Evolving the ETSI Test Description Language

Philip Makedonski[1(✉)], Gusztáv Adamis[2], Martti Käärik[3], Finn Kristoffersen[4], and Xavier Zeitoun[5]

[1] Institute of Computer Science, University of Göttingen, Göttingen, Germany
makedonski@cs.uni-goettingen.de
[2] Test Competence Center, Ericsson Hungary Ltd., Budapest, Hungary
gusztav.adamis@ericsson.com
[3] Elvior OU, Talinn, Estonia
martti.kaarik@elvior.com
[4] Cinderella ApS, Copenhagen, Denmark
finn@cinderella.dk
[5] CEA, LIST, Gif-sur-yvette, France
xavier.zeitoun@cea.fr

Abstract. Increasing software and system complexity due to the integration of more and more diverse sub-systems presents new testing challenges. Standardisation and certification requirements in certain domains such as telecommunication, automotive, aerospace, and health-care contribute further challenges for testing systems operating in these domains. Consequently, there is a need for suitable methodologies, processes, languages, and tools to address these testing challenges. To address some of these challenges, the Test Description Language (TDL) has been developed at the European Telecommunications Standards Institute (ETSI) over the past three years. TDL bridges the gap between declarative test purposes and imperative test cases by offering a standardised language for the specification of test descriptions. TDL started as a standardised meta-model, subsequently enriched with a graphical syntax, exchange format, and a UML profile. A reference implementation of TDL has been developed as a common platform to accelerate the adoption of TDL and lower the barrier to entry for both end-users and tool-vendors. This article tells the story of the evolution of TDL from its conception.

Keywords: Model-based testing · Test description language · Domain-specific modeling

1 Introduction

Increasing software and system complexity due to the integration of more and more diverse sub-systems presents new testing challenges. Standardisation and certification requirements in certain domains such as telecommunication, automotive, aerospace, and health-care contribute further testing challenges for systems operating in these domains, especially as they need to evolve and operate

J. Grabowski and S. Herbold (Eds.): SAM 2016, LNCS 9959, pp. 116–131, 2016.
DOI: 10.1007/978-3-319-46613-2_8

over long periods of time. Consequently, there is a need for suitable methodologies, processes, languages, and tools to address these testing challenges. The European Telecommunications Standards Institute (ETSI) has a long experience with the development of test specifications for standardised systems. To facilitate the efficient development of standardised test specifications and address some of the continuously evolving challenges, the Technical Committee Methods for Testing and Specification (TC-MTS) at ETSI has been active in developing methodologies, processes, and languages for testing, and in particular in the context of standardisation. The ETSI test development process [17] follows a stepwise approach based on the ISO/IEC 9646-1 [25] norm. Each step on the way from base standard to executable test cases results in intermediate artifacts at different levels of abstraction, which are intended for particular stakeholders such as standardisation experts, technology experts, and test engineers.

With strong emphasis on test automation, TC-MTS lead the work on the development and maintenance and evolution of the Testing and Test Control Notation version 3 (TTCN-3) [10] over the past 15 years. While TTCN-3 has been established as the language of choice for the implementation of test cases at ETSI, on the higher levels of abstraction there was a distinctive lack of well-established notations. Even with overall agreement on the basic structure and content of test purposes and test descriptions, there was a proliferation of dialects and customised notations for different standards. The Test Purpose Notation (TPLan) [11] sought to provide a notation for the standardised specification of test purposes. This left a gap between the declarative test purposes and the imperative test cases. Without a suitable and standardised language for this purpose, the development of test descriptions by means of different notations and dialects lead to significant overhead and frequent inconsistencies that needed 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. More recently, TC-MTS has also explored the requirements for the application of Model-Based Testing (MBT) in standardisation [12] with MBT technologies becoming more mature and finding wider acceptance in the industry.

The Test Description Language (TDL) [13] seeks to bridge the methodological gap between declarative test purposes and executable test cases by providing a formalised model-based solution for the specification of test descriptions. At the core of TDL there is a common meta-model with well-defined semantics, which can be represented by means of different concrete notations. A TDL test description created in one notation can be reviewed and approved in other notations, customized to suit the preferred level of abstraction and notational conventions of the different users. TDL can also serve as an exchange and visualisation platform for generated tests, contributing to the ongoing activities within TC-MTS to establish MBT technologies within standardisation at ETSI [9,12,17].

