

**Methods for Testing and Specification (MTS);
Model-Based Testing (MBT);
Requirements for Modelling Notations**



Reference

DES/MTS-00128 MBTmodConcept

Keywords

testing, TTCN

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

Important notice

Individual copies of the present document can be downloaded from:

<http://www.etsi.org>

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at

<http://portal.etsi.org/tb/status/status.asp>

If you find errors in the present document, please send your comment to one of the following services:

http://portal.etsi.org/chaicor/ETSI_support.asp

Copyright Notification

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 2011.
All rights reserved.

DECT™, **PLUGTESTS™**, **UMTS™**, **TIPHON™**, the TIPHON logo and the ETSI logo are Trade Marks of ETSI registered for the benefit of its Members.

3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.

LTE™ is a Trade Mark of ETSI currently being registered

for the benefit of its Members and of the 3GPP Organizational Partners.

GSM® and the GSM logo are Trade Marks registered and owned by the GSM Association.

Contents

Intellectual Property Rights	4
Foreword.....	4
Introduction	4
1 Scope	5
2 References	5
2.1 Normative references	5
2.2 Informative references.....	5
3 Definitions and abbreviations.....	6
3.1 Definitions	6
3.2 Abbreviations	7
4 Model-based test development.....	7
5 General modelling notation requirements	8
5.1 Modularization	8
5.2 Algorithms.....	9
5.3 Documentation	9
6 Modelling the system interface	10
6.1 Actions	10
6.2 Operations	10
6.3 Ports.....	11
6.4 Configurations	11
7 Modelling the system behaviour	11
7.1 System state.....	11
7.2 System state transitions	12
7.3 Non-determinism.....	13
Annex A (informative): Examples of modelling notation styles	14
A.1 Rule-Based Notation	14
A.2 State Chart Notation.....	14
A.3 Process-oriented notation	15
History	16

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: "*Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards*", which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<http://webapp.etsi.org/IPR/home.asp>).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This final draft ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS), and is now submitted for the ETSI standards Membership Approval Procedure.

Introduction

Based on the recent success and deployment of automated test design by use of models in industry, TC MTS investigated work on model-based testing specifically in the context of standardized test specification development [i.1]. Contrary to other methods and approaches, which focus mainly on automation of test *execution*, the present document considers the use of model-based testing for the automation of test *design*.

Model-based testing facilitates a more thorough and earlier validation of standards as well as the efficient automatic generation of test case artefacts, e.g. in a textual or tabular descriptions, scripts or programs, which perform testing of the external behaviour of a system. Due to its independence of the output format and its higher level of abstraction, model-based testing enables a more direct review of the requirements imposed by a standard compared to test case artefacts. In addition, automation of test design allows ETSI as well as other organizations to more efficiently create test suites, coping with the ever-growing demand for interoperability and conformance testing in standardization.

The motivations for the development of the present document were:

- to collect in one document agreed terminology and concepts required for the specification of models specifically for testing for all interest groups that are exposed to model-based testing technology such as product vendors, makers of model-based testing tools, test service providers, test engineers, software developers, government agencies, procurement personnel and researchers
- to support the specification of models for derivation of standardized conformance and interoperability test cases
- to facilitate the use of model-based testing for product certification
- to create a basis for an open, competitive model-based testing tool market which process such models and where such models can be exchanged between different tools
- to enable consumer accountability (including also legal issues)

To ensure its success and quality, the present document has been developed by a group of experts from all types of stakeholders involved in test specification development, i.e. researchers, tool makers, industrial users, as well as testing experts of ETSI's Centre for Testing and Interoperability.

The present document lays the foundation for the deployment of model-based testing in standardization since it specifies requirements for modelling notations to be suitable for the generation of tests in the context of standardization. Such tests need to adhere to well established concepts defined and used in manual test specification [i.2]. In addition, the present document defines the criteria that need to be fulfilled by a model in order to be included in a standardized ETSI test specification, and the relation that models have to the generated tests.

