# MVF Submission Team Meeting notes

26 April 2019 & 3 May 2019

Participants: Evan Wallace, Elisa Kendall, Fred Cummins, Pete Rivett

EK: Roy Bell may be able to help with IDL for the interface, and review the interfaces for completeness/gaps.

Pete: We need a Pattern for representing an interface PIM in UML. Thought Ed S was going to provide this.

## Discussion about CRUD interfaces for MVF Metamodel elements

Pete: we can assume that the metamodel itself is enough to specify this.

## Walking through Pete’s scenarios resent in email 22 March 2019 and originally sent 11 July 2018

1. User adds a new class on a diagram called C1. The tool asks the user if it corresponds with any MVF entries in current vocabulary associated with a term C1 (if there are any). Let’s assume the user says No so a new class is created in the tool and a new MVF entry is created by the tool and associated with the internal id the tool has for the class.

The tool would use Davide’s Resolve( Term ) : List<Concept> operation to return the list of MVF entries to present to the user in 2nd sentence above.

Resolve (Term): List<Concept>

* Could add optional Context parameter for the case where the tool may expect multiple MVFEntries for the same term.
* Could also put some parameter for options for string matching
* Do we want to provide an optional parameter for Model Element Type? Should there be an attribute for this in the metamodel associated with an MVFEntry?

For sentence 3 above, we need a new operation to combine a bunch of elementary operations needed in creating a new MVFEntry and VocabularyEntry with attributes.

Two options:

SimpleCreateEntry (Term, Language): MVFEntryID

FullCreateEntry (Term, Language, optional externalReference, optional Definition): MVFEntryID

[Editor’s note: Davide uses Concept and ConceptID in his signature for what will call MVFEntry and MVFEntryID respectively.]

B) If the user wants to reuse C1 in another diagram they will normally drag the one just created onto a new diagram. But let’s say they don’t and just enter the name C1 on another diagram. The tool will prompt the user as before. The user can choose to select the existing class just created (making use of the MVF entry if needed) or decide it’s a different class and request to create a class in the tool and new MVF entry. At that point the tool will apply any uniqueness constraints to the newly created class, applying the rules of the modeling language (which as we’ve seen can get quite complex). In so doing it may need to get the names of the other elements in scope (i.e. that need checking) from the current vocabulary (though often it will have done that already e.g. to show the correct names in the containment tree for the current model or package). So that will make use of the bulk MVF operation to get a list of names for a bunch of MVF entries based on the current vocabulary. Having got the name, it will apply its local rules – based on both the modeling language and the tool-specific approach to errors (e.g. whether to disallow errors or flag them up by displaying them in red or with a ! icon).

The bulk operation mentioned above may look like:

LookupMVFEntries (List<MVFEntryID>): List<MVFEntry, Term, Definition>