TC-MTS laid down the foundation of TDL with Specialist Task Force (STF) 454 in 2013 in terms of the basic concepts of the language and their semantics. In 2014, STF 476 added language functionality for the integration of TDL test descriptions into test automation frameworks as well as a standardised graphical

syntax for end-users. STF 476 also contributed an exchange format in order to foster tool interoperability, as well as an extension to TDL enabling refined test objective specification. In 2015, STF 492 developed an open reference implementation intended to serve as a common platform to accelerate the adoption of TDL and lower the barrier to entry for both end-users and tool-vendors. The work on the reference implementation contributed to the public launch of TDL at User Conference on Advanced Automated Testing (UCAAT) 2015 sharing the work on TDL with a broader audience. The discussions during the launch event generated feedback from numerous stakeholders from different domains that will influence the future development of the language.

This article is structured as follows: Sect. 2 showcases the different parts of the TDL standard in their current form. Section 3 contains a technical overview of the reference implementation of TDL. Section 4 discusses related work. Finally, Sect. 5 provides a summary and an outlook on the future of TDL.

2 The TDL Standard

The TDL specification evolved into a multi-part standard. In this section, we first provide a broad overview of the core principles behind the design of TDL and then take a closer look at the different parts and illustrate some of the features of the language with examples.

2.1 Core Design Principles

The TDL is intended for the design, documentation, and representation of formal test descriptions for black-box testing following a scenario-based approach describing interactions with the System Under Test (SUT). Test objectives derived from requirements may be attached to different scenarios or even to parts of a scenario. The scenarios are then used as basis for deriving and automating tests. TDL can also be used for the representation of test sequences derived from MBT tools, system simulators, or traces from test execution runs.

At the core of TDL is the Meta-model (MM) [13] standard specifying the abstract syntax including the concepts of the language, the relationships among them, their properties, and their intended semantics. The Exchange Format (XF) [15] serves as basis for the interoperability of tools. Concrete syntax notations may be mapped to the abstract syntax making the elements of the meta-model accessible to users by means of different representations, such as graphical, textual, tabular, tree-based, possibly targeting different levels of abstraction. The Graphical Representation (GR) [14] provides a standardised concrete syntax for the graphical representation of TDL elements as a common ground. The Structured Test Objective Specification (TO) [16] provides an extension of TDL that introduces additional concepts to the MM, as well as corresponding representations in the GR and the XF.

The decomposition of TDL into a multipart 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 group of users may opt to support only the MM and XF parts, rather than supporting the GR which may be unnecessary for the targeted user group.

2.2 The Meta-Model

Part 1 of the TDL standard defines the abstract syntax, static and dynamic semantics of TDL. The abstract syntax is specified in terms of a Meta-Object Facility (MOF) [33] meta-model describing the concepts of the language and the relations between them. Constraints on the concepts and their relationships are formalized by means of the Object Constraint Language (OCL) [30]. The meta-model concepts are defined in packages covering different aspects of TDL.

The *Foundation* package defines the basic structural concepts of a TDL specification. These include the abstract notion of an element as the common ancestor of all other concepts of the language, packages for grouping elements together, concepts for importing elements from packages, annotations, and comments.

The *Data* package contains concepts for data definition and data use. The data definition concepts cover abstract data type definitions, data instance specifications, actions, functions, parameters, and variables. Members and member assignments may be used to specify the internal structure of data type and data instance definitions, respectively. The data concepts in TDL are abstract symbols that can be related by means of data element mappings to concrete data representations stored in external resources, such as TTCN-3 or Extensible Markup Language (XML) documents, which are referenced by means of data resource mappings. The defined data elements can be referenced in various contexts, such as parameters, interactions, timers, etc. by means of data use concepts. The data use concepts also include wildcards such as any value of a given type.

The *Test Configuration* package defines the concepts related to the test architectures for TDL test descriptions. 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 their gate instances. A component instance inherits the gate instances, timers and variables of the component type it conforms to. A gate instance conforms to a gate type which specifies the data that can be used in interactions over gate instances of that gate type.

The *Test Behaviour* package defines the concepts necessary for the specification of behaviour. These include atomic behaviours, such as interactions for exchanging data between gates of component instances, local or global actions, references to other test descriptions, as well as explicit verdict assignments. Compound behaviours, including conditional, alternative, repeated, interrupt, and default behaviour, are used to group atomic behaviours.