1 Scope

The present document identifies and collects all concepts of a modelling notation required for specifying models in particular but not limited to the purpose of functional testing of communicating systems. Such models form the basis for generating abstract test cases which follow the principles of ISO/IEC 9646-1 [i.2] as, for example, put forward in the TTCN-3 test suites [i.3]. Model-based testing presents an alternative to manual test design, but does not eliminate the need for test systems [i.4], [i.5] which complement and execute generated test cases automatically. Model-based testing tools that use a modelling notation that complies with the requirements stated in the present document can be used to automatically generate abstract test cases suitable for standardization.

The concepts and requirements described in the present document have been developed mainly from the recommendations collected in TR 102 840 [i.1], complement the theoretic foundation of modelling standard specifications specified in ITU-T Recommendation Z.500 [i.6], and considered the meta-object facility of OMG formal/05-07-07 [i.10]. They are specified independent of a specific modelling notation or tool. Mapping of concepts to concrete modelling notations is intentionally not treated in the present document and preserved for future standards.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

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 necessary for the application of the present document.

- [1] ISO/IEC 11404: "Information technology - General- Purpose Datatypes (GPD)".

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

- [i.1] ETSI TR 102 840: "Methods for Testing and Specifications (MTS); Model-based testing in standardisation".
- [i.2] ISO/IEC 9646-1: "Information technology - Open Systems; Interconnection - Conformance testing methodology and framework - Part 1: General concepts".
- [i.3] 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".
- [i.4] 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)".

NOTE: Also published as ITU-T Recommendation series Z.140.

- [i.5] 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)".

NOTE: Also published as ITU-T Recommendation series Z.140.

- [i.6] ITU-T Recommendation Z.500: "Framework on formal methods in conformance testing".

- [i.7] Y. Gurevich, "Evolving Algebras 1993: Lipari Guide", Specification and Validation Methods, Oxford University Press, 1995.
- [i.8] Object Management Group: "Unified Modeling Language (UML), Infrastructure".
- [i.9] Object Management Group: "Unified Modeling Language (UML), Superstructure".
- [i.10] Object Management Group formal/05-07-07: "UML Testing Profile", Version 1.0, 2005.
- [i.11] ITU-T Recommendation Z.120: "Message Sequence Chart (MSC)".
- [i.12] "Extended Finite State Machines".
- NOTE: Available at http://en.wikipedia.org/wiki/Extended_finite_state_machine, retrieved 2011-01-25.

3 Definitions and abbreviations

3.1 Definitions

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

abstract test case: See ISO/IEC 9646-1 [i.2].

NOTE: A complete and independent specification of the actions required to achieve a specific test purpose. An abstract test case may be represented as a set of informal instructions or a formal specification like a TTCN-3 test case.

abstract test suite: See ISO/IEC 9646-1 [i.2].

NOTE: A test suite composed of abstract test cases.

action: atomic activity of the system triggered or observed via the system interface, consisting of an action name and a set of data parameters

NOTE: Actions are partitioned into *input* and *output* actions.

(functional) behaviour: functional behaviour of a system as specified by a set of requirements in a specification and given as a set of action sequences, where each sequence represents a legal scenario, and every sequence not in this set represents an illegal scenario

deterministic behaviour: behaviour of a system in which for each input action sequence there exist no more than one possible output action sequence

input action: action stimulated by the environment representing a message, operation, or other kind of communication means

NOTE: An input action may carry parameters.

model-based testing: umbrella of approaches that generate tests from models

modelling notation: formal language used for the specification of models

non-deterministic behaviour: behaviour of a system where for one input action sequence more than one possible output action sequences exist

offline test generation: test generation from a model ahead of test execution time

online test generation: dynamic test generation from a model during test execution

output action: action issued by the system or SUT on the environment as a reaction on input actions, or spontaneously

NOTE: An output action may carry parameters.

requirement: documented need of what a system should be or perform

(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, in terms of the system interface

NOTE: Depending on the purpose, a system model may only capture aspects of real system behaviour, as determined by the abstraction level chosen by the system interface.

system interface: model element that defines the input and output actions of the system on the level of abstraction selected for the given modelling and testing problem

(system) state: modality in which the SUT accepts certain input actions and/or issues certain output actions

(system state) transition: transition in the SUT from one system state to the next, usually associated with an input or output action which causes the transition

system under test (SUT): See ISO/IEC 9646-1 [i.2].

