Software Quality Journal
https://doi.org/10.1007/s11219-018-9423-9

Test descriptions with ETSI TDL

Philip Makedonski[1] · Gusztáv Adamis[2] · Martti Käärik[3] · Finn Kristoffersen[4] · Michele Carignani[5] · Andreas Ulrich[6] · Jens Grabowski[1]

Published online: 17 October 2018

Abstract
To address the need for abstract, high-level test descriptions that can be shared among different stakeholders, the European Telecommunications Standards Institute (ETSI) commissioned the design of the Test Description Language (TDL). TDL is designed as a domain-specific language for testing, consisting of a standardised abstract syntax (meta-model) and concrete syntaxes for textual specification, graphical design, and model exchange between tools. Its main purpose is to support a test methodology that is followed in the standardisation work for software-intense systems at ETSI and is applicable in industrial projects as well. TDL enables the formal specification of both test objectives derived from system requirements and test descriptions refining the test objectives. The latter serve as blueprint for the implementation of executable tests. A standardised mapping of TDL specifications to test scripts in the standardised test execution language TTCN-3 widens the reach of TDL to ensure compatibility and consistency of generated executable tests. An open-source toolset has been developed as a common platform to accelerate the adoption of TDL and lower the barrier to entry for users and tool vendors. Reports from pilot applications within three ETSI standardisation groups demonstrate the practicality of the chosen approach.

Keywords Model-based testing · Test description language · Domain-specific modelling · Test methodology · Testing in standardisation

1 Introduction

The increase of software and system complexity due to the integration of more and diverse sub-systems into a system and system-of-systems presents new testing challenges. Standardisation and certification requirements in domains such as telecommunication, automotive, aerospace, and healthcare contribute further testing challenges for systems that need to operate under stringent safety requirements over a long period of time. Consequently, there is a need for supporting test methodologies, processes, languages, and tools that support standardisation work and also integrate well with existing agile development practices used in industry.

✉ Philip Makedonski
makedonski@cs.uni-goettingen.de

Extended author information available on the last page of the article.

The European Telecommunications Standards Institute (ETSI)[1] has a long experience with the development of test specifications for standardised systems. To facilitate their efficient development and address the evolving challenges, the ETSI Technical Committee "Methods for Testing and Specification" (TC MTS) has been active in integrating upcoming trends such as Model-Based Testing (MBT) and Domain-Specific Language (DSL) design into standardisation processes. The existing ETSI test development process ETSI TR 102 840 (2011) follows a stepwise approach based on the ISO/IEC 9646-1 (1994) norm. Each step on the way from base standard to executable test cases results in intermediate artefacts at different levels of abstraction, which are intended for particular stakeholders such as standardisation experts, technology experts, and test engineers.

With a strong emphasis on test automation, ETSI TC MTS has been leading the work on the development, maintenance, and evolution of the Testing and Test Control Notation version 3 (TTCN-3), published in ETSI ES 201 873-1 (2016), over the past 15 years. While TTCN-3 has been established as the language of choice for the implementation of test cases at ETSI, there was a distinctive lack of well-established notations at higher levels of abstraction for the provision of test objectives and test descriptions. A multitude of dialects and customised notations has been proliferated at different standardisation working groups. Without a suitable, standardised language for this purpose, the development of test descriptions leads to a significant overhead and frequent inconsistencies that need to be checked and fixed manually. The consequences are particularly severe for test descriptions related to technologies and standards that continue to evolve over decades.

The ETSI Test Description Language (TDL), published in ETSI ES 203 119-1 (2018), seeks to bridge the methodological gap between declarative test objectives and executable test cases by providing a formalised model-based solution for the specification of test descriptions. At its core there is a common meta-model (abstract syntax) with well-defined semantics which can be represented by means of different concrete notations (concrete syntaxes). A TDL test description created in one notation can be reviewed and approved in other notations, customised to suit the preferred notational conventions of different stakeholders. TDL can also serve as an exchange and visualisation platform for generated tests, contributing to the ongoing activities within ETSI TC MTS to establish MBT technologies within standardisation at ETSI (ETSI TR 102 840 2011; ETSI EG 203 130 2013; ETSI ES 202 951 2011). The principles which allow the adaptation of TDL according to the users' needs are based on a formal basis, which in turn paves the way to tool support and the introduction of the approach to different application domains.

The design of TDL took place at ETSI between 2013 and 2017. It has been recently complemented with the development of an open-source toolset for TDL under the TDL Open-Source Project (TOP) which is intended to serve as a proof of concept for the designed language features and, more importantly, as a common platform to accelerate the adoption of TDL and lower the barrier to entry for both, users and tool vendors.

This article sets forth previous work on TDL by Ulrich et al. (2014) and Makedonski et al. (2016). It includes new material related to the latest language design enhancements and a first discussion on the mapping of TDL to TTCN-3. To demonstrate the applicability of TDL, results from pilot application cases in different ETSI standardisation areas are summarised. The article is structured as follows. Next, Section 2 discusses the use of a model-based test methodology in standardisation. Section 3 showcases the different parts of the TDL standard and introduces the language. The mapping of TDL to TTCN-3 is

[1]http://www.etsi.org

discussed in Section 4. Section 5 continues with a technical overview of the open-source implementation of TDL. Afterwards Section 6 summarises first results from the application cases in ETSI standardisation groups. Related work on test specification languages is discussed in Section 7. Finally, Section 8 provides a summary and an outlook on the future of TDL.

2 Model-based test methodology in standardisation

The section introduces a test methodology that supports test modelling and test generation in the context of standardisation work. It is also applicable within many software development projects in industry.

2.1 Test modelling

The central concept in MBT is the notion of a Test Model, from which executable test cases are derived. The focus of TDL is mainly on functional conformance and interoperability testing of applications in standardisation and industry. (Extensions are being discussed towards testing of non-functional properties as well.) For this purpose, the TDL Test Model comprises a behavioural model of *interactions* (inputs and outputs) exchanged between the System Under Test (SUT) and its environment and described from the perspective of the environment, i.e. the tester. That is, an output of the tester is a test stimulus consumed by the SUT as an input; conversely the SUT performs an output, i.e. the test response, which shall match an expected input to the tester (for a passing test run). This behavioural model is called the Environment Model in Fig. 1. It is a different MBT approach than the one that takes a model of the SUT itself as the base to derive test cases.

There are multiple reasons why an environment model is the chosen principle in TDL. First, in a top-down test methodology applied in standardisation (see Fig. 2) the starting point is a set of system requirements that express the expected system behaviour from a user's point of view (what shall the system do?). A system model is already the result of the system design phase trying to address the requirements (how shall the system do it?). Therefore, a test model based on an environment model can be considered as a direct refinement of the system requirements that does not involve any system design activities, which is

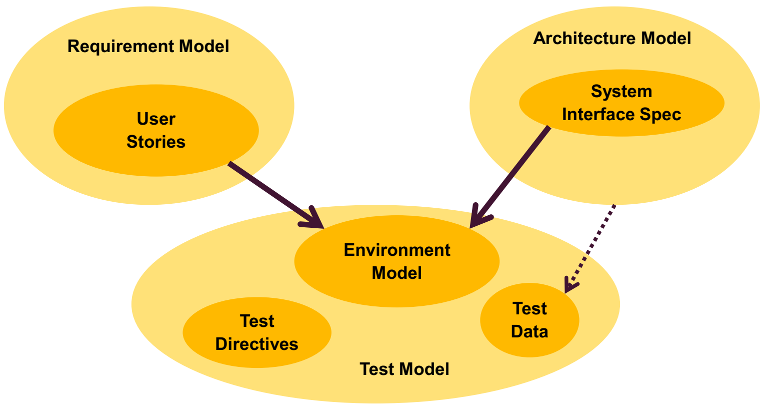


Fig. 1 Test model, its ingredients and dependencies

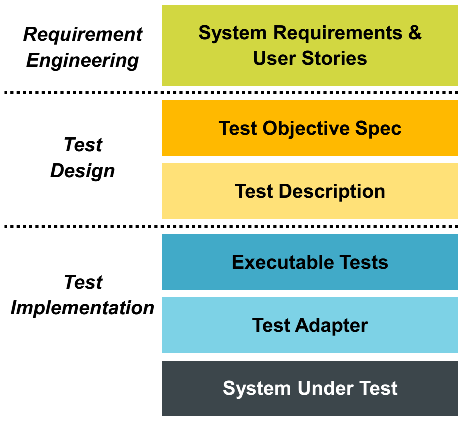


Fig. 2 Test methodology in standardisation

left to the system vendors, except for the specification of the system interfaces. Moreover, testing complex and potentially distributed systems requires efforts into a proper test architecture design, which an environment model supports well. Last but not least, there is also a socio-technical aspect. System test engineers are trained to describe tests from a user's perspective. An environment model is a natural fit for this way of thinking.

Further constituents and relationships of the Test Model are illustrated in Fig. 1. It comprises all elements needed to generate executable tests that are required to validate the correctness of the SUT against the specified system requirements and specific user stories. Such tests are *black-box tests* that perform interactions at the system boundary to conclude about correct or incorrect SUT behaviour. After sending a test stimulus to the SUT, these tests verify the SUT responses alone to conclude about a pass or fail verdict. Supporting this principle, the Test Model takes over only the System Interface Specification from the Architecture Model as the starting point for creating a behavioural model of the environment of the system that details the user requirements captured in the Requirement Model. The high-level architectural design model and system requirement model are made available through Base Standard Specifications which are the output of the standardisation process of a certain technology or are produced within the system development life cycle process followed in most industrial projects. The Environment Model within the Test Model is a behavioural specification that focuses on the description of the expected flow of *interactions* at the system boundary that a system has to obey. Any deviation from this specified expected behaviour in a test run results in a fail verdict. If the specified behaviour is executed, the test passes.

Interactions are always directed and possess a sender and a receiver (in a 1-to-1 communication relation). They are distinguished between test stimuli (e.g. messages sent to the SUT, invoked SUT functions, or SUT input signals that are set to particular values) and test responses (e.g. messages received from the SUT, the return values from SUT function invocations, or SUT output signals that are read by the tester). The data exchanged in these interactions is captured as Test Data extracted from the available Architecture Model or, if available already, directly from the requirements. Different from data used to define a test stimulus, data of a test response may contain wildcards to denote any value, a range of acceptable values or omission of a value.

Because the environment model and the data used in the interactions are abstract in the sense that they cannot be directly used for test execution, behaviour and data need to be refined to produce concrete, executable tests. This refinement step is referred to as *test* *derivation*. It can be (partially) automated or performed entirely manually. It produces a number of executable tests according to a test selection strategy and other criteria described as Test Directives.

2.2 Test methodology

The whole test methodology applied in standardisation is illustrated in Fig. 2. It starts with an available set of requirements. Depending on the granularity of the requirements, a Test Objective Specification consisting of a set of *test objectives* is created. It provides detailed information of how a requirement has to be tested. This additional level of detail is achieved by adding actors, their actions, an abstract interaction scenario and a definition of the major test data to be used.

