Input to TDL SG discussion (updated)

Andreas Ulrich, Siemens AG

8.2.2015

This revision of my previous input became necessary because I recognized that an intermediate version of the TDL MM draft as basis for discussion is used, which does not contain latest changes.

# On the use of verbal forms for the expression of provisions in the MM draft

ETSI requires the use of 'shall' and 'shall not' (among others) to "indicate requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted." (see "ETSI Drafting Rules", clause 14a)

It was observed during the last meeting that the semantics section of a clause in the MM draft does not use 'shall' to formulate requirements in the semantics sections.

I provide the following explanation for this case: The semantics is given in a **declarative style**. This is true in particular for the data part but also to a large extend for the behaviour part. It is therefore completely useless and actually wrong usage to enrich declarative statements with 'shall'. A declaration is not a requirement, but a **statement of fact**. In this case, the EDR requires that 'is / is not' "shall be used to indicate statements of fact". This rule is followed in the MM draft.

Example: clause 6.3.1 DataUse

"A 'DataUse' denotes an expression that evaluates to a 'DataInstance' of a given 'DataType'."

It would be a wrong style of writing "A 'DataUse' shall denote…" This would overburden the sentence. The use of 'shall' is commonly associated with verbs of action, e.g. "a user shall do this", "a protocol entity shall behave like this".

Therefore, the MM draft makes use of 'shall' when a user could have a choice which needs to be restricted. Such cases typically occur within the constraints sections because here the options on the syntax a user has are clearly restricted.

Counterexamples might exist in the MM draft, but do not invalidate the above reasoning.

# On "<undefined> values causes undefined semantics" in clause 9.4.6 Interaction

As outcome from last discussion it was agreed to add a note. The text will now read as follows:

The occurrence of the <undefined> value within the 'DataUse' specification of 'argument' causes undefined semantics of the 'Interaction' behaviour at runtime.

NOTE: How the <undefined> value within the 'DataUse' specification of 'argument' is resolved is outside of the scope of this document.

# On semantics of OmitValue

Ericsson's proposal is flawed and therefore completely unacceptable and needs to be reverted.

First we have to obey to the rules of meta-modelling which say (adapted to TDL):

* Concrete syntax is defined in parts 2, 3;
* Abstract syntax is defined in part 1 in the diagrams;
* Static semantics is defined in part 1, Constraints;
* Execution semantics is defined in part 1, Semantics.

It is common practice in the definition of a computer language to explicitly separate these 4 ingredients. Any expert in meta-modelling will recognize them and treat them accordingly. Moreover, the MM draft is structured similarly to UML documents and those documents are listed as normative references. UML uses the same approach, which TDL borrows. Therefore anyone literate in UML design can easily follow the TDL design without giving further explanations about the meaning of the different sections of a clause.

**Now on the clause 6.3.8 OmitValue from MTS(15)000014 which reads as follows:**

Semantics

An 'OmitValue' denotes a symbolic value that represents an optional 'Member' is omitted (that is not present). An 'OmitValue' can be used in a 'MemberAssignment' for an optional 'Member' of a 'StructuredDataType' to indicate that the value for this 'Member' is deliberately omitted in the related 'StructuredDataInstance'..

Constraints

'OmitValue' can be assigned only to optional 'Member' s of 'StructuredDataInstance's.

This definition has the following problems that render it unacceptable:

1. The semantics definition is context-dependent on assignments to optional members and therefore only partially defined.
2. The given constraint restricts the syntactical use of OmitValue only and does not carry over to runtime semantics.
3. The given constraint cannot be checked by a tool at that location when the abstract syntax tree of a TDL spec is built.

**On 3)** The constraint requires the context of assignments to members in order to be checked. The correct context needs to be in a DataUse. There a constraint is already put in place. That is, the constraint shall be deleted.

The issue will become obvious when the constraints will be formalized as OCL expressions, which is planned in TDL3.

**On 2)** We cannot extend a static semantics rule to mean also execution semantics. See the introductory text of this section. That is, the semantics needs to be completely defined within the semantics section alone.

**On 1)** The definition of OmitValue in the semantics section is conditionally dependent on the existence of an optional member. However OmitValue occurs in a syntactically correct spec also outside of this context.

In the Ericsson example: if (x.field == omit) then doSomething();

The omit occurs as an argument to a function. That is, it appears independently from an optional member. Other cases could be constructed like:

1. Msg = messageBuildingFunction(data, omit);
2. X = omitReturningFunction(msg);
3. Y = anotherFunction(msg, omitReturningFunction(msg));
4. Call testDescription(1, 2, omit);

That is, the **semantics of omit in the examples above is not defined**. This shortcoming needs to be fixed.

When defining the semantics of syntactical elements one should apply common sense. Common sense tells us that the meaning of an object does not change depending on the context it occurs. Plain example: A laptop is a computer regardless whether it is used in office (the context) or at home.

As a consequence, the semantics of an element shall be defined using its properties.

The property of OmitValue is to represent no value. Therefore the semantics shall be defined as follows:

An 'OmitValue' denotes a symbolic value that represents no value at all.

We could also add a note to give a hint to a user about its purpose. But this note is not absolutely necessary as it becomes clear when the constraints are followed. Given the examples above, it is also not the only usage of OmitValue.

NOTE: An 'OmitValue' can be used explicitly in a 'MemberAssignment' for an optional 'Member' of a 'StructuredDataType' to indicate that the 'Member' is omitted in the related 'StructuredDataInstance'.

Finally, because the semantics definition of OmitValue does not make use of omitted member values, its name shall be reverted to **NoneValue** to avoid confusion by a reader of this MM draft. The name in TDL-GR shall still stay as **omit**.

In fact, we come back to the original definitions contained in the MM draft, because Ericsson's proposal is not a lively option and is incorrect as outlined above.

# On assigning omit to variables and non-optional members

It is ok to delete the newly introduced statements on undefined because they are in a sense redundant and do not add much.

There needs to be a **restriction in semantics of DynamicDataUse** that a data use expression that evaluates to OmitValue at runtime causes undefined semantics at TDL level. This restriction still allows OmitValue being assigned as argument explicitly because it is a StaticDataUse.

The following statement shall be added to the semantics section of clause 6.3.9 DynamicDataUse:

If a 'DynamicDataUse' expression evaluates to 'OmitValue' at runtime, it shall cause undefined semantics.