NOTE: The real open system in which the implementation under test resides.

test generation: automatic derivation of abstract test cases in one or more different formats from a model based on user defined test selection criteria

test purpose: See ISO/IEC 9646-1 [i.2].

NOTE: A prose description of a well defined objective of testing.

test selection: process or the result of choosing a subset of tests during test generation from a larger or infinite set of tests which can be derived from a model

test selection criterion: property that is satisfied by a set of test cases generated from a model

3.2 Abbreviations

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

ASM	Abstract State Machine
EFSM	Extended Finite State Machine
MBT	Model-Based Testing
MSC	Message Sequence Chart
SUT	System Under Test
TTCN-3	Testing and Test Control Notation
UML	Unified Modeling Language

4 Model-based test development

In model-based test development, an engineer starts from a set of requirements of a system to be tested, usually given in a specification written in natural language. The engineer authors a model using a modelling notation which fulfils the requirements stated in the present document. The model encodes these requirements and describes the aspects of the functional behaviour as well as the interfaces via which these are to be tested.

The model is then instrumented for the purpose of test generation by adding or selecting test selection criteria, i.e. coverage goals or test purposes specifying what is to be covered, and heuristics specifying how these goals are to be covered. Test selection is necessary since from every non-trivial model, an infinite or huge amount of tests can be derived. A model-based testing tool then automatically generates an abstract test suite that complies with these criteria. This resulting abstract test suite may need to be adapted [i.4], [i.5] to enable test execution against the SUT.

NOTE: The specification of test selection criteria is beyond the scope of the present document.

In model-based testing, abstract tests may be generated offline in one or more different formats including informal instructions for manual test execution, graphical formats such as tables or message sequence charts [i.11], scripting or programming languages such as Java, C#, or TTCN-3 [i.3]. In a second step, offline tests may then be compiled together with the test adaptation into executable tests and executed against the system under test. Abstract tests may also be generated online, i.e. individual test steps are immediately executed via the test adaptation against the system under test and observations resulting from the execution of a test step are directly fed back into the test generation engine. During test execution the test system finally issues a pass, fail, or inconclusive test verdict based on the outcome of each generated test.

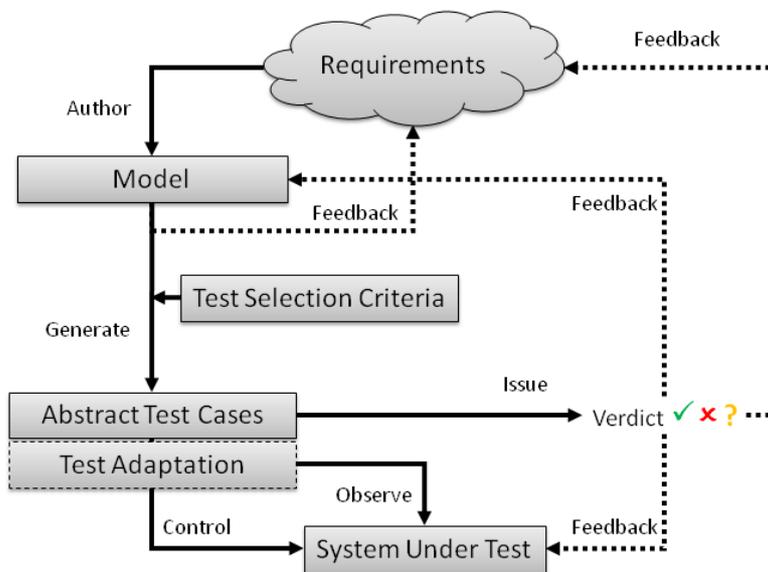


Figure 1: Model-Based Test Development

Model-based test development delivers feedback for the involved artefacts on multiple levels. First, the process of authoring a model which captures functional requirements provides feedback for the consistency of the system specification, potentially before any test is executed, or any part of the system is implemented. Second, the inspection of generated test cases and feedback from model analysis - like checking for deadlocks and safety conditions - can reveal issues in the system specification or the model. Third, when the tests are finally executed, issues in the SUT, in the system specification, or in the model can be discovered.