There are different languages already available to express test objectives such as Gherkin for expressing *Given-When-Then* clauses in the context of Behaviour Driven Development (BDD) (Solis and Wang 2011) or the Test Purpose Notation (TPLan) (ETSI ES 202 553 2009). In addition, TDL offers the TDL Structured Test Objective Specification (TDL-TO) extension as a standardised language to express test objectives. While simple test objective languages allow for a direct translation into Executable Tests, the creation of a Test Description is a desirable intermediate step for more complex test scenarios and systems. The Test Description then refines a test objective as a test which in turn is defined in a declarative manner (what shall be tested?), in opposition to the typical imperative form of test implementations (how shall it be tested?). Moreover, test descriptions are provided at an abstract level which is the same as the abstraction used in the Test Model from which they are derived. In addition, they are still independent from implementation details such as the concrete test data being exchanged in a test or details about the test behaviour when the test fails. The dedicated test description language ETSI TDL discussed in Section 3 is designed to express such test descriptions.

Having a test description defined manually or generated automatically from other system development artefacts, a test generator can be used for the refinement towards Executable Tests in a given test scripting language such as TTCN-3 or a general-purpose programming language such as Java, C# or Python. Section 4 provides details about the defined translation from TDL to TTCN-3. Though in most practical cases the generated concrete tests must be augmented further to become executable. Typically the generated tests are complemented by a manually created Test Adapter to bridge the specific implementation details of a SUT interface with the abstract stimulus/response pairs used in test descriptions. The efforts needed to produce a test adapter can be substantial.

The standardisation process stops at the creation of Executable Tests in TTCN-3 or even before that. The least commonly agreed output for testing a standardised technology is a Test Objective Specification, keeping the laborious detailing work of producing concrete tests to the technology vendors or accreditation institutions. At that point it is expected that ETSI TDL creates an impact because of its ability to link directly between Requirement Models and Test Models via a standardised model-based engineering approach.

3 The TDL standard

The TDL specification evolved into a multi-part standard. This section provides a broad overview of the core principles behind the design of TDL and illustrates some of the features of the language by example.

3.1 Core design principles

The TDL is intended for the design, documentation, and representation of formal test descriptions for system testing following a scenario-based approach describing interactions with the SUT (black-box testing). In addition, test objectives that reflect requirements on the system can be specified and attached to a test description or even to a part of it. The test descriptions are then used as basis for deriving executable, concrete tests. Because TDL provides for an abstract representation for tests, the language can also be used for the representation of test sequences derived from MBT tools, system simulators, or traces from test execution runs.

TDL is built around the TDL Meta-Model (TDL-MM) (ETSI ES 203 119-1 2018) which specifies the abstract syntax of the language concepts, the relationships among them, their properties, and their intended semantics. The abstract syntax is defined in terms of the Meta-Object Facility (MOF) (OMG MOF 2014) meta-model. Constraints on the concepts and their relationships which formally define the static semantics of the language are formalised by means of the Object Constraint Language (OCL) (OMG OCL 2012). The TDL-MM is structured in packages covering different aspects of TDL. Concrete syntax notations may be mapped to the abstract syntax making the elements of the meta-model accessible to end-users by means of different representations, such as graphical, textual, tabular, tree-based, targeting different levels of abstraction, as shown in Fig. 3. Annex B of the TDL-MM contains a non-normative example of a text-based notation in order to showcase the language concepts and their use. It can serve as the basis for user-defined domain-specific test modelling language based on the TDL-MM.

The TDL Graphical Representation (TDL-GR) (ETSI ES 203 119-2 2018) provides a standardised, general-purpose concrete syntax for the graphical representation of TDL concepts and their properties and relationships. One of the key design requirements was that the graphical representation of TDL shall resemble the graphical format of the most frequently used modelling notations in order to preserve familiarity and ensure that it is easy to learn for users. Considering that Unified Modelling Language (UML) (OMG UML 2015) is a widely accepted and used language in the industry, the TDL graphical representation was aligned with UML to the extent of which the corresponding TDL concepts have (almost) direct equivalents in UML. Concepts from the TDL-MM that have no direct equivalent or have different semantics are represented differently in order to avoid confusion.

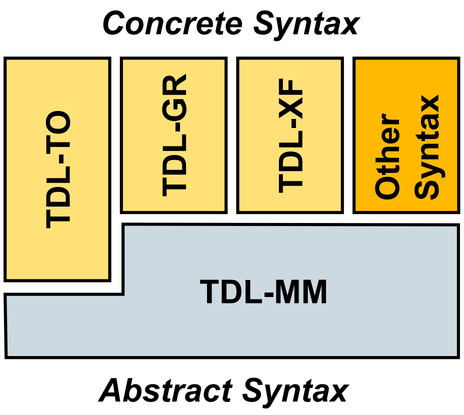


Fig. 3 TDL abstract syntax (meta-model) and concrete syntaxes

The TDL Exchange Format (TDL-XF) (ETSI ES 203 119-3 2018) serves as basis for the interoperability of tools. It describes the rules for serialisation and de-serialisation of TDL models which enable the interchange of TDL models between tools. The TDL-XF is based on the XML Metadata Inter- change (XMI) specification (OMG XMI 2014). The TDL-XF specifies production rules for deriving a TDL XMI Schema from the TDL-MM, as well as production rules to derive a TDL XMI document from the serialisation of a TDL model instance.

The TDL Structured Test Objective Specification (TDL-TO) is defined in ETSI ES 203 119-4 (2018) as an extension of TDL that introduces additional concepts to the TDL-MM, as well as a corresponding concrete textual syntax to help users in supporting a structured and formal approach to the specification of test purposes that precede the process of specifying test descriptions. The standardised TDL to TTCN-3 mapping is defined in ETSI ES 203 119-6 (2018). It states how concepts of TDL are mapped to TTCN-3 constructs such that the dynamic behaviour of the TDL specification is preserved in the resulting TTCN-3 specification. Extensions to the TDL test configuration concept is defined in ETSI ES 203 119-7 (2018). These extensions are introduced to allow for re-use of TDL test configurations and to support the management of larger test specifications in TDL.

The decomposition of TDL into a multi-part standard has the advantage that tool vendors and users can decide which parts they want to conform to. For example, tools providing customised syntax representations for a specific user group may opt to support only the TDL-MM and TDL-XF parts, rather than supporting the TDL-GR which may be unnecessary for the targeted user group.

The specification of a TDL test description requires three major parts: (1) data and data type specification, (2) test configuration specification, and (3) test behaviour specification. These parts can be grouped and organised in packages. Additionally, time-related aspects can be expressed in TDL by means of different concepts. In the following, we illustrate the use of the main concepts of TDL by means of the TDL-GR. Examples illustrating the textual representation of TDL are discussed in Section 6.

3.2 Data and data type specification

Data elements in TDL are abstract symbols that can be related to concrete data representations by means of *data element mappings*. The concrete data representations may be stored in external resources, such as TTCN-3 or XML documents, which are referenced by means of *data resource mappings*. The defined data elements can be referenced in various contexts of a test configuration or within test behaviour.

The specification of simple and structured data types and data instances is shown in Fig. 4. The graphical representations for type definitions have double borders in TDL-GR. Simple data types and instances do not have an internal structure within TDL. In this example, the definitions of data type String and the data instance Hello shown on the top of Fig. 4 represent simple data types and data instances.

To specify relevant parts of the internal structure of a data type, structured data types and instances can be created which define members and member assignments, respectively. On the bottom-right of the example shown in Fig. 4, the Message data type is illustrated. It contains the sessionId and content members, of data types Integer and String, respectively. In the Request data instance shown on the bottom-left of Fig. 4, the data instance Hello is assigned to the content member. The sessionId member is left unassigned. It can be assigned to a data instance by means of parameters when the Request data instance

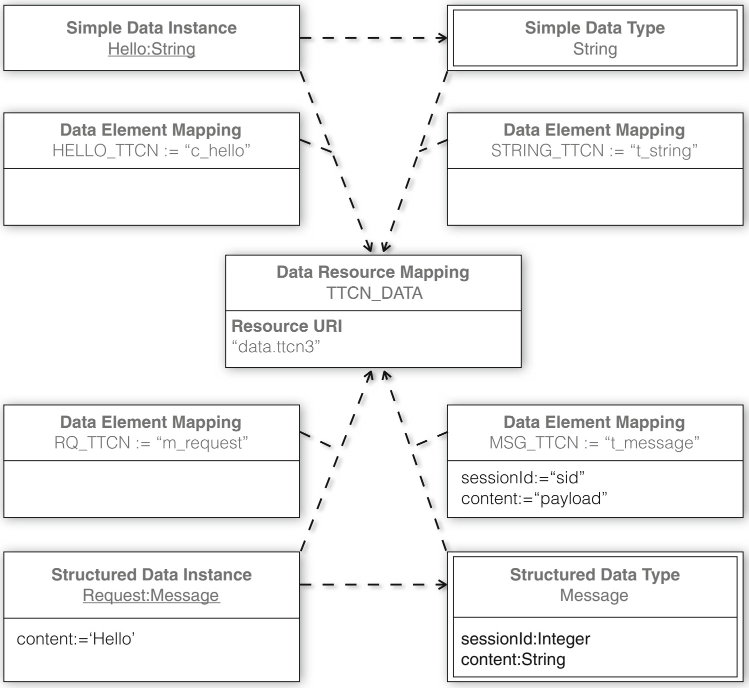


Fig. 4 Data definition and mapping specification example

is used. Parameters can also be used to override the data instances already assigned to members, e.g. replacing the Hello assigned to content by another data instance, such as Hi. In addition, user-defined functions can be provided that operate over the defined data types.

TDL only provides abstract symbols for data. The data type and data instance definitions need to be mapped to concrete types and values of an underlying executable language, such as TTCN-3 or another programming language.

The data mappings for the data types and instances discussed above are shown in the middle of Fig. 4. In this example, a file data.ttcn3 containing some concrete data specifications in TTCN-3 is made accessible in TDL by means of a data resource mapping. The data element mappings then specify how the abstract data types and data instances defined in TDL are related to the corresponding concrete data specifications. The Hello data instance is mapped to a constant c_hello defined in data.ttcn3 and the String data type is mapped to the concrete t_string data type. Similarly, the structured data type Message and the structured data instance Request are mapped to corresponding concrete data elements in data.ttcn3, where for the structured data type Message the mappings for the members are also specified.

While this mapping appears as an additional overhead on a first glance, it bears the possibility to abstract away unneeded details of structured data leading to concise test specifications that enable users to focus only on the relevant structural elements. Furthermore, it also allows the re-use of existing test assets without the need to consider all the details that are required for the operationalisation of a test.

3.3 Test configuration specification

Test configurations in TDL are composed of at least two component instances, one in the role of Tester and one in the role of SUT, with at least one connection between them. An example for a test configuration defaultTC is shown in Fig. 5. A component is defined via its component type (e.g. defaultCT) which contains at least one gate (e.g. g) of a given gate type (e.g. defaultGT). Gates enable the communication between components, where the gate type specifies the data that can be used in interactions between components over gates of the gate type in question. A component type may also define additional variables or timers that are local to instances of that component type.

3.4 Test behaviour specification

The specification of behaviour includes so-called *atomic* and *combined behaviour* concepts. Atomic behaviours include interactions for exchanging data between component instances, local or global actions, references to other test descriptions, as well as explicit verdict assignments.

An interaction in TDL represents an occurrence of any of the commonly used types of communications, such as exchanging messages asynchronously, calling blocking procedures, and accessing shared variables. The concrete realisation of an interaction must be addressed in the underlying test execution language.

Combined behaviours, which include parallel, conditional, alternative, repeated (looping), interrupt, periodic, and default behaviour, are used to group atomic behaviours. Some of them such as parallel, conditional, and repeated behaviours are also known from UML, while others are introduced to support the needs of test specification.

