test specifications is desirable, it is recognized that considerable study is required into the modelling of interoperable behaviour and the extraction of tests from it.

6.5 Generation of test descriptions

It is recommended that the test generation tool can be configured to produce test descriptions in a human-readable format.

The generated test descriptions should be structured so that they represent passing scenarios only, with the understanding that a test execution that deviates from a test description, for example in its message flow or message data or timings, prior to its conclusion, should result in a failing test verdict.

NOTE: The above paragraph is in practice a requirement on the structure of test descriptions at ETSI in general, and should be documented elsewhere.

Message sequence diagrams should be produced to illustrate the message flows at the interfaces between external entities and the IUT as well as, in the case of interoperability testing, the message flows between the IUTs.

RATIONALE: Many ETSI member experts are very familiar with message sequence diagrams and therefore they are a good format for illustrating the generated tests.

Time, time constraints, data and data constraints on the exchanged messages should be presented also as applicable. Especially undefined values should have a clearly recognizable presentation.

RATIONALE: It is not enough to present message types only as typically message data and timing information has crucial effect on the behaviour of an SUT.

The presentation of the tests should allow for the easy identification of the preamble, test body, and postamble phases. In the preamble clauses, message data could be presented only to an extent critical to understanding how to reach the test precondition, e.g. message types or fields directly affecting the ability to check for data constraints in the test body.

EXAMPLE: In the case of an IMS test the checking of the correct handling of the P-Asserted-Identity header in messages relies on the existence of appropriate user profiles in the SUT, e.g. that a user has a public user identity following the Tel URI format. Here, as part of the preamble a user has to register using a specific user name in the Request URI field in REGISTER requests.

The generated test descriptions should contain references to the requirements covered by the individual tests as well as to the required SUT configuration. For informational purposes, requirement traceability should be also provided for preambles and postambles in addition to test bodies.

RATIONALE: It is helpful to be able to quickly identify all the requirements that a specific test description depends on, in addition to which requirements are actually targeted by the test (raison d'être, i.e. why the test was generated).

6.6 Generation of executable TTCN-3 tests

In general, it is recommended that the test generation tool should be able to be configured to generate TTCN-3 that follows the TTCN-3 standard [CORE, OS] and the guidelines presented in [IPT_FWK].

NOTE: The TTCN-3 code that is generated is what is known as the abstract test suite. An adaptation layer, including proper implementations of the TRI [TRI] and TCI [TCI] interfaces, will be needed to produce an actual executable test suite. The adaptation layer would not be typically be produced by the test generation tool.

6.6.1 Syntax

It is recommended that the test generation tool should support, or be able to be configured to support, the generation of TTCN-3 code that follows the TTCN-3 naming conventions specified in [IPT_FWK].

EXAMPLE: Prefix TTCN-3 variable identifiers with "v_".

The generated TTCN-3 should reuse identifiers used in the models to allow the tracing of test execution to the model as easily as possible.

6.6.2 Structure

It is recommended that the test generation tool should support, or be able to be configured to support, the generation of TTCN-3 code where definitions are collected in the following types of modules:

1) TypesAndValues: contains only type and constant definitions.
2) Templates: contains only templates.
3) Functions: contains only functions and altsteps.
4) TestCases: contains only test case statements and potentially functions that are referenced in start statements.
5) ModuleParams: contains only module parameter definition.
6) Interface: contains only port type definitions and potentially component type definitions (in case of libraries).
7) TestSystem: contains only component type definitions which are type compatible to library component types (if applicable).
8) TestControl: contains only the control part.
9) TestConfiguration: contains functions with map/unmap/connect/disconnect operations and interactions with the SUT adapter.

It should be possible to generate a separate set of modules for separate interface types, and to generate parallel test components for individual test interfaces.

EXAMPLE 1: The test definitions and test components for an IMS system that has a SIP and a DIAMETER interfaces could be kept separate to increase readability and facilitate test system understanding.

There should be comments including *@desc* and *@param* tags for every *testcase*, *altstep*, *function,* and *modulepar* in the generated code.

The generated TTCN-3 should be organized so that variable initialization, configuration, preamble, test body, and postamble clauses are identified by comments or by structure (e.g. separate functions).

In the generated code, type definitions should be in alphabetic order. Within the type module, the PDU message types should be collected in a single group.

There should not be *goto* statements in the generated code and not nested *alt* statements.

