|  |
| --- |
|  |
| **Title\*:** | Response to Ericsson's comment on TDL v2 |
|  |  |
| from **Source**\*: | Siemens AG |
| Contact: | Andreas Ulrich  |
|  |  |
| input for **Committee**\***:** | MTS |
|  |  |
| Contribution **For\*:** | Decision | **X** |  |
|  | Discussion |  |  |
|  | Information |  |  |
|  |  |
| Submission date**\***: | 2015-01-27 |
|  |  |
| Meeting & Allocation: | **MTS#64** -  |
| Relevant WI(s), or deliverable(s): |  RES/MTS-203119-1v1.3.1 |
|  |

**Decision requested:** Please consider this document together with MTS(15)64\_020 and approve the proposed actions as suggested in the summary section.

**ABSTRACT:** *This document is a review of Ericsson's submission "MTS(15)64\_020" on the raised issues mainly related to TDL Part 1 v1.2.1 in the revision "MTS(15)64\_003r3" and the proposed changes on the TDL-MM.*

# Summary

This document is a review of Ericsson's submission "MTS(15)64\_020" on the raised issues mainly related to TDL Part 1 v1.2.1 in the revision "MTS(15)64\_003r3", in short: TDL-MM, and the proposed changes on the TDL-MM. Ericsson raises two requirements that they consider not covered in the current TDL-MM, which are: 1) Partial test data (message) definition and 2) Support for optional message fields.

This document shows that the raised requirements are indeed covered in TDL-MM and that Ericsson's objection is therefore pointless. Furthermore there appears to be no reason to implement the proposed changes on the TDL-MM. Instead it appears that the objection stems probably from wrong presumptions and misreading of the related documents.

Ericsson provides no reasoning how the proposed changes attached to their submission would help solve the raised requirements. A review of the changes discloses that they cause further consequences that do not improve the quality of the existing TDL-MM. One of them is that the proposed changes would fundamentally change the design approach for TDL from a self-contained, standalone specification language to an extension of TTCN-3. While it is possible to go the latter way, a broader discussion within all stakeholders within MTS would be required to decide for it.

It is suggested that Ericsson reconsiders their objection taking the feedback provided in this document into account, corrects their wrong statements and resubmits a revised version with the goal to come to an agreement about a solution for their raised requirements. In particular, Ericsson needs to discuss: a) why the proposed DataElementMapping concept does not solve requirement 1, b) why a data flow analysis in combination with an explicit assignment of NoneValue to an optional member does not solve requirement 2. Additionally Ericsson shall provide c) a reasoning how their proposed solution solves the addressed requirements and the consequences they inflict on the remaining TDL-MM.

In the meantime, the old STF 476 will provide an updated version of TDL-MM that tries to clarify some statements made on the semantics of DataUse and Interaction, but otherwise keeps the current design, which demands a complete DataInstance specification and asks for an explicit use of NoneValue to express an omitted optional member, unless Ericsson shows sufficient evidence of a shortcoming.

# Introduction

The remainder of the document discusses mainly the received input from Ericsson by citing the relevant text parts. Citations are marked as indentions. In addition and first, it briefly introduces the basic principles of meta-modelling and how they are covered in the standard series of TDL as some misunderstandings are observed from reading Ericsson's comment.

# Basic principles of meta-modelling and mapping to document structure

## Basic principles of meta-modelling

A software language consists of two independent parts: syntax and semantics. The syntax describes the structure of a text provided in this language (or of graphical symbols for a graphical language), while the semantics describes the meaning of individual syntax elements used in the text.

With upcoming multi-representation syntaxes for a single language, the distinction between abstract syntax and concrete syntax becomes necessary. The abstract syntax defines the structure of the text without referencing keywords that are provided in a concrete syntax only.

A text given in one concrete syntax can be transformed into a text of another concrete syntax if the text is syntactical correct w.r.t. concrete syntax and abstract syntax and both concrete syntaxes refer to the same abstract syntax. This transformation is completely independent from the semantics definition (which does not change).

Semantics is divided into static semantics and dynamic semantics. The static semantics defines further restrictions on the structure of the text that cannot be defined alone in syntax rules. The dynamic semantics provide the meaning of a syntax element when it is put into an execution environment.

An editor provides the possibility of checking the concrete syntax of a text of a given language, typically already when the text is entered, and stores the text in its abstract syntax for further processing. It may optionally check also the static semantics to provide early feedback to a user. A compiler relies on a syntactical correct text as input and considers the static and dynamic semantics of the syntax elements when generating executable code.