The specification of behaviour is exemplified in Fig. 6. The example features a simple interaction sequence between the previously defined component instances. The SS in the role of Tester sends a Request message, setting the unassigned sessionId to 1. The UE in the role of SUT may respond to the SS in different ways which are captured by means of an

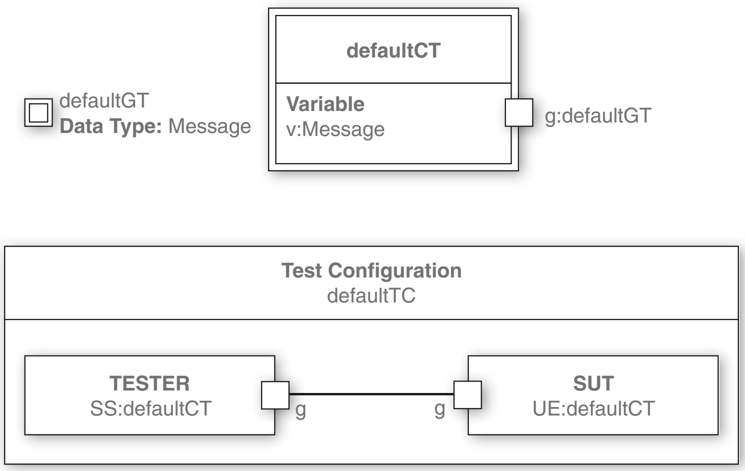


Fig. 5 Gate type, component type, and test configuration examples

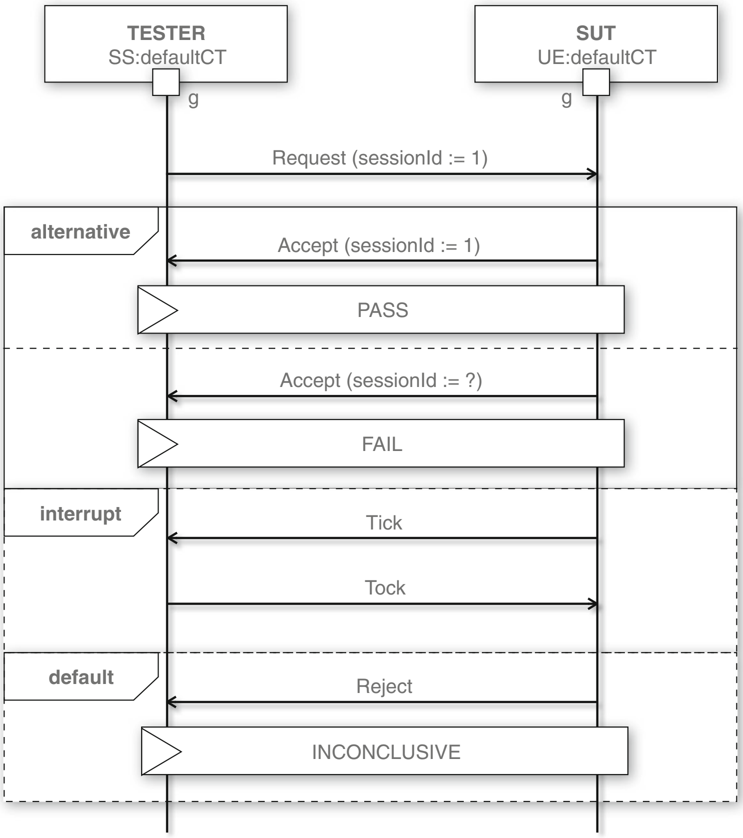


Fig. 6 Behaviour specification example

alternative behaviour. It may respond by sending an Accept message with the same sessionId (1) which results in a verdict PASS, or it may send an Accept message with a different sessionId (?) which results in a verdict FAIL. At any point, there may be an exchange of synchronisation or timing messages (Tick and Tock) between the UE and the SS. They are captured by an interrupt behaviour attached to the alternative behaviour. Every time the interrupt behaviour is triggered by the corresponding interaction, it returns to the point at which it was triggered after executing the behaviours contained within it. Finally, a default behaviour is attached to the alternative behaviour to handle special cases such as receiving a Reject message, in which case the verdict is set to INCONCLUSIVE and the execution shall continue outside the alternative behaviour.

TDL assumes an implicit test verdict assignment where the specified behaviour defines to expected behaviour which receives the PASS verdict when executed. Any deviation from the specified behaviour, it implies a FAIL verdict. In this sense, the specification of the PASS verdict in the example is not strictly necessary, but it can still be helpful for readability. The FAIL verdict highlights a special case of failing behaviour. Alternatively, the alternative behaviour specification can be omitted altogether where only the expected behaviour is specified.

3.5 Time specification

Time in TDL is global and progresses monotonically in discrete quantities. TDL offers time operations including *wait* and *quiescence*, which are used to delay the execution or ensure that no interactions occur in the gates of a component instance in the role of Tester. In addition, timer operations (*start*, *stop*, *timeout*) can be declared that operate on component instance timers. Finally, *time constraints* can be used to specify temporal relationships between behaviours.

The use of time constraints and time operations is illustrated in Fig. 7. The time labels @T_request and @T_accept are specified to indicate the time points at which the sending of Request to the UE and the receiving of Accept from the UE do occur. Then, a time constraint @T_accept < @T_request + 3 is specified to indicate that Accept shall be received from the UE no later than 3 time units after Request was sent to the UE. Further on, a wait (W) operation is specified indicating that the SS shall wait for 2.5s before sending Close to the UE. Finally, the quiescence (Q) operation is used to indicate that the SS shall not receive any further messages from the UE during the next 2.5s.

The specification of temporal properties by means of timers and timer operations is illustrated in Fig. 8. In this case, the equivalent of the wait operation from Fig. 7 is expressed by means of timer operations. After receiving a Response from the UE, the SS starts a timer for 2.5s and waits for the timer to expire by means of a time out specification, before continuing with the sending of Close to the UE.

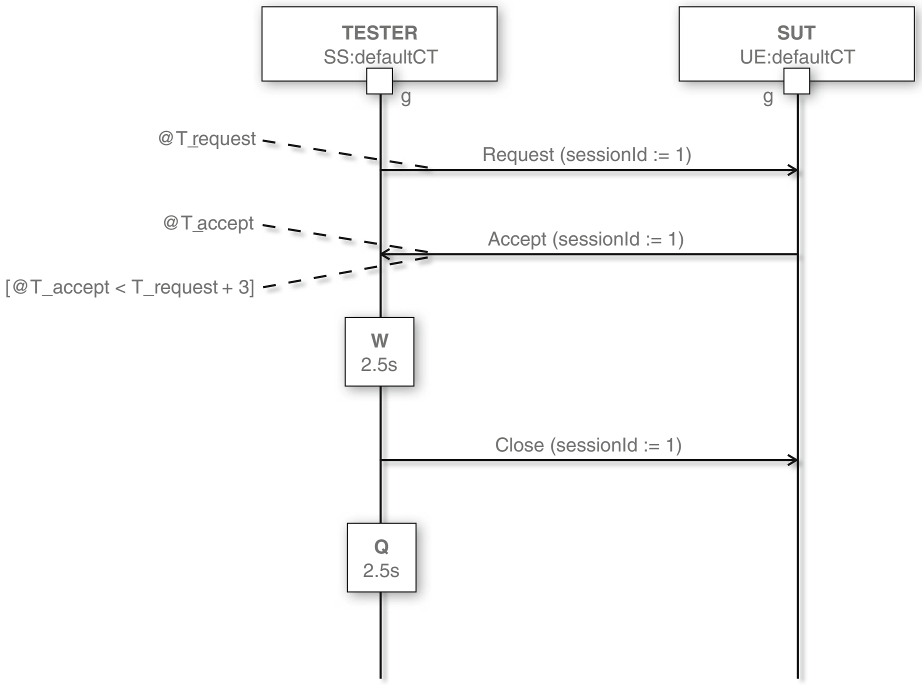


Fig. 7 Time constraints and time operations examples

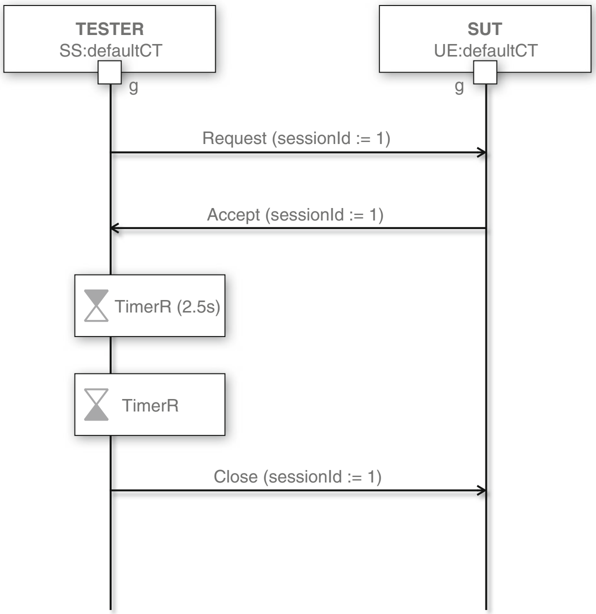


Fig. 8 Timer operations examples

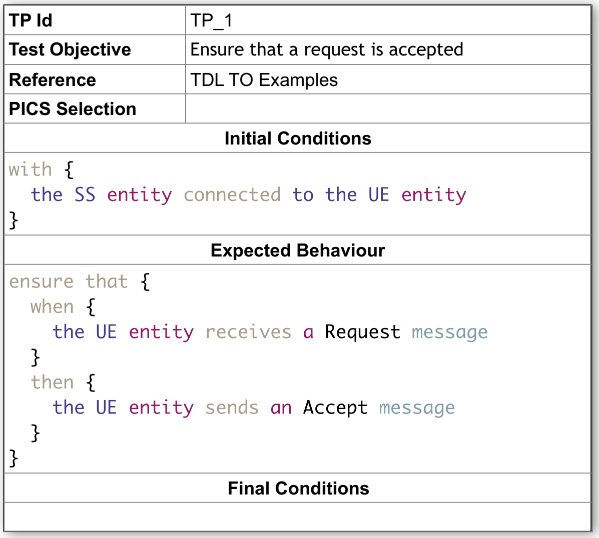
3.6 Structured test objective specification

By default, the description of a test objective in TDL can be specified only as informal text. The TDL-TO extension defines additional concepts to support the specification of structured test objectives in a formalised manner. These concepts capture additional features, including:

– Domain specification concepts such as abstract events and entities, as well as reusable event occurrence templates,
– Structured test objectives containing event occurrence sequences for initial conditions, expected behaviour, and final conditions,
– Event occurrence specifications containing entity references, event references, event occurrence arguments, and time constraints,
– Extended data specification for literal data used as event occurrence arguments that do not require the definition of data types and data instances.

The concrete syntax notation is related to the TPLan notation used within ETSI in order to align it with existing procedures and capitalise on user familiarity. It also takes up the notion of *Given–When–Then* from BDD-style development. An example is shown in Fig. 9.

While the notation for the structured test objectives as a whole is graphical in nature, its contents describing the event occurrences are in the form of structured natural language, where entity and event references are surrounded by keywords and free text in the form of comments. This adds sufficient formalisation to the specification of test objectives in order to enable tool support for the validation of test objectives and for mapping them to test descriptions, while still retaining a natural language feel. The TDL-TO also contains an



TP Id	TP_1
Test Objective	Ensure that a request is accepted
Reference	TDL TO Examples
PICS Selection	

Initial Conditions

```
with {
  the SS entity connected to the UE entity
}
```

Expected Behaviour

```
ensure that {
  when {
    the UE entity receives a Request message
  }
  then {
    the UE entity sends an Accept message
  }
}
```

Final Conditions

Fig. 9 Structured test objective specification example

informative text-based notation in its *Annex B*, which focuses on the concepts introduced as extensions to the TDL-MM.