In the remainder of the present document, requirements for modelling notations which enables the described model-based test development are captured on a conceptual level. For an informative discussion of commonly used modelling notation styles fitting these requirements, see annex A.

5 General modelling notation requirements

Models are well defined language artefacts which are used in engineering processes much similar to programming languages. They can become large for complex systems, can address different viewpoints on a system, are authored in teams, evolve in iterations, are reused in different ways, require documentation, and need lifecycle management. Moreover, models often require algorithmic descriptions for some of the behavioural aspects they describe. As such, a modelling notation shall support concepts to those common for programming languages in software engineering. These concepts are described in this clause.

5.1 Modularization

An overall model may describe numerous complex aspects of a system, which can be best understood and maintained in isolation. A modelling notation should therefore support the software engineering principle of separation of concerns by providing means for modularization, allowing separating and recombining aspects of the system specification, such that they can be independently developed, understood, evolved, and composed into an overall system. Modularization should also support model reuse of individual components in different configurations or versions of one or more systems.

More specifically, the modelling notation should support the following:

- a) Provide a way to isolate aspects of the overall model in an independent artefact, like a document or set of documents.
- b) Provide a way to specify the dependencies of an isolated artefact from other model artefacts.
- c) Provide a way to layer model artefacts in a hierarchical manner.
- d) Have a well-defined semantics of the composition of isolated model artefacts.

Modularization can be achieved in a number of ways, details of which are beyond the scope of the present document. In general, modularization can be achieved in very similar ways as in programming languages by using concepts like components, modules, namespaces, and classes with well-defined interfaces. However, modularization can also be achieved by modelling-specific concepts, like model composition, model transformation, etc.

5.2 Algorithms

Nearly every non-trivial modelling problem requires the specification of algorithms which compute output from input data, compute the next system state, check for conditions, define constraints on data values, etc. Even if the modelling language is based on a diagrammatic notation, algorithmic language support is required for describing, for example, changes in the state of the SUT.

A modelling notation shall therefore provide basic means for algorithmic design and data manipulation, as described below:

- a) The notation shall be based on an unambiguous operational semantics.
- b) The notation shall support at least the basic data types boolean, integer and character string as well as the user defined data types record and array from conventional programming languages as defined in ISO/IEC 11404 [1], together with their related operations.
- c) The notation should support more advanced data types, like floating point numbers, enumerations and associative arrays.
- d) The notation should support unbounded data like arbitrary precision integers or arrays of non-fixed size.
- e) The notation shall support basic control constructs like variables, assignment, and conditional statements.
- f) The notation should support advanced control constructs like loops.
- g) The notation should support procedural abstraction, i.e. to allow the definition of procedures, methods, or functions which abstract the realization of a particular algorithm.

While providing those features can be achieved in numerous ways, it is considered to be beneficial that these features are based on established notations instead of being defined from scratch for the particular modelling notation.

5.3 Documentation

While a model provides a precise formalized description of the system, it needs to be accompanied by natural language documentation to make it comprehensible for reviewers and other third parties. Therefore the modelling notation shall support means to augment the formal definition of model elements with comments and more formal documentation, similar to that of many programming languages.

More specifically:

- a) A modelling notation shall support ways to attach informal comments to relevant model element definitions.
- b) A modelling notation should support ways to attach formal documentation to relevant model element definitions.

Note that the difference between (a) and (b) above is the degree of formalization: an informal comment may appear in the original model artefact but its format is arbitrary and it cannot be processed by a tool chain, whereas formal documentation has a well-defined format and can be processed by a tool chain; for example, can be validated for consistency, or used as input for model report generators.

6 Modelling the system interface

In order to facilitate testing, a modelling notation for model-based testing shall provide ways to precisely define the interface available for communicating with the system. The system interface defines input and output actions which allow to control and observe the system. The test suite generated from the model uses the input actions to stimulate functionality on the system, and observes the output actions which represent the system's responses, validating whether they conform to the modelled behaviour. Figure 2 illustrates the relations.