It is recommended that the test generation tool should be able to be configured to reuse TTCN-3 code fragments and definitions from other sources.

EXAMPLE 2: The test generation tool could generate TTCN-3 that depends on the definitions and synchronization primitives in the ETSI Common TTCN-3 Library.

EXAMPLE 3: The generation of type definitions could be omitted in favour or referring to a pre-existing, hand-written set of type declarations for a test interface, or test adapter configuration functions.

6.6.3 Behaviour

TTCN-3 code generated by the Test Generation tool should set test verdicts in all the possible execution branches of the generated tests.

Each *setverdict* operation with verdicts INCONC or FAIL should be preceded by a relevant *log* statement.

6.6.4 Configurability and late binding

It is recommended that the test generation tool should support generation of TTCN-3 code that is parameterized and can be configured during test execution time on the IXIT level, e.g. to bind the actual IP addresses of SUT components.

7 Recommended methodology

7.1 Using model driven testing in a test design project

The usual ETSI procedure for test definition is explained briefly in clause 4.

When model driven test generation technology is employed for test design, particular attention will be paid to minimizing changes to the test design process that are visible outside the group of people using the model driven test generation technique.

As the test descriptions and test cases are the primary output from a test design effort, the computer generated test descriptions and test cases should resemble in format the human-written ones as closely as possible. In particular, the test assets should be easy to review by humans and to edit later by hand if necessary. How easy this is to implement depends of course on the test generation tool used.

7.2 Requirements traceability

If a requirements catalogue exists, and a model is constructed directly based on that, the model should be annotated with requirement annotations that refer to entries of the catalogue.

If a set of test purposes has been already designed based on the requirements (e.g. in TPLan [TPLAN]), and a model is constructed based on the test purposes, the model should be annotated with requirement annotations that refer to the test purposes as functional requirements.

Alternatively, both requirements catalogue items and test purpose identifiers can be used in a single model.

In the case a model is constructed directly based on standards documents without existing requirements catalogues or test purposes, references to paragraphs and clauses in the standards documents should be used as requirement annotations in the model.

7.3 What to generate

In principle, model driven test generation might be used to generate either test purposes, test descriptions, or executable test cases.

However, it is recommended that model driven test generation is not used to generate test purposes, but only test descriptions and executable test cases. Both test descriptions and executable test cases may be generated from the same model. In this case, attention should be paid to the fact that the executable test cases are not in fact derivatives of the test descriptions, but they are both direct derivatives of the model.

However, test purposes can be well used as an input for model construction. In this case, the model should refer to the test purposes using the requirements traceability mechanisms available.

It is not recommended that a model is used to generate test descriptions, and then test cases are derived by hand from the test descriptions. Instead it is recommended to add extra detail to the model as necessary and to configure the test generation tool correctly to generate also the test cases. This is especially important when test case maintenance is considered.

7.4 Selection of tools

When a model driven test generation tool chain is used in an ETSI context, it is recommended that the tool chain is selected based on both the requirements listed in clause 6 and the particular needs of the project.

7.5 Required skill set

It is recommended that a model driven test generation effort should be staffed with people who are adept in programming.

RATIONALE: Depending on the tool set, the level of abstraction and the system under test, the models can be mostly graphical diagrams whose construction requires very little programming, or they can be almost exclusively textual computer programs. But as a rule, it is difficult to model systems that handle data without programming skills.

7.6 Normative and non-normative models

The system models used in model driven test generation can either be treated as normative or non-normative references. Most often the model should be considered as a non-normative and intermediate artifact that is mostly internal to the test design process, because it has been observed that people tend to think that a model that is normative regarding the external behaviour is also normative or "suggestive" regarding implementation structure and architecture. This does not rule out why models could not be treated as normative references especially when they can be provided to ETSI members and technical bodies as useful artifacts.

7.7 Normative and non-normative tests

Model driven test generation can be used to produce both normative and non-normative test descriptions and test cases.

Hence it is recommended that in those cases where resources allow, models should be developed so that it is possible to generate from them both a concise, normative test set, and a larger, non-normative test set for additional benefit to the ETSI members.