3.7 The UML profile for TDL

The UML Profile for TDL (UP4TDL) (ETSI ES 203 119-5 2018) enables the application of TDL in UML based working environment. Profiling in UML is a way to adapt the UML meta-model with domain-specific concerns. In simplified terms, domain-specific concepts are represented in a UML profile by means of *stereotypes* which enable the extension of a UML meta-class with additional properties, relations, or constraints.

The overall architecture of UP4TDL is similar to the package structure of TDL. One major addition is related to the data element mapping which allows the binding of abstract mappable data elements to concrete data specifications stored in a data resource. For the specification of data use, an almost independent meta-model is defined as the basis for integrating data-related expressions in TDL models. These expressions are used in the specification of arguments of interactions. Additionally, time-related concepts are currently mapped in a manner that is specific to TDL. In future work, linking these concepts with the Modelling and Analysis of Real-Time Embedded Systems (MARTE) profile (OMG MARTE 2011) may help to refine the embedding of UP4TDL into the UML environment.

The UML Testing Profile (UTP) (OMG UTP 2013) provides high-level test-related concepts for use within UML-based working environments. UTP and UP4TDL share similar global choices for representing various aspects of both languages. The arrangement of test components in a test configuration may be described by a composite-like diagram, while test behaviours can be described by sequence-like diagrams. The data types and data instances may be presented in a class-like diagram. However, there are also a number of differences between UP4TDL and UTP. Since TDL and UTP evolved independently, UP4TDL follows design decisions that keep it closer to TDL. While behaviour specifications in UTP may have different kinds of representations, the behaviour-related stereotypes in UP4TDL are also intended to be directly mapped to corresponding behaviour-related concepts in TDL, such as exceptional behaviour and periodic behaviour.

4 Mapping TDL to TTCN-3

Both TDL and TTCN-3 are standardised test languages, but the way how they describe the test system architecture and its behaviour and the chosen levels of abstraction are different, with TDL being the more abstract language. Hence, there are various ways to derive TTCN-3 code from a TDL test description, which depend on the test directives and further details related to the tester deployment within a specific hardware configuration that are not available at the level of TDL. This may result in different or even incompatible code intended to implement the same test description. Without a standardised mapping of TDL to TTCN-3, there could be a proliferation of different and possibly incompatible tool- and user-specific mappings of TDL test descriptions to executable test cases which can present new challenges to users and tool vendors.

A standardised mapping between the two languages provides a consistent approach for producing executable tests from high-level test descriptions specified in TDL. It enables the generation of executable tests from TDL test descriptions in a (semi-) automatic way, and by extension of the re-use of existing test tools and frameworks for test execution. This way, test engineers can concentrate on the specification of test descriptions at a higher level of abstraction, while having a clear expectation of what the resulting test implementation will look like.

The two languages are based on several fundamentally different assumptions regarding how tests are described. This section collects the most important differences between the semantics of the two languages and summarises the work on the standardised mapping of TDL to TTCN-3.

4.1 Levels of abstraction

In practical terms, TTCN-3 is used for the specification and execution of automated tests at a low level of abstraction which is close to test implementation. While it provides the means for abstracting away some details, enabling its use at higher levels of abstraction, the strict syntax and required structural details can be considered a limiting factor in such application scenarios.

In contrast, TDL is designed for use at higher levels of abstraction, for the specification of higher level test designs. On one hand, it supports the design of highly abstract structured test objectives via the TDL-TO extension which provide little information of how the test will be eventually implemented. On the other hand, TDL supports the specification of test case designs that refine a provided test objective at a medium level of abstraction.

4.2 Mapping data

TTCN-3 has a comprehensive data-type system as well as a powerful template mechanism including extensive matching operators. TDL on the other hand relies on mappable abstract symbolic elements while still providing facilities for checking the consistency of data definitions and uses by means of data types. Parameters and functions in TDL allow to express the principal data flow through a test description, however, most of the details are left to the lower level of test implementation. Steering the generation of an implementation from a test description, data mappings are required, where upon the occurrence of data instances in TDL is substituted by the respective mapping targets. In case when no data mappings are provided in a TDL specification, basic data generation is defined to provide users with a starting point for concrete data representations.

4.3 Mapping configurations

TTCN-3 is a test system centric language that specifies tests from the test system point of view. Consequently, while it provides means to specify distinct test components, from within the core language there is only one unified SUT interface accessible via different ports. The system ports themselves can still be associated with different SUT components in the test adapters for TTCN-3, but distinct SUT components cannot be specified explicitly within TTCN-3. In contrast, TDL considers a system centric view of both, SUT and tester. It includes all tester and SUT components involved in a test. This is illustrated in Fig. 10, with the TDL view on the top and the TTCN-3 view on the bottom.

In order to map a test configuration comprising multiple SUT components, a unified SUT interface in TTCN-3 needs to be inferred. The inferred unified SUT interface in TTCN-3 shall have as many ports as there are gate instances in the corresponding SUT component instances in TDL. While test configurations in TDL are static and defined up front, the configuration initialisation in TTCN-3 needs to take care of creating the corresponding test components, as well as mapping and connecting their ports. As TDL is focused on higher level designs, and TTCN-3 is aimed more towards test implementation, there are certain restrictions regarding the possible ways of interconnecting ports in

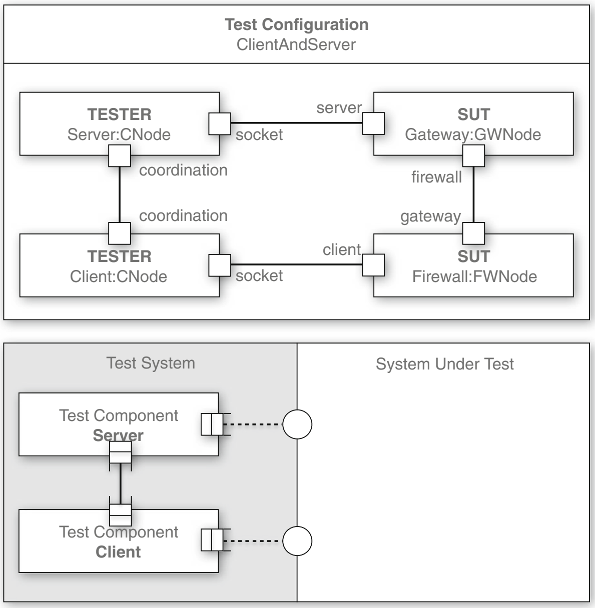


Fig. 10 TDL and TTCN-3 perspectives on the SUT

TTCN-3. As a consequence some ports may need to be cloned in order to respect these restrictions.

For example, in TTCN-3 a port can only be mapped to one system port, while there is no such a restriction in TDL. On the other hand, in TDL, it is possible to connect a gate of a tester component to a gate of another tester component and to a gate of an SUT component at the same time. In TTCN-3, however, this is not possible. A port of a test component shall be connected either to another test component (by means of the connect instruction) or to the system component (by means of the map instruction). This difference requires that each TDL gate is mapped to two TTCN-3 ports; one of them is connected to test components, while the other one is mapped to the SUT interface.

Figure 11 illustrates a possible mapping on an example. A TDL test configuration, in which a tester component UE_A is connected to two SUT components USER_A and IMS_A, is shown at the top of the figure. The equivalent architecture in TTCN-3 containing the replicated ports is shown in the lower half of Fig. 11.

The standardised mapping also provides unique solutions on treating corner cases of test configurations limitations in TTCN-3 without introducing additional restrictions on TDL. A separate issue is the deployment of test components to physical nodes of a test system. While a 1-to-1 deployment may be the obvious choice, in some cases, more varied solutions such as an *n*-to-1 or *n*-to-*m* deployment may be desirable. The mapping generates in TTCN-3 a parallel test component (PTC) for each TDL component instance. The way how these PTCs are to be deployed on physical nodes is left to the responsibility of the test engineer. If details on the deployment are already available at the time of writing the TDL specification, they can be provided as annotations which a code generation can exploit, differently to pure comments.

4.4 Mapping behaviour

In TTCN-3, the test behaviour is specified for independent concurrently executing test components. All behaviours are strictly local to a test component. Explicit synchronisation shall be specified whenever required. TDL test descriptions define a test behaviour from a global point of view. Rather than specifying the independent behaviour of individual components, TDL is concerned with the entire scenario. While there are certain types of behaviour that can be specified locally, interactions and combined behaviours span more than one component. Synchronisation can be explicit or implicit.

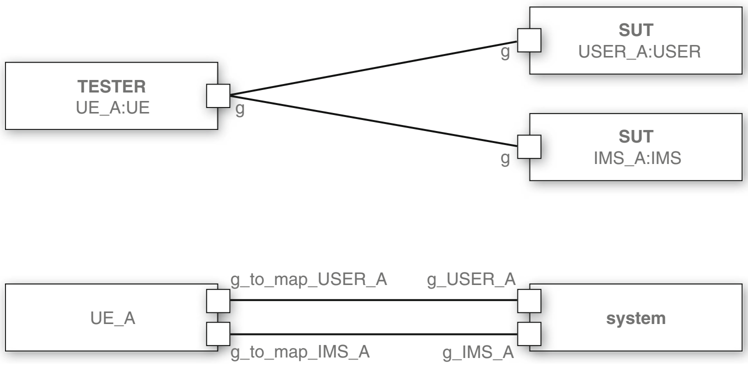


Fig. 11 TDL test configuration with multiple SUTs and its TTCN-3 equivalent

Since TTCN-3 is concerned only with the behaviour of the test components, the mapping captures only the behaviour specified for component instances in the role of Tester. Combined behaviours involving multiple component instances in the role of Tester shall be split for each participating component. Interactions shall be split into *send* and/or *receive* operations. Deviations from the specified behaviour need to be handled by means of *altsteps* activated as defaults, according to the test directives.

4.5 Mapping time

While TDL provides various means for expressing temporal aspects, including time constraints, time operations, and timers, TTCN-3 only provides timers in its core language. Extensions for TTCN-3, such as the TTCN-3 Performance and Real-Time Testing (ETSI ES 202 782 2015) and the Support of Interfaces with Continuous Signals (ETSI ES 202 786 2017) provide additional time-related features. The time constraints and time operations are currently mapped to timers and timer operations in TTCN-3 with certain restrictions. In the future, the mapping may also integrate time-related features from the extensions of TTCN-3.

5 An implementation of TDL

An open implementation of TDL is essential for validating the TDL standards by enabling the application of the specified concepts in practice and checking their applicability, consistency, and usability. The open implementation can also help tool vendors to accelerate the adoption of TDL in their existing solutions by using it as an import and/or export facility for TDL models, by adding custom features on top of it, or by integrating it with existing products.

5.1 Requirements

A basic TDL toolset shall be able to read and write models serialised according to the TDL-XF. It shall also allow users to check static semantics of models to verify their compliance with TDL according to the constraints specified in the TDL-MM.