## Mapping to document structure

The four pieces of a software language, concrete syntax, abstract syntax, static semantics, dynamic semantics, are mapped to the standards series of TDL as follows:

* Concrete syntax: defined in parts 2 on TDL-GR (graphical format) and 3 on TDL-XF (exchange format)
* Abstract syntax: defined in part 1 on TDL-MM in the contained diagrams representing the technical meta-model, which correspond to the individual sub-clauses "Generalization" and "Properties".
* Static semantics: defined in part 1 on TDL-MM in the sub-clauses "Constraints" and partly, if constraints are hard to formulate, in the sub-clauses "Semantics".
* Dynamic semantics: defined in part 1 on TDL-MM in the sub-clauses "Semantics". To highlight the dynamic semantics, sometimes the expression "at runtime" is used.

In addition part 4 on TDL-TO (test objective specification) defines an extension of the abstract syntax and semantics of part 1 together with a concrete textual syntax for this extension.

While the syntax is formally defined, the semantics is defined only informally and frequently together with further explanations beyond pure semantics definition to ease reading of the document.

Building an editor requires basically knowledge about the concrete and abstract syntax and the static semantics as defined under the "Constraints" sub-clauses. Building a converter from one representation format into another one requires knowledge about the two concrete syntaxes and the abstract syntax. Building a compiler requires knowledge about the abstract syntax and the static and dynamic semantics.

# Response to Ericsson's claims

## Requirement: Partial test data (message) definition

Section 2.1.2 "Consequences" summarizes the consequences from Ericsson's point of view as follows:

The user SHALL assign a value or a wildcard value to ALL FIELDS of ALL MESSAGES before they are used in an interaction.

This statement is correct and intentional as it ensures that a TDL spec can be checked to be self-contained and fully specified. If a user can decide ad libido whether a member of a data instance can be specified or kept unspecified, there is no check possible to analyze whether a TDL spec is complete.

Therefore, the complete message definition requirement of the current TDL draft is simply impossible to fulfil in many of its real use scenarios.

This final conclusion is given without sufficient evidence as it misses to discuss the available data mapping concept. Ericsson shall provide an update and discuss why the available concept in insufficient for their purposes.

The need for the data mapping concept is obvious because it is the means to define abstraction for the data part, while the test description concept defines the abstraction for the behaviour part.

## Requirement: Support of optional message fields

At some point in the STF work, the decision to support optional members was taken, which is reflected in the TDL-MM draft.

Again, I concentrate on the claims under the section 2.2.2 "Consequences":

The semantic meaning of NoneValue is clearly defined as ‘undefined’.

This statement is WRONG. The semantics of NoneValue in TDL is defined in its semantics sub-clause of 6.3.8 as follows: "A 'NoneValue' denotes a symbolic value that represents no value at all." This definition is very different from 'undefined'.

The cited statement from the semantics part of an interaction in clause 9.4.6 that is given as a counterargument: "The occurrence of 'NoneValue' other than for optional 'Member's of a 'StructuredDataInstance' causes undefined semantics of the 'Interaction' behaviour at runtime." refers to the semantics of an Interaction, but not to the semantics of NoneValue. The sentence clearly states this.

a) The note in $6.3.8 contradicts to the mandatory text in clauses 6.2.11 and 6.3.8, which define 'NoneValue' in the meaning of ‘undefined’.

As said, NoneValue has the only meaning as specified in its semantics sub-clause (without note). The note gives a hint to a user of how the syntactic element NoneValue can be used in a TDL spec. Otherwise, i.e. if NoneValue does not occur as a representation in a concrete syntax, e.g. as a keyword, it does not need to be defined as an abstract syntax element and, hence, there would be no need for this meta-class element. The note can be completely skipped to understand the semantics of NoneValue as it is only explanatory text on the usage of the NoneValue syntax element.

a.a) there is no requirement and property to store if 'NoneValue'has been assigned implicitly or explicitly

The constraint "Use of a 'StructuredDataInstance' with non-optional 'Member's" under "6.3.1 DataUse" states that NoneValue can be assigned to optional members explicitly. Together with the assumption of completely specified data instances, the existence of such an assignment can be checked at compile time by performing a classical data flow analysis that validates the correct definition-use paths of variables and structured data instances.

It is very similar to a data flow analysis performed in programming languages. Consider the following C-code fragment: int x; if (x == 0) do\_something(). A good compiler raises a warning that variable x is used in the condition without being defined. A similar check can be implemented for TDL. There will be no difference whether the data flow analysis in TDL will be performed on assignments of NoneValue or yet another value, say OmitValue. The task remains the same.

a.b) if the specification is stored in XF and then read by a tool, there is no way to know what was the semantic meaning of 'NoneValue' in the original TDL specification

This argument is plain WRONG. As stated in "Basic principles of meta-modelling" above, a TDL spec, say in a concrete textual syntax, can be converted to TDL-XF if it is syntactical correct. The ability to convert or store a TDL spec in TDL-XF is completely independent from the semantics definition of the syntactical elements. Semantics refer here to both parts: static and dynamic semantics. In particular, neither an editor nor a compiler modifies a provided TDL spec, but treats it as it is.

b) … there is no formal specification of how the user can identify optional ‘Member’s that shall not – by the test specification - be present in an interaction.