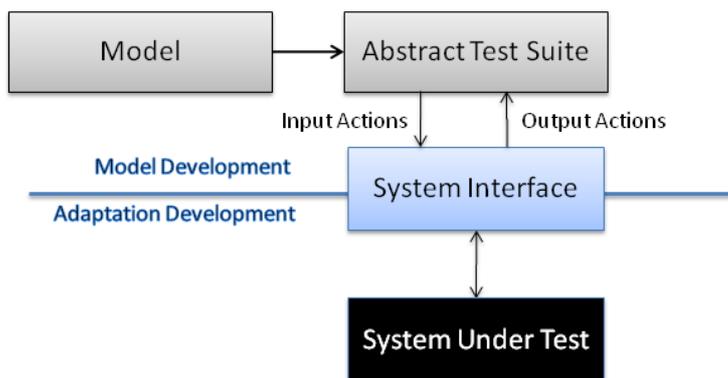


Figure 2: The role of the system interface

The system interface specified in modelling may only be a part and is often an abstraction of the real system interface. Abstraction results from focusing on testing specific aspects of a system while hiding others, or it may result from simplifying details, like for example excluding low-level data representation of messages or operations, modelling only a subset of all available interfaces offered by the system, or modelling only a subset of message or operation parameters. For the model and the generated test suite the actual system is a black box; model and tests are defined in terms of the chosen system interface abstraction.

6.1 Actions

An action is an atomic activity of the system, triggered or observed via the system interface, consisting of an action name, a set of parameters, and directionality (input or output). Actions are used to represent both, asynchronous as well as synchronous communication, such as messages, events, invocation of or return from operations, or other kinds of communication means.

A modelling notation shall support actions as described below:

- a) The modelling notation shall support the declaration of actions together with a name, whether they are input or output actions, and with parameter types.
- b) Parameter types shall include at least the basic types defined in clause 5.2 b), and should include advanced types defined in clause 5.2 c).

6.2 Operations

An operation is a set of actions where one action represents an input initiating either an operation (e.g. a "call" action) or a call-back operation (e.g. a "get call" action), and the other actions represent outputs for different ways of termination of the operation (e.g. a "return" or "exception" action) or call-back operation. In a domain where synchronous communication is dominant, it is beneficial if this concept is directly supported. For such domains, a modelling notation should support declaring operations as shorthand for declaring the basic constituting actions.

6.3 Ports

A port represents a collection of actions (or operations) which together constitute a particular logical interface of the overall system. For example, a port may represent one of several access points or services provided by the system. Clustering actions in ports aids the structural clarity of the model. Ports may also exist in multiple instances, for example, to provide a similar service to different clients on request.

- a) A modelling notation should support ports or a similar concept (access point, endpoint of a communication channel, interface, contract etc.) as a way of grouping actions and operations.
- b) A modelling notation should support multiple instances of ports (or a similar concept). Alternatively to multiple instances, a modelling notation may use dedicated parameters of input and output actions or operations of a port to distinguish the instance of the port on which the actions or operation is operating.

6.4 Configurations

Often more complex systems are compromised by a configuration of entities which concurrently interact with each other using some form of communication channels. In some cases the topology of system components may be dynamically evolving, in other cases it may be statically defined for the lifetime of a system.

- a) The modelling notation should support the specification of individual model components with concurrent activity and communication channels between them as well as the instantiation of static component configurations. Ideally, the specification of model component interfaces follows the same notion as that of the system interface between the SUT and the test suite derived from a model - i.e. it is given in terms of input and output actions or operations, clustered in one or more ports.

7 Modelling the system behaviour

The specification of the functional behaviour of a system is the core modelling activity in model-based testing. Functional behaviour can be modelled in a variety of ways, using, for example, rule-oriented or process-oriented textual notations, or diagrammatic notations like state machines, state charts, sequence charts, and flow charts. The present document does not prescribe a particular notational style, but rather captures the requirements for behavioural modelling on a conceptual level.