TDL users may apply domain-specific textual notations mapped to the TDL-MM, such as the notation described in Annex B of the TDL-MM. Graphical representations are better suited for use in (standards-related) documents and during high-level discussions, for the visualization of tests generated from MBT tools and of test execution traces, as well as for instructional purposes. One of the main application scenarios for structured test objectives is in standards-related documentation. Instead of graphical shapes exported as images, the graphical representations of structured test objectives need to be exported as tables in a Word document according to user-defined templates. All these facilities need to be provided for efficient use of TDL in practice.

For the use of the UP4TDL, the profile itself needs to be implemented first. Using the facilities provided within modelling environments for manipulating features provided by UML profiles can be rather cumbersome. Hence, customised editing facilities for UML models using the UP4TDL are necessary to improve the usability of the UP4TDL implementation.

The availability of necessary tools and technologies for implementing model-based domain-specific languages is an important factor influencing the choice of platform for the open implementation of TDL.

5.2 Realisation

Eclipse and associated technologies were chosen as the base platform for the implementation as it is the most widely used modelling platform today. A high-level architectural overview of the implementation is shown in Fig. 12. At the core of the Eclipse modelling platform is the Eclipse Modelling Framework (EMF) which provides an implementation of MOF (named Ecore) which was used for the implementation of the TDL meta-model. The EMF provides the necessary facilities for model serialisation and de-serialisation based on XMI. These facilities were configured to work with XMI documents that are compliant with the TDL-XF. The semantic rules that are defined in the TDL standard as OCL constraints were specified as an add-on which can be applied to the model instances as needed. The Epsilon Validation Language (EVL) (Kolovos et al. 2009) provides means for the implementation of add-on constraints and extends the capabilities of OCL.

The Eclipse Graphical Modelling Framework (GMF) links EMF modelling capabilities with graphical editing. Diagram editors can be implemented by creating mappings between meta-model elements and diagram shapes, from which the code for graphical editors is generated. This separation facilitates the inclusion of custom shape and layout implementations. The Sirius project[2] utilises GMF and allows the declarative definition of diagram viewers and editors as opposed to code generation. Sirius was chosen as an implementation platform to avoid the maintenance overhead that comes with code generation. While the open implementation focuses on the visualisation of TDL models, it also provides essential editing capabilities.

The labels related to data use in the TDL-GR specification exhibit a more complex structure than most other labels. For the realisation of these labels, an Xtext-based serialisation relying on a partial grammar specification mapped to the TDL-MM was used. Xtext[3] is a framework for developing textual languages on top of EMF. Xtext was also used for the realisation of the syntax specified in Annex B of the TDL-MM enabling the quick creation and manipulation of TDL models. Similarly, an Xtext-based realisation of the syntax specified in Annex B of TDL-TO is also included for users relying on TDL mainly for the specification of structured test objectives.

The export of structured test objectives into tables in Word documents relies on facilities from the Docx4j[4] library providing an Application Programming Interface (API) for manipulating Word documents. A set of predefined templates, including a template according to the concrete syntax defined in TDL-TO, are included in the implementation. The templates describe the overall structure of the representation and contain a set of placeholders for the different labels which are substituted in a similar manner as the labels in the TDL-GR viewer. The described implementation architecture is illustrated in Fig. 12.

The implementation of the UP4TDL as well as supporting editing facilities was realised on top of Papyrus.[5] Papyrus provides a graphical editor and an extensible framework for working with UML models. It was customised with three new kinds of diagrams in order to allow users to create graphical representations of UML models applying UP4TDL with dedicated editing facilities.

[2]https://www.eclipse.org/sirius/

[3]https://eclipse.org/Xtext/

[4]http://www.docx4java.org

[5]https://www.eclipse.org/papyrus/

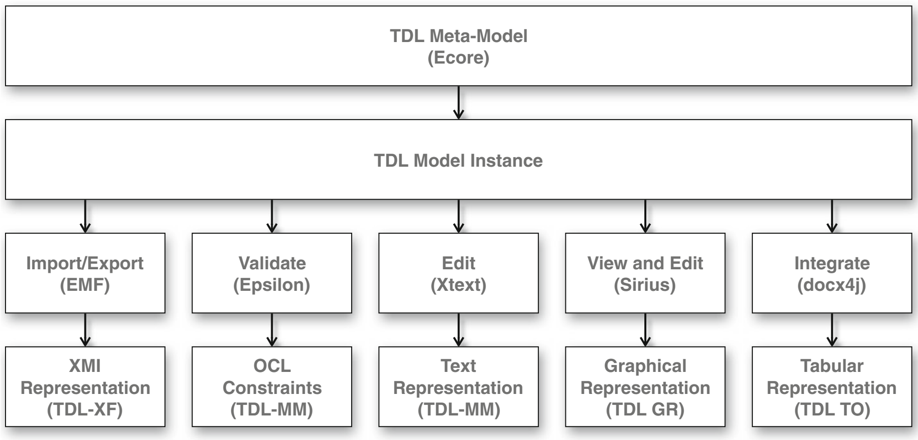


Fig. 12 Implementation architecture overview

5.3 Open-source project

In order to facilitate cooperation among the various stakeholders interested in the open implementation, TC MTS initiated an open-source project for the development of standard-compliant TDL tools in order lower the barrier to entry for both users and tool vendors. ETSI has contributed its existing source code to the project and made it available under Eclipse Public License (EPL). The TDL Open-Source Project (TOP)[6] is governed by TC MTS, but contributions are accepted from everyone according to contributor license agreements. A screen shot of some of the facilities provided by the TOP toolset is shown in Fig. 13. Shown on the top, from left to right: the project explorer and model tree, the textual editor, and the graphical editors for packages and test description behaviours. On the bottom part, there is a graphical or structural outline view as well as property inspectors and editors for the graphical editors where the arguments of an interaction can be specified, for example.

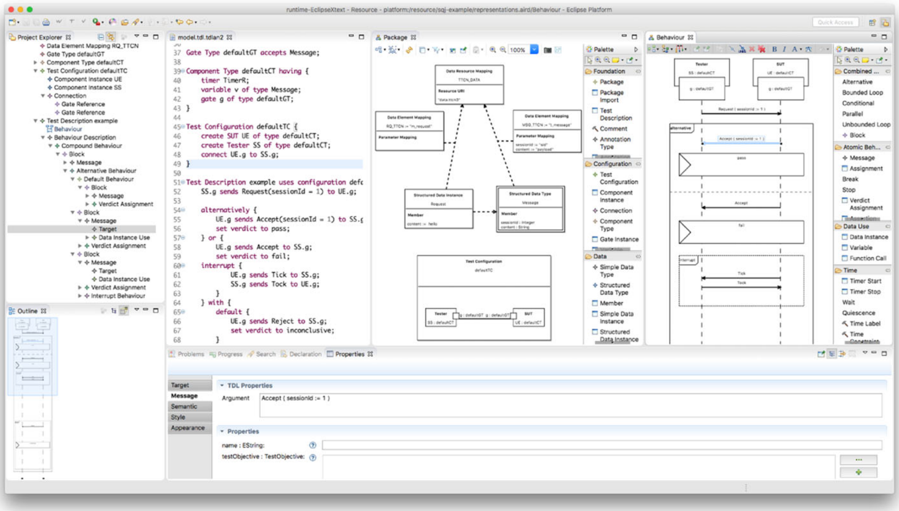
6 Using TDL in standardisation

In the scope of ETSI's mission to enable high-quality and interoperable standards, ETSI Technical Bodies (TBs) and funded projects produce public and open standardised test specifications for the industry. TDL, being an ETSI standard, is the language of choice to specify test descriptions during standardisation activities involving diverse technologies and communities.

In order to support both a wider adoption of TDL and the enhancement of tests standardisation activities within several ETSI TBs, the ETSI Centre for Testing and Interoperability (CTI)[7] initiated a study on the applicability of TDL for the standardisation of

[6]http://tdl.etsi.org and http://top.etsi.org

[7]http://www.etsi.org/about/how-we-work/testing-and-interoperability/centre-for-testing-and-interoperability



```
Gate Type defaultGT accepts Message;

Component Type defaultCT having {
    timer TimerR;
    variable v of type Message;
    gate g of type defaultGT;
}

Test Configuration defaultTC {
    create SUT UE of type defaultCT;
    create Tester SS of type defaultCT;
    connect UE.g to SS.g;
}

Test Description example uses configuration defc
    SS.g sends Request(sessionId = 1) to UE.g;

    alternatively {
        UE.g sends Accept(sessionId = 1) to SS.g
        set verdict to pass;
    } or {
        UE.g sends Accept to SS.g;
        set verdict to fail;
    interrupt {
            UE.g sends Tick to SS.g;
            SS.g sends Tock to UE.g;
        }
    } with {
        default {
            UE.g sends Reject to SS.g;
            set verdict to inconclusive;
        }
```

Fig. 13 TOP toolset

diverse technologies. The study focused on the flexibility of TDL in different contexts. It resulted in a set of non-trivial examples of test purposes and test descriptions which were obtained from manually translating the original publication format into TDL. These examples serve as a starting point for a more widespread use of TDL within ETSI standardisation groups.

Prior to using TDL, these specifications were typically provided in informal or, at best, semi-formal text documents that contain some parts of the specification in a tabular format which is augmented with additional text in free format whenever necessary.

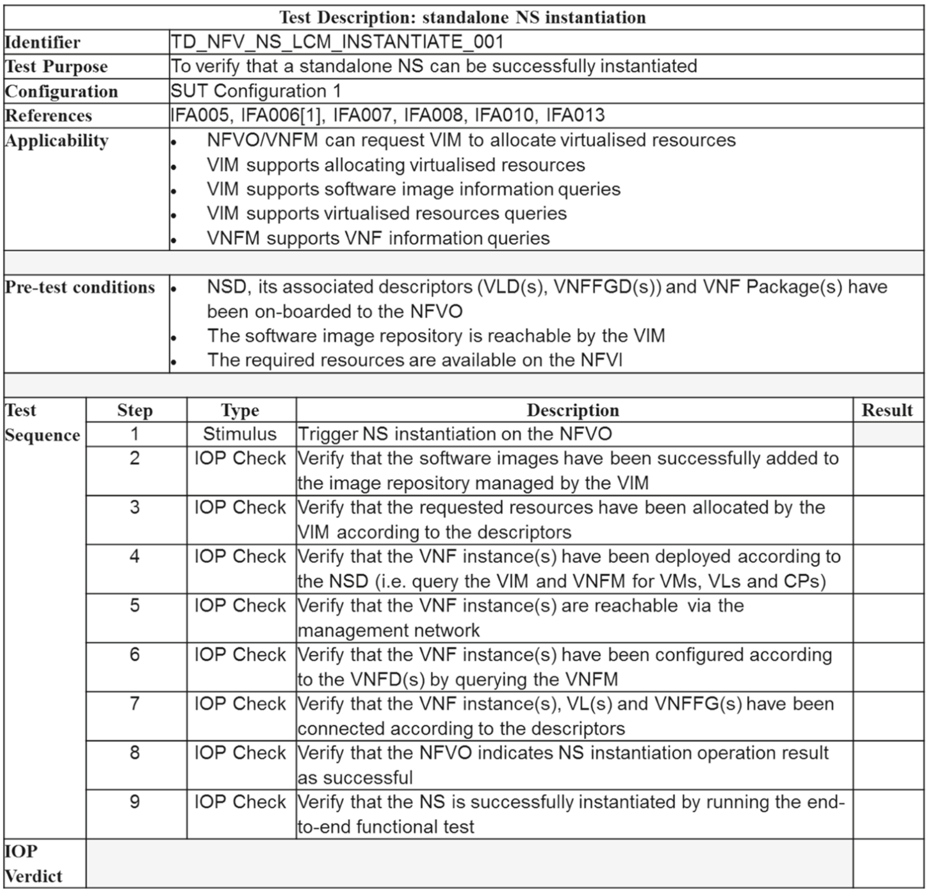
We report on our experiences from applying TDL on selected examples from the work of the ETSI Network Functions Virtualisation (NFV) group and Industry Specification Group (ISG) as well as the ETSI partnership projects oneM2M and 3GPP. Discussing all the details of the three examples at length would require introducing additional terminologies and test configurations which are out of scope of the present publication. Instead, we summarise each application context and focus on the requirements in terms of test modelling and test description and how they are met by TDL.