The provided solution relies on the explicit assignment of NoneValue to an optional member. As said above, the existence of this explicit assignment can be checked in the definition-use path of the data instance used in an interaction. The assumption of completeness of a TDL spec (see above) ensures that no undefined data instances and variables can escape via an interaction. Ericsson shall provide reasoning why they consider this approach as insufficient.

c) … Consequently, the tools shall distinguish between NoneValue assigned implicitly and explicitly for semantic checking purposes. However, no flag is defined for this purpose in the MM draft … Thus, this information is lost as soon as the TDL spec. is saved or transferred to another tool

The provided solution does not rely on different syntactical elements, but a data flow analysis which is the standard means for this kind of problem. Moreover adding another syntax element, say OmitValue, would not be a solution because NoneValue can be only assigned to optional members. An OmitValue would be defined with the same meaning, which implies that one of the syntax elements is redundant.

The latter statement "this information is lost…" is WRONG. An explicit assignment of NoneValue will remain present in a TDL spec. An editor does not modify a given TDL spec on its own. See discussion above.

d) Backward incompatible: The current way of using NoneValue in the MM guarantees incompatibility between different versions of the language

This is all guessing at the moment. Why will there be a need to modify the semantics of NoneValue in the future? All I can say now is that the provided solution in TDL-MM is sound under the given assumptions.

# Review of the proposed solution

Ericsson proposes a list of changes on the current TDL-MM, but provides no discussion on how these changes address the issues they claim to have identified. It is therefore requested that Ericsson provides this discussion in an updated version. I tried to comprehend the suggested changes as discussed below. Ericsson may comment on this interpretation. But before going too much into these details, I suggest to concentrate on working out the shortcomings, if any, of the existing TDL-MM.

Inspection of the list of changes reveals the following major modifications:

* Generalization of NoneValue which can be used without any constraints in DataUse specifications and semantics redefinition to an undefined value;
* Introduction of a new OmitValue syntax element whose usage is constraint in a similar way as the usage of NoneValue in the current TDL-MM;
* Redefinition of the semantics of Interaction to enable the use of NoneValue in parallel with OmitValue.

The generalization of NoneValue means that it can be used in any assignments of members or variables explicitly by a user. The given semantics of NoneValue to denote "a symbolic value that is not specified on TDL level" makes it however very close to the semantics of AnyValue, which states: "An 'AnyValue' denotes an undefined symbolic value from the set of all possible values of a 'DataType'…". The only difference is that NoneValue is assumed to be the default value for members and variables. A similar result could be obtained if AnyValue is declared to be the default value. A user would be confused to decide to use NoneValue or AnyValue in assignments. On current grounds, this definition of NoneValue is hardly acceptable.

A new syntax element OmitValue is introduced which becomes now a necessity because OmitValue and NoneValue can appear side-by-side in a DataUse of an Interaction. It is a side-effect of the generalization of NoneValue. As discussed above, without the generalization of NoneValue the introduction of OmitValue would be redundant. Furthermore this new OmitValue does not improve the correctness of a TDL spec by avoiding mistakes due to missing explicit assignments: A missing OmitValue cannot be detected at compile time, because the default value NoneValue for an optional member is equally correct. In this respect the proposed change is worse compared to the existing solution in the TDL-MM.

Allowing a NoneValue to be an accepted value for any member of a data instance in an interaction (not just for optional members as defined before) has the following consequence. In a given test description a user can decide ad libido whether he wants to assign a value to a member of a structured data instance, i.e. field of a message. For example, in the first message the session-id is set, in the second message it is not, in the third it is set again. Because there is no rule defined that governs the need for setting values, a TDL compiler cannot check whether a TDL spec is correct on its own. That is, the semantics of a TDL spec can be only understood together with the code of the test harness that adds the missing pieces of a message. This test harness needs to have the intelligence to set the right value in the right context of the test description (in the example: shall in case of the missing session-id in the second message the old id be used or a fresh value?). This approach implies further that it will be hard to develop TDL spec and test harness independently from each other and TDL becomes eventually an extension of TTCN-3.

Ericsson does not provide any clue whether this development of TDL towards a TTCN-3 extension is intended or not. At least it will mean a departure from the current way how TDL is designed. Here a statement is clearly necessary.