The *Time* package in TDL contains concepts for the specification time operations, time constraints, and timers. 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*. Timer operations (*Start*, *Stop*, *Timeout*) operate on component instance timers.

Finally, *Annex B* of Part 1 contains an example text-based notation for illustrative purposes in order to showcase some concrete examples of the use of the TDL MM. It can serve as the basis for a textual concrete syntax for TDL, however, the notation itself is only informative at this point in time.

2.3 The Graphical Representation

Part 2 of the TDL standard defines the default, general-purpose graphical representation format for TDL meta-model elements. One of the key design requirements was that the graphical representation of TDL shall resemble the graphical format of the most frequently used modeling notations in order to preserve familiarity and ensure that it is easy to learn for users. Considering that Unified Modeling Language (UML) is widely used in the industry and also gaining momentum in standardisation, the graphical representation of TDL was aligned with UML to the extent to which the corresponding TDL elements have (almost) direct equivalents in UML. Elements of the TDL meta-model that have no direct equivalent or have different semantics are represented in a different way in order to avoid confusion.

The specification of a TDL test description typically requires three major parts—data and data type specification, test configuration specification, test behaviour specification. An example of a data type and a data instance specification is shown in the bottom part of Fig. 1. In this case, the definition of a data type `Message` and a data instance `Request` of data type `Message` is illustrated. In TDL, the type definition symbols have double borders. As noted in Sect. 2.2, TDL only provides abstract symbols for data that can be mapped to external data. A data mapping specification is shown in the upper part of Fig. 1. In this example, it is assumed that there is a file `data.ttcn3` containing some concrete data specifications in TTCN-3, which is made accessible as a mapping target in TDL by means of a data resource mapping. The data element mappings then specify how the abstract data type and data instance symbols specified in TDL are related to the corresponding concrete data specifications. In the example shown in Fig. 1, the `Request` data instance is mapped to a template `m_rq` defined in `data.ttcn3`. Similarly, the abstract `Message` data type defined in TDL is mapped to the concrete `t_msg` data type defined in `data.ttcn3`. The resolution and validation of the correctness and consistency of such mappings is left to tool implementations.

The definition of a test configuration is illustrated in Fig. 2. In order to define a test configuration, first one or more gate types need to be defined. The gate type `defaultGT` which can send and receive data instances of type `Message` is shown in the top left part of Fig. 2. Next, one or more component types need to be defined. The component type `defaultCT` which has a gate `g` of type `defaultGT`, as well as a variable `v` of type `Message`, is shown in the top center part of Fig. 2. Finally, a simple test configuration `defaultTC` containing two component instances of type `defaultCT` (`SS` in the role `TESTER` and `UE` in the role `SUT`), as well as a connection between their gates, is shown in the bottom part of Fig. 2.