6.1 Interoperability testing in NFV ISG

Founded in November 2012 by seven of the world's leading telecommunications networks operators, ETSI NFV ISG[8] became the home of the NFV standardisation.

In the same way that applications are supported by dynamically configurable and fully automated cloud environments, virtualised network functions allow networks to be agile and capable to respond automatically to the needs of the traffic and services running over them.

[8]http://www.etsi.org/technologies-clusters/technologies/nfv



Test Description: standalone NS instantiation				
Identifier	TD_NFV_NS_LCM_INSTANTIATE_001			
Test Purpose	To verify that a standalone NS can be successfully instantiated			
Configuration	SUT Configuration 1			
References	IFA005, IFA006[1], IFA007, IFA008, IFA010, IFA013			
Applicability	• NFVO/VNFM can request VIM to allocate virtualised resources • VIM supports allocating virtualised resources • VIM supports software image information queries • VIM supports virtualised resources queries • VNFM supports VNF information queries			
Pre-test conditions	• NSD, its associated descriptors (VLD(s), VNFFGD(s)) and VNF Package(s) have been on-boarded to the NFVO • The software image repository is reachable by the VIM • The required resources are available on the NFVI			
Test Sequence	**Step**	**Type**	**Description**	**Result**
	1	Stimulus	Trigger NS instantiation on the NFVO	
	2	IOP Check	Verify that the software images have been successfully added to the image repository managed by the VIM	
	3	IOP Check	Verify that the requested resources have been allocated by the VIM according to the descriptors	
	4	IOP Check	Verify that the VNF instance(s) have been deployed according to the NSD (i.e. query the VIM and VNFM for VMs, VLs and CPs)	
	5	IOP Check	Verify that the VNF instance(s) are reachable via the management network	
	6	IOP Check	Verify that the VNF instance(s) have been configured according to the VNFD(s) by querying the VNFM	
	7	IOP Check	Verify that the VNF instance(s), VL(s) and VNFFG(s) have been connected according to the descriptors	
	8	IOP Check	Verify that the NFVO indicates NS instantiation operation result as successful	
	9	IOP Check	Verify that the NS is successfully instantiated by running the end-to-end functional test	
IOP Verdict				

Fig. 14 An interoperability test description example from NFV

The presented example deals with the instantiation of a network service (NS) which, in NFV terminology, is a composition of virtualised network functions and is defined by its functional and behavioural specification.[9]

The test case is taken from ETSI GR NFV-TST 007[10] which provides test descriptions formatted as tables containing two main parts: test information and test sequence. The original format is shown in Fig. 14. The test description is formatted as table containing steps in informal natural language. Some *Applicability* items are omitted for brevity.

The same test description translated to TDL and represented by means of the textual syntax is shown in Fig. 15 trivial and repetitive parts are omitted for brevity). Both the test information and the test sequence parts of the test description can be easily mapped to TDL elements. This example demonstrates the capability of TDL to contain portions of natural

[9]See ETSI NFV 003, http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_NFV003v010201p.pdf.

[10]http://www.etsi.org/deliver/etsi_gr/NFV-TST/001_099/007/01.01.01_60/gr_NFV-TST007v010101p.pdf

```
Test Objective TP_NFV_NS_LCM_INSTANTIATE_001 {
    from: "ETSI NFV-TST 007 7.6.2.1";
    description: "To verify that a standalone NS can be successfully instantiated";
} with { APPLICABILITY : "NFVO/VNFM can request VIM to allocate virtualised resources [...]"; }

Action preConditions: "NSD, its associated descriptors (VLD(s), VNFFGD(s)) and [...]";
Action triggerNSInst: "Trigger NS instantiation on the NFVO";

Test Description TD_NFV_NS_LCM_INSTANTIATE_001 uses configuration NFV_SUT_Configuration_1 {
    perform action preConditions with { PRECONDITION ; };
    perform action triggerNSInst with { STEP : "1"; TYPE: "stimulus" ; } ;
    perform action : "Verify that the software images have been successfully added to the image
                      repository managed by the VIM" with { STEP : "2"; TYPE: "IOP Check" ; };
    /* Similar for steps 3-9 ... */
}
```

Fig. 15 TDL representation of the example from Fig. 14 (abbreviated)

text which fit to the level of abstraction chosen by the editors of the NFV report, without the need to define more structured expressions when these are unnecessary.

6.2 Conformance testing in OneM2M

OneM2M[11] is a global standards initiative for Machine to Machine (M2M) communications and the Internet of Things (IoT). At their core, oneM2M standards specify a common service layer for storing, representing, and publishing data in a distributed database organised as a tree of containers and resources. The Common Service Element (CSE) is a standardised component meant to be deployed on each element in the network (e.g. smart objects, cloud infrastructure) and to be responsible for the management of the stored resources with respect to other CSEs or local Application Elements (AEs).

The OneM2M TST Working Group produces standardised test suites in the form of test purposes described in tabular format following a TPLan-like syntax (ETSI ES 202 553 2009) as well as standardised test cases specified in TTCN-3 (ETSI ES 201 873-1 2016).

The test purpose example used for the study is shown in its original publication format in Fig. 16. The test describes how a CSE should support the applications to delete a resource in a given resource container in a RESTful approach via the DELETE operation. The corresponding representation in the TDL-TO extension by means of the textual concrete syntax is shown in Fig. 17. TDL fits easily for describing such test purposes in the given context.

6.3 Conformance testing in 3GPP

We consider an example taken from the test suites standardised by the 3rd Generation Partnership Project (3GPP),[12] the worldwide initiative to standardise mobile networks. The example is taken from 3GPP TS 36.523-1 (version 14.2.0).[13] In the context of conformance tests for user equipment in Long Term Evolution (LTE) networks, it specifies how to test the behaviour of the User Equipment (UE) when dealing with a specific case of mobility among radio cells.

Within 3GPP, the Technical Specification Group (TSG) Technical Specification Group (TSG) Working Group (WG)5 has the objective to specify test purposes and test descriptions

[11] http://www.onem2m.org

[12] http://www.3gpp.org

[13] http://www.etsi.org/deliver/etsi_ts/136500_136599/13652301/14.02.00_60/ts_13652301v140200p.pdf

TP Id	TP/oneM2M/CSE/DMR/DEL/005	
Test objective	Check that the stateTag attribute of a container resource is increased when a child resource is deleted	
Reference	TS-0001 10.1.4	
Config Id	CF01	
PICS Selection	PICS_CSE	
Initial conditions	**with {** the IUT **being** in the "initial state" **and** the IUT **having registered** the AE **and** the IUT **having created** a container resource **containing** a child resource **and** the AE **having** privileges to perform DELETE operation on the TARGET_CHILD_RESOURCE_ADDRESS **}**	
Expected behaviour	**Test events**	**Direction**
	when { the IUT **receives** a valid DELETE Request **from** AE **containing** To **set to** TARGET_CHILD_RESOURCE_ADDRESS **and** From **set to** AE_ID **}**	IUT 🡐 AE
	then { the IUT **increments** the stateTag attribute of the container resource **and** the IUT sends a valid Response **containing** Response Status Code **set to** 2002 (DELETED) **}**	IUT 🡒 AE

Fig. 16 Original representation of a test purpose from OneM2M TS 0018, Sec 7.2.2.3

to assess conformance (to the 3GPP standards) of the the UE (e.g. a mobile phone), when it is communicating with the network (radio access network and core network). Test cases are specified in RAN WG5 by means of tables in text documents, where every row is a step in the test sequence. In order to express parallel or optional behaviour sub-tables are created and referenced between the steps of the main test sequence. In the specification

```
Test Purpose {
    TP Id "TP/oneM2M/CSE/DMR/DEL/005"
    Test objective "Check that the state Tag attribute of a container
                    resource is created when a child resource is deleted"
    Reference "TS-0001 10.1.4" PICS Selection PIC_CSE
    Initial conditions
        with {
            the IUT entity being_in the initial state
        and the IUT entity having registered the AE
        and the IUT entity having created a non empty container resource
        and the AE entity having the DELETE operation privileges
                for the target child resource entity
        }
    Expected behaviour ensure that {
        when {
            the IUT entity receives a valid DELETE Request containing
                To indicating value TARGET_CHILD_RESOURCE_ADDRESS,
                From indicating value AE_ID;
        }
        then {
            the IUT entity increments the stateTag attribute
                in the container resource entity
        and the IUT entity sends a valid Response containing
                Response Status Code set to 2002; /* (DELETED) */
        }
    }
}
```

Fig. 17 The test purpose example from Fig. 16 modelled in the TDL-TO textual syntax

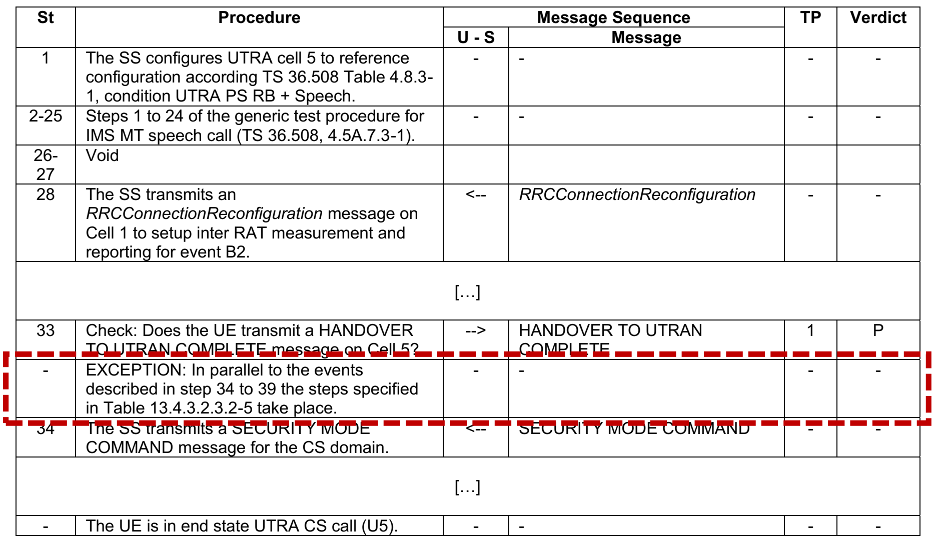
taken as an example, for the inter-system mobility voice conformance test more than 30 test purposes and related test descriptions are defined. The behaviour of the test case in the present example contains around 40 steps with possible parallel behaviours and data types overall. They are described using around a dozen tables with references to other data types and behaviour described in other sections of the specification or in other specifications. Understandably, the amount of work needed to maintain, update, and implement the test cases is quite substantial.

To exemplify how the test behaviours are described, a partial representation of table containing the main behaviour of the test is shown in Fig. 18 (some of the steps are omitted for brevity). The dashed box highlights a reference to a behaviour specified in another table in another part of the specification. The referenced behaviour is shown in Fig. 19. The row after step 3 contains further information on how to interpret the subsequent steps.

