Input to TDL SG discussion

Andreas Ulrich, Siemens AG

Unfortunately I'm on a different assignment on Thursday. Nevertheless my input.

# On "<undefined> values causes undefined semantics"

It means that it defines undefined semantics at TDL level, which is a valid statement. It can be understood also implicitly under the assumption that if something is not explicitly written it is also not defined. It can be therefore deleted (again) in the Interaction clause and at other places. On similar grounds, the last (new) paragraph in 6.1 could be deleted in its entirety. The hint on tools could be a note.

Note by György: "NOTE: The  <undefined> value within the 'DataUse' specification of 'argument' is resolved outside of the TDL specification that may leave to an unexpected runtime behaviour of the SUT."

Could be simplified further to "Note: The <undefined> value within the 'DataUse' specification of 'argument' may be resolved outside of the TDL specification ". Because we cannot speculate in TDL how the outside world will behave.

# On semantics of OmitValue

This needs more elaboration. 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.

Static semantics is implemented by a (good) editor, while execution semantics needs to be considered by a compiler.

That is, we cannot extend a static semantics rule to mean also execution semantics automatically. This statement is a fact and not open for discussion (at least if we want to be taken seriously by other language engineers).

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);

If we forbid omit being an argument of a function (because an argument is not an optional member), we have no chance to check whether an optional member actually has omit assigned to it. Therefore we have to relax the meaning of omit (if we want or not).

Which cases from above examples are valid in static semantics and dynamic semantics??? Please discuss!

Proposal: Extend the semantics of omit as follows:

"An 'OmitValue' denotes a symbolic value indicating that no value has been assigned. It is a valid value for an optional member indicating that the value of this member is omitted from… (old text)."

We also keep all static semantics constraints in place.

# On assigning omit to variables and non-optional members

It is ok to delete the newly introduced statements on undefined. See also above.

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

Please check on the above examples how it works out. I do not have enough time doing this.