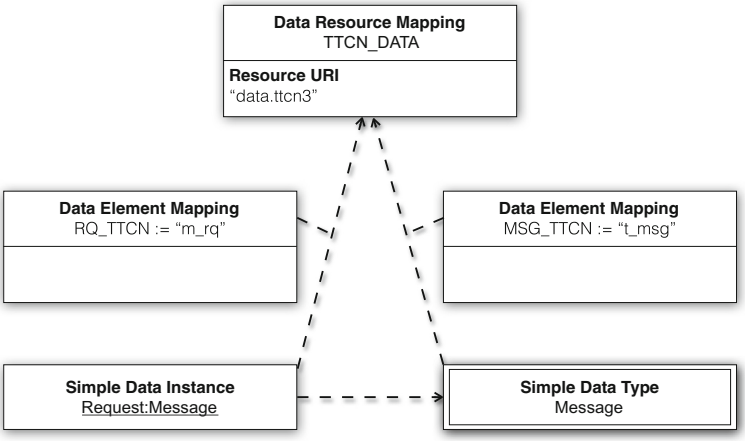

Fig. 1. Data definition and mapping specification

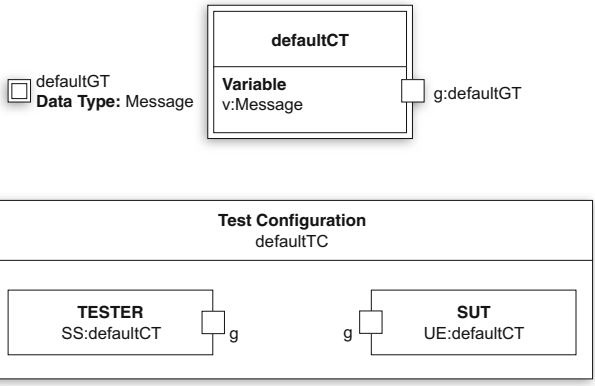

Fig. 2. Gate type, component type, and test configuration

The specification of behaviour is exemplified in Fig. 3. The example features a simple interaction sequence between the previously defined component instances. The `SS` in the role of `TESTER` sends a `Request` message. The `UE` in the role of `SUT` may respond to the `SS` by either an `Accept` or a `Reject` message. The verdict in the first case will be `PASS`, while in the second case it will be `FAIL`.

2.4 The Exchange Format

Part 3 of the TDL standard defines the exchange format for TDL models. The exchange format describes the rules for serialization and de-serialisation of TDL models which enables interchange of TDL models between tools. The TDL XF is based on the XML Metadata Interchange (XMI) specification [34]. The TDL XF

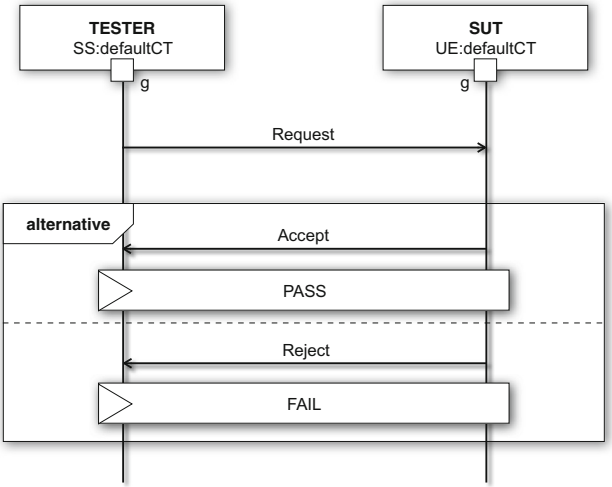

Fig. 3. Behaviour specification

specifies production rules for deriving a TDL XMI Schema covering the complete definition of the TDL MM, including the MM extensions defined in the TDL TO, as well as production rules for TDL XMI documents for the serialization of TDL models. However, the semantic correctness of a TDL model represented in the form of an XMI document cannot be validated based on the exchange format rules alone. After de-serialisation, the resulting TDL model still needs to be checked with respect to the rules defined in the MM.

2.5 Extending TDL: Structured Test Objectives

Part 4 of the TDL standard defines an extension of TDL to support the specification of structured test objectives in a formalised manner within TDL. In Part 1 of TDL, the description of a test objective can be specified only as informal text. In Part 4, additional concepts are introduced as extensions to the meta-model in order to capture additional features of test objectives. These include:

- 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. An example is shown in Fig. 4. While the notation for the structured test

TP Id	TP_1
Test Objective	Ensure that a request is accepted
Reference	SAM 2016 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. 4. Structured test objective specification

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. Similar to Part 1, Part 4 contains an informative text-based notation in its *Annex B* focusing on the concepts introduced as extensions to the meta-model.

2.6 The UML Profile for TDL

To enable the application of TDL in UML based working environments, a UML Profile for TDL (UP4TDL) was developed. 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*. A stereotype enables the extension of a UML meta-class with additional properties, relations, or constraints in order to address domain-specific concerns.

The overall architecture of UP4TDL is similar to the package structure of TDL. The top-level package of the UP4TDL contains stereotypes that enable binding annotations to any TDL elements. The test objective concept is also declared there. For structuring TDL models in packages and importing TDL elements from other packages, the UP4TDL relies on already existing concepts in UML (*UML::Package* and *UML::ElementImport* respectively). The *DataDefinition* package contains the equivalents of the concepts from the *TDL:Data* package, most of which are already present in UML. 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. The *DataUse* package contains an almost independent metamodel that is used as the basis for integrating data-related expressions in TDL models. These expressions are used in the specification of arguments of interactions. The *Behaviour* package contains two kinds of stereotypes. A set of stereotypes that extend *UML::CombinedFragment* represents the set of elements defined in *TDL::CombinedBehaviour*. The other stereotypes mostly extend *UML::OccurenceSpecification* and represent corresponding elements defined in *TDL::AtomicBehaviour*. Finally, the *Time* package contains stereotypes representing elements related to the management of time in TDL. Currently, they are specific to TDL. In future work, linking these concepts with the Modeling and Analysis of Real-Time Embedded Systems (MARTE) profile [29] may help to refine the embedding of UP4TDL into the UML environment.