In Fig. 20 the same main behaviour is described (as a skeleton) in a few lines by using the textual concrete syntax for TDL. As this representation encodes structural dependencies between behaviours in a formalised way, it enables automation for many maintenance tasks. The possibility to extract and refer to sub-sequences of the test behaviour improves the readability and re-usability of the standardised test descriptions with respect to the actual representation, thus improving quality and efficiency of the standardisation process. The example illustrates the ability of TDL to assist with managing lengthy and complex test behaviours within a test description more efficiently at scale.

7 Related work

Domain-specific languages for testing offer a convenient solution, as they allow domain experts to use familiar concepts to express and describe the system behaviour for the purposes of testing. For example, automotive engineers may rely on dedicated concepts, such as Electronic Control Unit (ECU), bus, power port, etc., from the automotive domain in a domain-specific testing language tailored for testing automotive systems.



St	Procedure	Message Sequence		TP	Verdict
		U - S	Message		
1	The SS configures UTRA cell 5 to reference configuration according TS 36.508 Table 4.8.3-1, condition UTRA PS RB + Speech.	-	-	-	-
2-25	Steps 1 to 24 of the generic test procedure for IMS MT speech call (TS 36.508, 4.5A.7.3-1).	-	-	-	-
26-27	Void				
28	The SS transmits an *RRCConnectionReconfiguration* message on Cell 1 to setup inter RAT measurement and reporting for event B2.	<--	*RRCConnectionReconfiguration*	-	-
[…]					
33	Check: Does the UE transmit a HANDOVER TO UTRAN COMPLETE message on Cell 5?	-->	HANDOVER TO UTRAN COMPLETE	1	P
-	EXCEPTION: In parallel to the events described in step 34 to 39 the steps specified in Table 13.4.3.2.3.2-5 take place.	-	-	-	-
34	The SS transmits a SECURITY MODE COMMAND message for the CS domain.	<--	SECURITY MODE COMMAND	-	-
[…]					
-	The UE is in end state UTRA CS call (U5).	-	-	-	-

Fig. 18 The main behaviour for the test case 13.4.3.2 in 3GPP TS 36.523-1

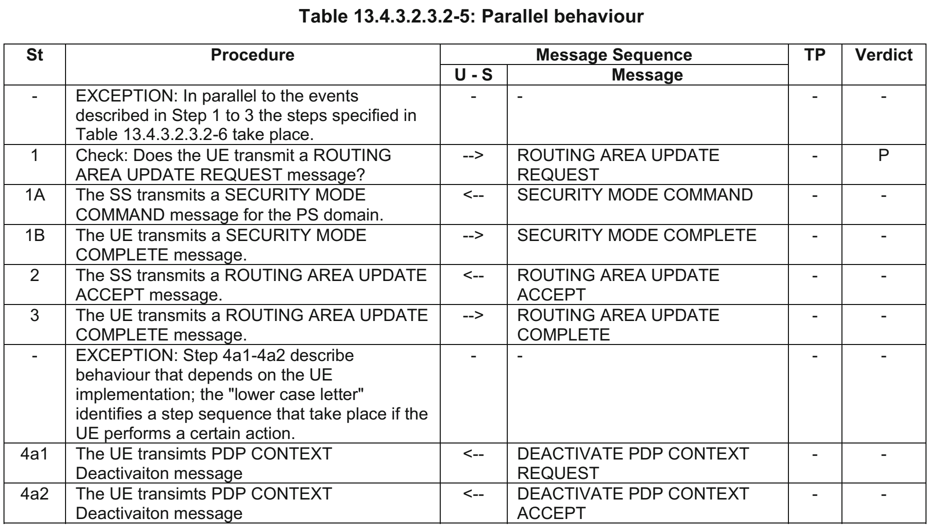


Table 13.4.3.2.3.2-5: Parallel behaviour

St	Procedure	Message Sequence		TP	Verdict
		U - S	Message		
-	EXCEPTION: In parallel to the events described in Step 1 to 3 the steps specified in Table 13.4.3.2.3.2-6 take place.	-	-	-	-
1	Check: Does the UE transmit a ROUTING AREA UPDATE REQUEST message?	-->	ROUTING AREA UPDATE REQUEST	-	P
1A	The SS transmits a SECURITY MODE COMMAND message for the PS domain.	<--	SECURITY MODE COMMAND	-	-
1B	The UE transmits a SECURITY MODE COMPLETE message.	-->	SECURITY MODE COMPLETE	-	-
2	The SS transmits a ROUTING AREA UPDATE ACCEPT message.	<--	ROUTING AREA UPDATE ACCEPT	-	-
3	The UE transmits a ROUTING AREA UPDATE COMPLETE message.	-->	ROUTING AREA UPDATE COMPLETE	-	-
-	EXCEPTION: Step 4a1-4a2 describe behaviour that depends on the UE implementation; the "lower case letter" identifies a step sequence that take place if the UE performs a certain action.	-	-	-	-
4a1	The UE transimts PDP CONTEXT Deactivaiton message	<--	DEACTIVATE PDP CONTEXT REQUEST	-	-
4a2	The UE transimts PDP CONTEXT Deactivaiton message	<--	DEACTIVATE PDP CONTEXT ACCEPT	-	-

Fig. 19 The parallel behaviour referenced by the table in Fig. 18

The general standard on software testing ISO/IEC/IEEE 29119-3 (2013) provides guidelines for the test management process documentation, such as test plan and test strategy, as well as test process documentation that includes the test design specification, test case specification and other documents. Although it proposes a structure for these specifications, they are defined without a strict semantics of their contents.

The Check Case Definition Language (CCDL) (Razorcat 2014) provides a high-level approach to requirements-based black-box system level testing embedded in its own testing process. Test simulations and expected results are specified in human readable form and can be compiled into executable test scripts. However, due to lack of standardisation, the test

```
Test Description TD_13_4_3_2 uses configuration BaseConfiguration {
    // placeholder for steps 1-33
    perform action : "Steps 1-33";
    run {
        // placeholder for steps 34-39
        perform action : "Steps 34-39";
    } in parallel to {
        // table 13.4.3.2.3.2-5
        execute TD_13_4_3_2_3_2_5;
        run {
            // placeholder for steps 1-3
            perform action : "Steps 1-3";
        } in parallel to {
            // table 13.4.3.2.3.2-6
            execute TD_13_4_3_2_3_2_6;
        }

        // placeholder for steps 4a1-4a2
        perform action : "Steps 4a1-4a2";
    }
}
```

Fig. 20 TDL representation of the behaviour from Figs. 18 and 19

descriptions in CCDL are heavily tool-dependent. High-level keyword-based test languages, using frameworks such as the Robot Framework,[14] have also been integrated with MBT (Pajunen et al. 2011). Beyond textual keyword-based languages, graphical domain-specific testing languages, such as the one built on top of TTCN-3 and discussed in Kanstrén et al. (2012), have also been developed.

There have been efforts to address the lack of standardisation in several application domains, e.g. the standardised meta-model for testing avionics embedded systems (Guduvan et al. 2013) and the Automotive Test Exchange Format (ATX) standard (ASAM ATX 2012) and TestML definition (Grossmann and Müller 2006) that focus on automotive systems. Additionally, the ISO standard on the Open Test Sequence Exchange Format (OTX) (ISO 13209 2011) aims at providing a tool-independent XML-based data exchange format for the formal description and documentation of executable test sequences in automobile diagnostics. These efforts have focused primarily on enabling the exchange of test specifications between involved stakeholders and tools and are hardly concerned with precise semantics. The domain and purpose specialisation of these languages limits their applicability outside of the originally intended domain and testing activity.

The Message Sequence Chart (MSC) standard (ITU-T Z120 2011) is one of the first languages for the specification of scenarios, not focusing strictly on testing. In addition to the main specification, Annex B (ITU-T Z120 Annex B 1998) provides a formal specification of the semantics of MSC. Some of the features of MSC were subsequently adopted in Object Management Group (OMG)'s UML in the form of Sequence Diagrams. While allowing specialisation of the sequence diagram for different situations and domains, the loose semantics of UML and the different potential usages and interpretations of sequence diagrams as discussed in Micskei and Waeselynck (2010) are a limiting factor for its use as a universal and consistent test description language.

The OMG UTP (2013) profile adds testing related concepts to the UML, thus enabling test modelling in UML. While it maintains the wide scope of UML, it also inherits its disadvantage associated with the feeble UML semantics which results in modelling solutions that are locked in to specific vendors. The UML Profile for TDL (UP4TDL) discussed in Section 3.7 transfers the concepts of TDL to the UML world and provides more specialised and integrated means for test modelling within the scope of UML by inheriting the TDL semantics.

The Precise UML approach for MBT (Bouquet et al. 2007) introduces a subset of UML and OCL to seen an end to the open-ended interpretations of the semantics of different diagrams supporting the specification of behavioural models. However its scope is narrowed down to UML class, object, and state-machine diagrams that are frequently used for test generation from SUT design models especially for embedded software systems that the authors of the approach mainly target.

Test specification approaches based on a concrete executable language with strict semantics, such as TTCN-3 (ETSI ES 201 873-1 2016), enable the exchange of executable tests between different stakeholders. However, such languages are not well suited for review and high-level discussions due to the level of detail these test cases contain. Moreover stakeholders need to be able to understand the programming language-like syntax before comprehending the meaning of a given test.

New testing techniques that stem from agile development methods such as TDD (Hammond and Umphress 2012) and Behaviour Driven Development (BDD) (Solis and Wang

[14]https://robotframework.org

2011) rely heavily on the specification of so-called *user stories*, from which test scenarios can be derived (Kaner 2003). TDL is able to directly support such a scenario-based testing approach. Model-based approaches appear somewhat contrary to agile methods, which are predominantly used in today's industrial practice, because they rely on the creation of models prior to their implementation as a separate and independent step in the development process. Therefore there is a strong need from practice to adapt and propose new model-based approaches that fit into agile development processes.

TDL is different compared to the approaches highlighted above due to its common standardised meta-model with an associated, well-defined semantics which provides a solid foundation for interoperability between tools and helps in consolidating different test design approaches. In particular in the standardisation process that delivers conformance and interoperability test suites together with high-level system design specifications, TDL can make a contribution to streamline the test design process and improve the quality of test objectives and test cases.

8 Conclusions

TDL has been designed to address existing challenges in the test development process. By abstracting away implementation details, TDL enables test engineers to focus on the task of describing the test scenarios that cover the given test objectives, rather than work their way through to produce executable tests on a specific test execution framework. It also helps to make test specifications easier to review by non-testing experts. The creation of test specifications is still a major step in the development of software systems and enforced by BDD and other agile practices in industry as well as in standardisation processes. TDL fits into this context and offers a way forward to improve the overall productivity and quality of test development.

In this article, an overview of TDL was presented covering its conception as a multi-part standard and its usage within a test methodology that is followed in standardisation work and industrial projects alike. The standard covers the different language facets of abstract syntax (meta-model), graphical syntax, exchange format, and the TDL UML profile. It also provides the TDL-TO extension for writing structured test objective specifications.

Moreover, a summary of the mapping from TDL to the executable test language TTCN-3 was presented, which is also subject to future research and standardisation efforts. This work is important because the standardised mapping of TDL test descriptions to TTCN-3 test cases enables the semi-automatic generation of executable tests and protects previous investments in TTCN-3 test tools and frameworks. In addition, the mapping will leverage the impact of TDL and ensure that there is a consistent and common way of implementing test cases based on TDL.