7.1 System state

Modelling the abstract state of the SUT is a central aspect of behavioural modelling, as it identifies the situations in which certain actions are allowed or not. For the modelling of such states a notation the following requirements apply:

- a) A modelling notation shall allow to model the system state by allowing at least one of the following :
 - The definition of a set of state variables where the current values assigned to these variables constitute the state of a system.
 - The definition of a set of constraints defining the state of a system logically.
 - The definition of a program where the current program counter and program stack constitute the state of the system.
 - The definition of a diagram representing a state machine such as a UML state machine diagram [i.8], [i.9], where every state in the diagram constitutes a state or part of the state of the system.
 - The definition of a diagram representing a flow chart such as a UML activity diagram [i.8], [i.9], where every edge between two action or activity nodes constitutes a state of the system.
 - The definition of a diagram representing a sequence chart such as a message sequence chart [i.11] or a UML interaction diagram [i.8], [i.9], where a given point on the life lines of all agents or instances related to the system represent a state of a system.

- A combination of two or more of the approaches described above; in particular, combination of the first approach, describing state using state variables, with any of the other approaches. In a combination, the state of the system is represented by the product of the states of the combined approaches.
- b) In order to deal with realistic systems, a modelling notation should allow to model an unbounded number of system states. This is usually achieved by allowing state variables to range over domains which are not bounded at model design time.

NOTE: Bounding of ranges at test selection time is not excluded by this requirement.

- c) A modelling notation should allow to associate informal requirement references with a system state by allowing at least one of the following:
- Defining a predicate over the state variables of the rule system [i.7] and associate it with a special identifier, such that when the predicate is true, the informal requirement associated with the identifier is met.
 - Supporting a special instruction in a program which allows to associate a informal requirement reference with a model state.
 - Annotating a set of constraints with a special identifier.
 - Annotating a state in a state machine such as a UML statechart diagram [i.8], [i.9] with a special identifier.
 - Annotating an edge in a flowchart such as a UML activity diagram [i.8], [i.9] with a special identifier.
- d) A modelling notation shall allow to identify the initial state and should allow to identify final system states.

7.2 System state transitions

To assess that a given functionality is implemented according to a system specification, the SUT needs to be moved through a sequence of state transitions. System state transitions are triggered by providing input actions and may require the observation of one or more output actions. State transitions are often also closely related to informal requirements stated in system specification. Finally, time plays often a central role when handling communication with other systems. For the modelling of state transitions a notation the following requirements apply:

- a) A modelling notation shall allow to define a transition between two system states by allowing at least one of the following:
- The definition of operational state transition rule [i.7], consisting of an enabling condition (a predicate over the state variables) and an algorithmic update of the state variables.
 - The definition of a declarative state transition rule [i.7], consisting of a pre-condition identifying the source state(s), and a post-condition identifying the target state(s), where both conditions are predicates over the state variables.
 - Receiving or sending a message from a port, or assigning new values to state variables in a program.
 - Drawing an arc between two states in a state machine such as a UML statechart diagram [i.8], [i.9].
 - Drawing an action or activity node in a flowchart such as a UML activity diagram [i.8], [i.9].
 - Drawing an arrow between two life lines in a sequence chart such as a UML interaction diagram [i.8], [i.9] or message sequence chart [i.11].
 - Combining two or more of the approaches above.
- b) A modelling notation shall allow to associate with a transition one input and one or more output actions or (if supported) operations including parameters by allowing at least one of the following:
- Associating the action or operation name and parameters with a state transition rule [i.7], and relating the parameters with the enabling condition, pre-condition or post-condition of the rule.
 - Receiving or sending a particular action or operation with parameters from or to a port in a program.

- Associating an arc in sequence chart such a UML interaction diagram [i.8], [i.9] or message sequence chart [i.11] with an action or operation name and parameters.
 - Associating an action or activity node in a flowchart such a UML activity diagram [i.8], [i.9] with an action or operation name and parameters.
 - Combining one or more of the approaches above.
- c) A modelling notation should be allow to associate informal requirement references with state transitions by allowing at least one of the following:
- Associating a requirement reference with a rule [i.7], such that the reference will be added to each state transition created from the rule.
 - Supporting a special instruction in a program which allows to associate a reference with a state transition.
 - Annotating an arrow in a sequence chart such as a UML interaction diagram [i.8], [i.9] or a message sequence chart [i.11] with a special identifier.
 - Annotating an action or activity node in a flowchart such as a UML activity diagram [i.8], [i.9] with a special identifier.
- d) A modelling notation should support a notion of time and be able to associate timing constraints with state transitions by allowing at least one of the following:
- The definition of an admissible delay of the transition for the firing of a rule [i.7].
 - Supporting the concept of timers.
 - Annotating an arc in a state chart such a UML statechart diagram [i.8], [i.9] or life line in a sequence chart such as a UML interaction diagram [i.8], [i.9] or message sequence chart [i.11] with an admissible delay.