The UML Testing Profile (UTP) [31] is published by the Object Management Group (OMG) and is also providing 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. 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 also follows some 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 (e.g. *ExceptionalBehaviour*, *PeriodicBehaviour*).

In TDL, a global synchronization mechanism is assumed. Consider the two communication events shown in Fig. 5. In UTP, these communication events are represented by means of *UML::Message*. Without additional constraints (such as *GlobalOrdering*) they are only locally ordered. Consequently, the following event sequence <*`Tunnel` sent, `Request` sent, `Request` received, `Tunnel` received*> would be considered valid. In TDL, such a communication sequence is

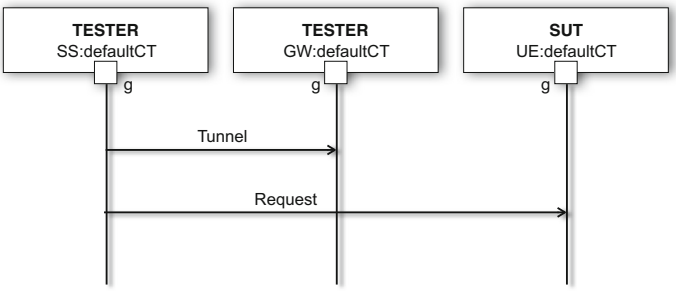

Fig. 5. Communication event ordering

represented by means of *TDL::Interactions* which are assumed to be atomic (no distinction between the send event and the receive event) and globally ordered. Consequently, the only acceptable event sequence would be <*Tunnel occurred, Request occurred*>. This simplification choice is compatible with the way test cases are implemented in many test management tools.

3 An Implementation of TDL

A reference 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 reference 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.

3.1 Requirements

A basic TDL implementation shall be able to read and write models serialized according to the XF. It shall also allow users to check static semantics of models to verify their compliance with TDL. The specific means of creating and editing TDL models is beyond the scope of the standard. TDL users may apply domain-specific textual notations mapped to the TDL MM, such as the notation described in *Annex B* of Part 1. Users may also utilize existing general purpose modeling tools or other customized combined notations. The TDL MM specifies constraints on the elements of the language, which are formalised by means of OCL expressions. In order to evaluate these expressions against TDL models, they also need to be integrated in the reference implementation.

One of the use cases for TDL from the very start was visualising tests generated from MBT tools as well as visualising test execution traces. The reference implementation focused on providing such facilities by implementing means for the visualisation of TDL models according to the TDL GR as the graphical representations are also better suited for use in (standards-related) documents and during high-level discussions, as well as for instructional purposes.

Due to the peculiarities and intended use of structured test objectives, it was determined that instead of graphical shapes that can be exported as images, the graphical representations of structured test objectives shall be realised as tables exported in a Word document according to user-defined templates, which can then be manipulated further in order to fit in within an existing document.

For the use of the UP4TDL, the profile itself needed to be implemented first. However, using the facilities provided within modeling environments for manipulating features provided by UML profiles can be rather cumbersome. Hence, customised editing facilities for UML models using the UP4TDL were also necessary in order 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 reference implementation of TDL.

3.2 Realisation

Eclipse and associated technologies were chosen as the base platform for the implementation as it is the most widely used modeling platform today. At the core of Eclipse modeling platform is the Eclipse Modeling Framework (EMF) which provides an implementation of MOF (named *Ecore*) which was used to for the implementation of the TDL meta-model. The meta-model specification in the form of a UML model was done in the Papyrus modeling environment [2]. The resulting UML meta-model for TDL was transformed into an Ecore meta-model for implementation purposes. 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 can be integrated into the meta-model by means of annotations, which can be used for automated validation of model instances. However, any modifications to the constraints would require changing the meta-model and related generated resources. Alternatively, the constraints can be specified as an add-on which can be applied to the model instances as needed. This allows add-on constraints to be modified, maintained, and extended independently from the meta-model. The Epsilon Validation Language (EVL) [27] provides means for the implementation of add-on constraints. The EVL also extends the capabilities of OCL by providing additional facilities for the specification of guards on constraints which are used to specify under which conditions the evaluation of a constraint shall be skipped.