To accelerate the adoption of TDL, an open-source toolset for TDL available under the TOP project was developed. It provides textual and graphical editors that can be deployed directly by users or used as a foundation to propel tool vendors in building their own solutions.

During the launch events of TDL at the User Conference on Advanced Automated Testing (UCAAT) 2015 and TOP at UCAAT 2017, multiple stakeholders expressed interest in adopting TDL for functional as well as non-functional testing, mainly security and performance testing in the information and communications technology domain. A next step is to identify and catalogue the requirements for the adaptation of TDL to these different types of testing in order to evolve the language according to practical needs.

With contributions from numerous partners from industry, academia, and standardisation, TDL is gaining ground as a test specification technique for software-intense systems. User feedback on the first application cases of TDL at ETSI provide optimism of an upcoming wide use at ETSI and beyond.

Funding information The work on Test Description Language (TDL) has been partially funded by the European Telecommunications Standards Institute (ETSI) in the context of the Specialist Task Force (STF) projects 454, 476, 492, and 522 between 2013 and 2017.

References

ASAM ATX (2012). Automotive test exchange format (ATX) v1.0.0. ASAM standard. https://www.asam.net/standards/detail/atx/.

Bouquet, F., Grandpierre, C., Legeard, B., Peureux, F., Vacelet, N., Utting, M. (2007). A subset of precise uml for model-based testing. In *Proceedings of the 3rd Int'l workshop on advances in model-based testing (A-MOST'07)* (pp. 95–104). New York: ACM.

ETSI EG 203 130 (2013). Methods for Testing and Specification (MTS); Model-Based Testing (MBT); methodology for standardised test specification development, v1.1.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 201 873-1 (2016). Methods for Testing and Specification (MTS); the testing and test control notation version 3; Part 1: core language, v4.8.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 202 553 (2009). Methods for Testing and Specification (MTS); TPLan: a notation for expressing test purposes, v1.2.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 202 782 (2015). Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 extensions: performance and real-time testing, v1.3.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 202 786 (2017). Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Extensions: support of interfaces with continuous signals, v1.4.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 202 951 (2011). Methods for Testing and Specification (MTS); Model-Based Testing (MBT); requirements for modelling notations, v1.1.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 203 119-1 (2018). Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 1: abstract syntax and associated semantics, v1.4.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 203 119-2 (2018). Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 2: graphical syntax, v1.3.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 203 119-3 (2018). Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 3: exchange format, v1.3.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 203 119-4 (2018). Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 4: structured test objective specification (extension), v1.3.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 203 119-5 (2018). Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 5: UML profile for TDL, v1.1.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 203 119-6 (2018). Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 6: mapping to TTCN-3, v1.1.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI ES 203 119-7 (2018). Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 7: extended test configurations, v1.1.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

ETSI TR 102 840 (2011). Methods for Testing and Specification (MTS); Model-Based Testing (MBT); Model-Based Testing in Standardisation, v1.2.1 European Telecommunications Standards Institute (ETSI). Sophia-Antipolis, France.

Grossmann, J., & Müller, W. (2006). A formal behavioral semantics for testML. In *Second international symposium on leveraging applications of formal methods, verification and validation, 2006. ISoLA 2006* (pp. 441-448).

Guduvan, A., Waeselynck, H., Wiels, V., Durrieu, G., Fusero, Y., Schieber, M. (2013). A meta-model for tests of avionics embedded systems. In *Proc. of the 1st Int'l conference on model-driven engineering and software development (MODELSWARD)* (pp. 5–13). Barcelona.

Hammond, S., & Umphress, D. (2012). Test driven development: the state of the practice. In *Proc. of the 50th annual southeast regional conference* (pp. 158–163). ACM.

ISO 13209 (2011). Road vehicles – Open Test sequence eXchange format (OTX) – Parts 1–3. International ISO multipart standard No. 13209-1 to 3. https://www.iso.org/ics/43.020/x/.

ISO/IEC 9646-1 (1994). Information technology – open systems interconnection – conformance testing methodology and framework – Part 1: general concepts. International ISO/IEC multipart standard No. 9646-1. https://www.iso.org/standard/17473.html.

ISO/IEC/IEEE 29119-3 (2013). ISO/IEC/IEEE international standard – software and systems engineering – software testing – part 3: test documentation. Tech. rep., ISO/IEC/IEEE. https://ieeexplore.ieee.org/servlet/opac?punumber=6588538.

ITU-T Z120 (2011). Series Z: languages and general software aspect for telecommunication systems – formal description techniques (FDT) – message sequence chart (MSC). International Telecommunication Union, Recommendation Z.120 (02/11), document http://www.itu.int/rec/T-REC-Z.120-201102-I/en.

ITU-T Z120 Annex B (1998). Formal semantics of message sequence charts, 04/98. International telecommunication union, recommendation Z.120 Annex B (04/98), document http://www.itu.int/rec/T-REC-Z.120-199804-I!AnnB/en.

Kaner, C. (2003). On scenario testing. *STQE Magazine*, 16–22.

Kanstrén, T., Puolitaival, O.P., Rytky, V.M., Saarela, A., Keränen, J.S. (2012). Experiences in setting up domain-specific model-based testing. In *2012 IEEE International conference on industrial technology (ICIT)* (pp. 319–324).

Kolovos, D.S., Paige, R.F., Polack, F.A.C. (2009). On the evolution of OCL for capturing structural constraints in modelling languages. In *Rigorous methods for software construction and analysis. Essays Dedicated to Egon Börger on the Occasion of His 60th Birthday* (pp. 204–218).

Makedonski, P., Adamis, G., Käärik, M., Kristoffersen, F., Zeitoun, X. (2016). Evolving the ETSI test description language. In *Proc. of the 9th system analysis and modelling conference (SAM 2016)*. Springer.

Micskei, Z., & Waeselynck, H. (2010). The many meanings of UML 2 Sequence Diagrams: A survey. *Software & Systems Modeling*, *10*(4), 489–514.

OMG MARTE (2011). UML profile for MARTE: modeling and analysis of real-time embedded systems, version 1.1. object management group, document number: formal/2011-06-02, document http://www.omg.org/spec/MARTE/1.1/.

OMG MOF (2014). Meta object facility core, version 2.4.2. Object management group, document number: formal/2014-04-05, document http://www.omg.org/spec/MOF/2.4.2/.

OMG OCL (2012). Object constraint language, version 2.3.1. Object management group, document number: formal/2012-05-09, document http://www.omg.org/spec/OCL/2.3.1/.

OMG UML (2015). Unified modeling language, version 2.5, chapter 12.3 (profiles). Object management group, document number: formal/2015-03-01, document http://www.omg.org/spec/UML/2.5/.

OMG UTP (2013). UML testing profile, version 1.2. Object management group, document number: formal/2013-04-03, document http://www.omg.org/spec/UTP/1.2/.

OMG XMI (2014). XML metadata interchange, version 2.4.2. Object management group, document number: formal/2014-04-06, document http://www.omg.org/spec/XMI/2.5.1/.

Pajunen, T., Takala, T., Katara, M. (2011). Model-based testing with a general purpose keyword-driven test automation framework. In *4th IEEE Int'l conference on software testing, verification and validation, ICST 2012* (pp. 242–251). Berlin: Workshop Proceedings.

Razorcat (2014). CCDL whitepaper. Razorcat Technical Report, 23 January 2014. https://www.razorcat.com/de/downloads-ccdl.html.

Solis, C., & Wang, X. (2011). A study of the characteristics of behaviour driven development. In *2011 37th EUROMICRO Conference on software engineering and advanced applications (SEAA)* (pp. 383–387). IEEE.

Ulrich, A., Jell, S., Votintseva, A., Kull, A. (2014). The ETSI test description language TDL and its application. In 2ND INT'L CONFERENCE ON MODEL-DRIVEN ENGINEERING AND SOFTWARE DEVELOPMENT (MODELSWARD) (pp. 601–608).

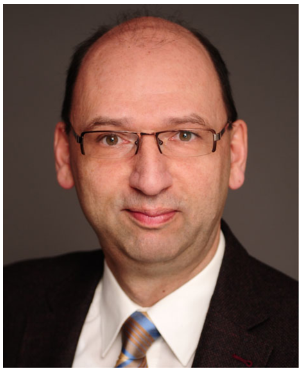
Philip Makedonski is a researcher at the Georg-August-Universität Göttingen. His research interests include software mining, human aspects in software evolution and maintenance, and software quality assurance. He is involved in the development and standardisation of TDL at ETSI within STFs 454, 476, 492, and 522, and also in the development and maintenance of the TTCN-3 open-source tools TRex, T3Q, and T3D.

Gusztáv Adamis is an honorary associate professor at the Budapest University of Technology and Economics. Since 2010, he is an architect at Ericsson's Test Competence Centre. He is involved in the development and standardisation of TDL at ETSI within STFs 454, 476, 492, and 522.

Martti Käärik is quality assurance project manager and test tool developer at Elvior. He is involved in the development and standardisation of TDL at ETSI within STFs 476, 492, and 522.

Finn Kristoffersen is a software engineer at the tool vendor, Cinderella ApS. He has specialised in tool development for system specification and validation and has been involved in the standardisation of specification languages and in development of standardised TTCN-3 test suites. He is involved in the development and standardisation of TDL at ETSI within STFs 454, 492, and 522.

Michele Carignani is Technical Expert within ETSI's Centre for Testing and Interoperability (CTI). He has a M.Sc. in Computer Science and Networking. Before joining ETSI, he was research fellow in CNIT (University Consortium for Telecommunication in Italy) working mainly on IoT and ITS research projects.



Andreas Ulrich works as an in-house consultant and technology adviser at Siemens AG, Corporate Technology in Munich, Germany. His main task is to support Siemens business units in coping with changes in software technologies and software development practices. His particular focus is on software system modelling, verification and validation, and testing for a wide scope of software systems, including embedded, cloud, and control software. Andreas is active at ETSI, where he has contributed to the development of TTCN-3 and later he became one of the initiators, designers, and promoters of the Test Description Language (TDL).

Jens Grabowski is professor at the Georg-August-Universität Göttingen. He is vice director of the Institute of Computer Science and is heading the Software Engineering for Distributed Systems Group at the Institute. Prof. Grabowski is one of the developers of the standardised testing languages TTCN-3 and UML Testing Profile. The current research interests of Prof. Grabowski are directed towards model-based development and testing, managed software evolution, and empirical software engineering.

Affiliations

Philip Makedonski[1] · Gusztáv Adamis[2] · Martti Käärik[3] · Finn Kristoffersen[4] · Michele Carignani[5] · Andreas Ulrich[6] · Jens Grabowski[1]

Gusztáv Adamis
gusztav.adamis@ericsson.com

Martti Käärik
martti.kaarik@elvior.com

Finn Kristoffersen
finn@cinderella.dk

Michele Carignani
michele.carignani@etsi.org

Andreas Ulrich
andreas.ulrich@siemens.com

Jens Grabowski
grabowski@cs.uni-goettingen.de

1 Institute of Computer Science, University of Göttingen, Göttingen, Germany

2 Test Competence Center, Ericsson Hungary Ltd., Budapest, Hungary

3 Elvior OU, Tallinn, Estonia

4 Cinderella ApS, Copenhagen, Denmark

5 Centre for Testing and Interoperability, European Telecommunications Standards Institute, Sophia-Antipolis, France

6 Corporate Technology, Siemens AG, Munich, Germany