7.3 Non-determinism

Non-determinism is a situation where in a given state the SUT can produce for one sequence of input actions or state transitions more than one possible sequence of output actions. The concept of non-determinism is particularly important when modelling communicating systems, as those systems may be subject to environmental influences which cannot be predicted or controlled via the interfaces available for testing. Non-determinism may also result from model abstraction, where the modeller chooses not to model a specific behavioural aspect of the system.

A modelling notation should be enable the capability to model the possibility taking one of multiple system state transitions from a given state by allowing at least one of the following:

- The definition of multiple state transition rules [i.7] which are applicable to the same source state.
- Supporting a non-deterministic choice statement which can be resolved at test execution time by querying the user for the state transition to be selected.
- Supporting multiple threads of control in a model which can send and receive from a port.
- Drawing of multiple arcs with the same trigger and different output actions from the same system state in a state chart such as a UML statechart diagram [i.8], [i.9].

Annex A (informative): Examples of modelling notation styles

This annex contains a short overview of some of the most common styles of modelling notations which satisfy the requirements specified in the present document.

A.1 Rule-Based Notation

Rule-based notations are textual modelling notations where state transition rules describe the behaviour of the system. They are also referred to as extended finite state machines (EFSM) [i.12] or abstract state machines (ASM) [i.7].

In a rule-based notation, the system's state is described by a set of state variables. A set of state transition rules is then provided in an operational style. Those transition rules consist of:

- An action name with its parameters, which describes how the transition created by the rule is labelled when the rule fires.
- An enabling condition, which is a predicate over the state variables and action parameters, and describes in which state and with which action values the rule fires.
- A state update, which describes how the state variables are changed by the rule if it fires.
- Other information like references to informal requirements or timing constraints.

Rule-based notations usually have one distinguished initial state, which is given by an assignment to the state variables.

Non-determinism in rule-based notations can be easily expressed by enabling rules with different output actions in given states.

Rule-based modelling notations satisfy the requirements in the present document provided the underlying algorithmic support for data domains as is used in state variable and action parameter modelling is sufficiently supported.

Extended finite state machine are a variation of rule-based notations where the number of states and transitions is bounded. This is not a contradiction with the present document, as long as providing these bounds is methodologically part of slicing for test selection.

A.2 State Chart Notation

State charts are a diagrammatic notation which exists in many variations in system modelling; they are, for example, part of UML [i.8], [i.9]. State charts combine aspects of rule-based notations with graphical structure.

In general, a state chart is a diagram which contains nodes for states and directed arcs for state transitions. A state chart may be associated with a set of state variables. The arcs of the state chart usually contain the following information:

- An action name with its parameters, which describes how the transition created by the arc is labelled when it is taken.
- An enabling condition, which is a predicate over the state variables and action parameters, and describes in which state the arc can be taken.
- A state update, which describes how the state variables are changed if the arc is taken.
- Other information like references to informal requirements or timing constraints.

In addition to these basic elements, statecharts also support hierarchical grouping of states, as well as parallel composition of states. There are more constructs in statecharts which go beyond the scope of the present document.

State chart based modelling notations satisfy the requirements in the present document provided the underlying algorithmic support for data domains as is used in state variable and action parameter modelling is sufficiently supported.

A.3 Process-oriented notation

In process-oriented modelling, a system of components is specified by describing the activity of each component as an independent sequential process (or thread). The process is usually described using an imperative modelling or based on a programming language. Each process has its independent data state, comprised by a set of state variables. During its lifetime, the process actively listens to inputs from its environment and produces outputs, usually by using the concept of port or communication channels.

Timing constraints are described by programmatic delays and timeouts.

Process-oriented modelling notations satisfy the requirements in the present document provided the underlying algorithmic support is sufficient.

History

Document history		
V1.1.1	May 2011	Membership Approval Procedure MV 20110724: 2011-05-25 to 2011-07-25