The Eclipse Graphical Modeling Framework (GMF) links EMF modeling 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 [4] utilizes 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 as the application matures. In addition, while the reference implementation focuses on the visualisation of TDL models, facilities provided by the Sirius platform can be extended with the specification of editing capabilities thus enabling an easy transition from a viewer to an editor in the future.

The labels related to data use in the 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 [6] is a framework for developing textual languages on top of EMF. Xtext was also used for the realisation of the syntax specified in the informative *Annex B* of Part 1 as part of the reference implementation enabling the quick creation and manipulation of TDL models. Apart from the grammar specification, it also includes further customisations in the scoping and linking facilities, as well as enhanced semantic syntax highlighting which provides customisable styles for identifiers based on their type and usage context. Similarly, a Xtext-based realisation of the syntax specified in the informative

Annex B of Part 4 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 Docx4j [1] library providing Application Programming Interface (API) for manipulating Word documents. A set of predefined templates, including a template according to the concrete syntax defined in Part 4, are included in the implementation. Users may define additional templates. The templates describe the overall structure of the representation and contain a set of placeholders for the different labels. Xtext is then used to substitute the label placeholders with serialisation of the corresponding model elements and their contents, in a similar manner as the labels in the GR viewer.

The implementation of the UP4TDL as well as supporting editing facilities was realised on top of Papyrus. Papyrus provides a graphical editor and an extensible framework for working with UML models. It was customised in order to allow users to create graphical representations of UML models applying UP4TDL with dedicated editing facilities. The customisations are shared among three new kinds of diagrams—the *DataDefinition* diagram, the *TestDescription* diagram, and the *TestConfiguration* diagram. Using the TDL *TestConfiguration* diagram, the user can declare component types, component instances, gates, and connections. To specify the types of data that are exchanged through gates, the user may need to open a *DataDefinition* diagram to specify the data types and data instances. In the *DataDefinition* diagram, the user can also map these data elements to a data resource containing concrete data representations. Finally, the user can start creating test descriptions that use the specified test configuration by means of the *TestDescription* diagram. Additional editing facilities are provided to streamline the work with UML models applying the UP4TDL. These include a customised property editor that shows only the TDL-related properties of stereotypes and an Xtext-based editor for the arguments of interactions.

The public availability of the implementation will be announced on the TDL website [5], once licensing questions have been addressed at ETSI.

4 Related Work

Domain-specific testing languages 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.

The Check Case Definition Language (CCDL) [32] provides a high-level approach for requirements-based black-box system level testing embedded in its own testing process. Test simulations and expected results specified in human readable form in CCDL can be compiled into executable test scripts. However, due to lack of standardisation, high-level test descriptions in CCDL are heavily tool-dependent. High-level keyword-based test languages, using frameworks such as

the Robot Framework [3], have also been integrated with MBT [35]. Beyond textual keyword-based languages, graphical domain-specific testing languages, such as one built on top of TTCN-3 [26], have also been developed.

There have been efforts to address the lack of standardisation in some application domains, such as the standardised meta-model for testing avionics embedded systems [19], and the Automotive Test Exchange Format (ATX) [7] and TestML [18] focusing on automotive systems. Additionally, the Open Test Sequence Exchange Format (OTX) [22–24] standardised at the International Organization for Standardization (ISO) aims to provide tool-independent XML-based data exchange format for the formal description and documentation of executable test sequences for 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) [21] standardised at the International Telecommunication Union (ITU) was one of the first languages for the specification of scenarios, not focusing strictly on testing. In addition to the main specification, Annex B [20] provides a formal specification of the semantics of MSC. Some of the features of MSC were subsequently adopted in OMG's UML in the form of *Sequence Diagram*. 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 [28] are a limiting factor for its use as a universal and consistent test description language.

The UTP [31] adds domain-specifics concepts related to testing thus enabling test modeling with UML. While it maintains the wide scope of UML, it also inherits the disadvantages associated with UML. The UP4TDL transfers the concepts of TDL in the UML world providing a more specialised and integrated means for test modeling with UML and inheriting the semantics from TDL.