EXAMPLE: An IMS model that is used to generate a test suite can be hard-coded to support exactly one session, or it can be developed so that it has potential for generating test cases that involve multiple simultaneous sessions. Typically, only single-sessions test cases would be included in a basic, normative conformance test suite. But multi-session test cases could be valuable to ETSI members as a more through way to ensure the quality of the implementation, for example regarding the proper implementation of concurrent and parallel behaviour-an IMS node, say, could deadlock or crash when multiple sessions are initiated towards the same UE in an interleaved fashion.

RATIONALE: Once the model is created, deriving test sets of different "depth" from it is typically a low-cost effort.

7.8 Reviewing procedures

Because tests generated from a model cannot be of higher quality than the model itself, both model and test case reviews should be conducted when model driven test generation is deployed.

The purpose of a model review is to make sure that the model architecture is sound and enables reuse and future extension (as required), and that the behaviour described in the model matches with the understood, correct behaviour of the system under test. Model reviews can be carried out only by people who are familiar with the modelling language, however. In order to enable wider reviewing, it is important to review the generated tests also.

The purpose of a computer generated test description or test case review is to verify that the behaviour in the test cases matches with the understood, correct behaviour of the system under test. As the test cases are depictions of linear interactions with the system under test, they can be easily reviewed by domain experts even if they are not familiar with the modelling language that was used to create a model from which the test cases were derived. The test case review is conducted on the test case level and is analogous to reviewing hand-written test descriptions or test cases.

7.9 Maintenance

If there are errors in test descriptions or test cases that are spotted only later in the process, it is possible to fix the test descriptions and test cases by hand. In this case, the changes should be carefully propagated back to the original model as soon as possible.

However, it is recommended that when possible, the changes should be immediately implemented in the model, and then the test cases regenerated instead of modifying (generated) tests manually. This enforces that any changes in the test set are reflections of changes in the model, and the model and the test set are in synchrony.

It is recommended that models are stored in a version control system and are properly change managed.

It is also recommended that whenever a set of tests is released outside the team working on model driven testing, the test set is **also** stored in version control system. This enables people to access different versions of the tests even when they do not have access to the model driven test generation tool, or to the different model versions, or do not know how to operate the tool.

7.10 Taking model driven testing into use

When model driven testing is taken into use in a new context or a new team, it is recommended that a person or team who has already experience in model driven test generation is consulted first.

From risk management perspective, the first deployments should be carried out in tasks that are not on the project's critical path, in order to account for e.g. underestimated deficiencies in the team's skill set.

7.11 Ceasing use of model driven test generation

It is possible that a team or a project needs to cease using model driven test generation, for example due to changes in the team's skill set, or due to any political, procedural or general reasons.

In this case, it is recommended that the use of model driven test generation is ceased in a controlled fashion, and that care is taken to generate a full, master test set from the last version of the model and store that test set in version control system. The model should be also properly archived in case it is needed later. After this, all further developments of the test set can then proceed using alternative means.

8 Conclusions

Test case design is a resource-consuming and error-prone task. One approach to reducing the resourcing burden and increasing the quality of the test case design process is model driven testing, a methodology and family of technologies for the automatic derivation of test descriptions and test cases from system models. In this approach, a model of the behaviour of the IUT at the abstract test system interface is provided that describes the intended behaviour of the IUT at a chosen level of abstraction. A test generator derives from the model a set of test cases or test descriptions, which are sequences of logical messages, to (inbound) and from (outbound) the test interface.

The present document has given a lists of requirements for modelling languages and tool chains used to potentially implement model driven test generation in the ETSI context. These requirements stem from the standardization context, but in the form of recommendations. A special attention has been given to test case generation, test selection, and organization, following the present ETSI methodologies as closely as possible.

In addition, the document has provided early recommendations for process-level pragmatic methodology that have been derived from industrial experience and adapted to the ETSI context.

It has been identified that methods for automatic generation of interoperability tests still require further research even though automatic generation of conformance tests is well understood currently.

Annex A (informative): Bibliography

ETSI ES 201 873-1: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

ETSI ES 202 553: "Methods for Testing and Specification (MTS); TPLan: A notation for expressing Test Purposes".

ETSI EG 202 568: "Methods for Testing and Specification (MTS); Internet Protocol Testing (IPT); Testing: Methodology and Framework".

ETSI ES 201 873-4: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 4: TTCN-3 Operational Semantics".

ETSI ES 201 873-6: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI)".

ETSI ES 201 873-5: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)".

History

Document history		
V1.1.1	September 2009	Publication