The Precise UML [8] introduces a subset of UML and OCL for MBT seeking to address the open-ended interpretations of the semantics of different diagrams focusing on the specification of a behavioural model of the SUT. Its scope is narrowed down the generation of test cases out of an SUT model.

Approaches based on a concrete executable language with strict semantics, such as TTCN-3, enable the exchange of executable tests between partners. However, such languages are not well suited for review and high-level discussions due to the level of detail and the need to be able to understand the programming language-like syntax.

While various domain-specific testing languages have been developed, they all share a common set of challenges, including imprecise, informal, or no semantics, lack of standardisation, lack of comprehensive tool support, and poor interoperability with other tools. A common standardised meta-model for testing with associated well-defined semantics can help in consolidating approaches and providing a solid foundation for interoperability between tools.

5 Conclusion

TDL has been designed to address existing challenges and streamline the test development process. By abstracting away from 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 a specific test execution framework. It can also make test specifications easier to review by non-testing experts. This is beneficial for the overall productivity and quality of test development both in industry and in the standardisation process.

In this article, we discussed the evolution of TDL from its conception into a multipart standard including a meta-model, standardised graphical syntax, exchange format, and a UML profile. To accelerate the adoption of TDL a reference implementation is provided both for users to get started with using TDL and for tool vendors to build their own solutions on top of it or integrate it with their existing products. To bring test purpose specifications into the modeling world and streamline the test specification process even further, an extension to TDL was developed to enable the specification of structured test objectives within TDL. The extension is based on TPLan and is targeted particularly towards supporting the standardised test development processes at ETSI.

The road ahead for TDL includes two main directions for the near future—mapping TDL to TTCN-3 and investigating the requirements for the adaptation of TDL to specific testing needs, such as security and performance testing. Mapping of TDL test descriptions to TTCN-3 test cases will enable generating of executable tests from TDL test descriptions in a semi-automatic way and allow re-using of existing TTCN-3 test tools and frameworks for test execution. A standardised mapping will leverage the impact of TDL and ensure that there is a consistent and common way of implementing test cases based on TDL test descriptions. This will significantly increase the efficiency of the test creation process and thus decrease cost and time-to-market of software products.

During the launch event at UCAAT 2015, multiple stakeholders expressed interest in adopting TDL for security and performance testing. However, they also requested certain features that they rely on. In addition, there are ongoing activities on security testing within TC-MTS. A next step is to identify and catalogue the requirements for the adaptation of TDL to different domains and types of testing in order to determine new language features and extensions.

The TDL working group within TC-MTS is dedicated to maintaining and updating TDL according to evolving user needs. With contributions from numerous partners from industry, academia, and standardisation, TDL is aiming to address testing needs from a wide spectrum of users across different domains.

Acknowledgement. The work on TDL has been funded by ETSI in the context of the STF projects 454, 476, and 492.

References

1. Docx4j. http://www.docx4java.org. Accessed 25 June 2016
2. Papyrus. https://www.eclipse.org/papyrus/. Accessed 20 June 2016
3. Robot framework. https://robotframework.org. Accessed 20 June 2016
4. Sirius. https://www.eclipse.org/sirius/. Accessed 20 June 2016
5. Tdl. http://tdl.etsi.org. Accessed 20 June 2016
6. Xtext. https://eclipse.org/Xtext/. Accessed 20 June 2016
7. Association for Standardisation of Automation, Measuring Systems (ASAM): Release Presentation: ASAM AE ATX V1.0.0, Automotive Test Exchange Format, July 2012. http://www.asam.net/nc/home/asam-standards.html
8. Bouquet, F., Grandpierre, C., Legeard, B., Peureux, F., Vacelet, N., Utting, M.: A subset of precise uml for model-based testing. In: Proceedings of the 3rd International Workshop on Advances in Model-based Testing, A-MOST 2007, pp. 95–104. ACM, New York (2007)
9. ETSI EG 203 130: 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, April 2013
10. ETSI ES 201 873–1: 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, July 2016
11. ETSI ES 202 553: Methods for Testing and Specification (MTS); TPLan: A notation for expressing Test Purposes, v1.2.1. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, June 2009
12. ETSI ES 202 951: 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, July 2011
13. ETSI ES 203 119–1: Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 1: Abstract Syntax and Associated Semantics, v1.3.0. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, July 2016
14. ETSI ES 203 119–2: Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 2: Graphical Syntax, v1.2.0. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, July 2016
15. ETSI ES 203 119–3: Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 3: Exchange Format, v1.2.0. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, July 2016
16. ETSI ES 203 119–4: Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 4: Structured Test Objective Specification (Extension), v1.2.0. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France, July 2016
17. ETSI ES 102 840: 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, February 2011
18. Grossmann, J., Müller, W.: A formal behavioral semantics for TestML. In: Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2006, pp. 441–448, November 2006
19. Guduvan, A., Waeselynck, H., Wiels, V., Durrieu, G., Fusero, Y., Schieber, M.: A meta-model for tests of avionics embedded systems. In: MODELSWARD 2013, Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development, Barcelona, Spain, 19–21 February 2013, pp. 5–13 (2013)

20. International Telecommunication Union (ITU): Recommendation Z.120 Annex B: Formal Semantics of Message Sequence Chart (MSC), 04/98. Online: Z.120 Annex B (04/98), Standard document. URL: http://www.itu.int/rec/T-REC-Z.120-199804-I!AnnB/en
21. International Telecommunication Union (ITU): Recommendation Z.120: Message Sequence Chart (MSC), 02/11. Online: Z.120 (02/11), Standard document. URL: http://www.itu.int/rec/T-REC-Z.120-201102-I/en
22. ISO: Road vehicles - Open Test sequence eXchange format - Part 1: General information and use cases. International ISO multipart standard No. 13209–1 (2011)
23. ISO: Road vehicles - Open Test sequence eXchange format - Part 2: Core data model specification and requirements. International ISO multipart standard No. 13209–2 (2012)
24. ISO: Road vehicles - Open Test sequence eXchange format - Part 3: Standard extensions and requirements. International ISO multipart standard No. 13209–3 (2012)
25. ISO/IEC: Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General Concepts. International ISO/IEC multipart standard No. 9646–1 (1994–1998)
26. Kanstrén, T., Puolitaival, O.P., Rytky, V.M., Saarela, A., Keränen, J.S.: Experiences in setting up domain-specific model-based testing. In: 2012 IEEE International Conference on Industrial Technology (ICIT), pp. 319–324, March 2012
27. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: On the evolution of OCL for capturing structural constraints in modelling languages. In: Abrial, J.-R., Glässer, U. (eds.) Rigorous Methods for Software Construction and Analysis. LNCS, vol. 5115, pp. 204–218. Springer, Heidelberg (2009). doi:10.1007/978-3-642-11447-2_13
28. Micskei, Z., Waeselynck, H.: The many meanings of UML 2 Sequence diagrams: a survey. Softw. Syst. Model. **10**(4), 489–514 (2010)
29. Object Management Group OMG: UML Profile For MARTE: Modeling and Analysis of Real-Time Embedded Systems, Version 1.1. OMG Document Number: formal/2011-06-02, Standard document, June 2011. URL: http://www.omg.org/spec/MARTE/1.1/
30. Object Management Group OMG: Object Constraint Language, Version 2.3.1. OMG Document Number: formal/2012-05-09, Standard document, May 2012. URL: http://www.omg.org/spec/OCL/2.3.1/
31. Object Management Group OMG: UML Testing Profile (UTP), Version 1.2. OMG Document Number: formal/2013-04-03, Standard document, April 2013. URL: http://www.omg.org/spec/UTP/1.2/
32. Object Management Group OMG: CCDL Whitepaper. Razorcat Technical Report, 23 January 2014, January 2014. http://www.razorcat.eu/PDF/Razorcat_Technical_Report_CCDL_Whitepaper_02.pdf
33. Object Management Group OMG: Meta Object Facility Core, Version 2.4.2. OMG Document Number: formal/2014-04-05, Standard document, April 2014. URL: http://www.omg.org/spec/MOF/2.4.2/
34. Object Management Group OMG: XML Metadata Interchange (XMI), Version 2.4.2. OMG Document Number: formal/2014-04-06, Standard document. URL: http://www.omg.org/spec/XMI/2.5.1/
35. Pajunen, T., Takala, T., Katara, M.: Model-based testing with a general purpose keyword-driven test automation framework. In: ICSTW, pp. 242–251. IEEE, March 2011