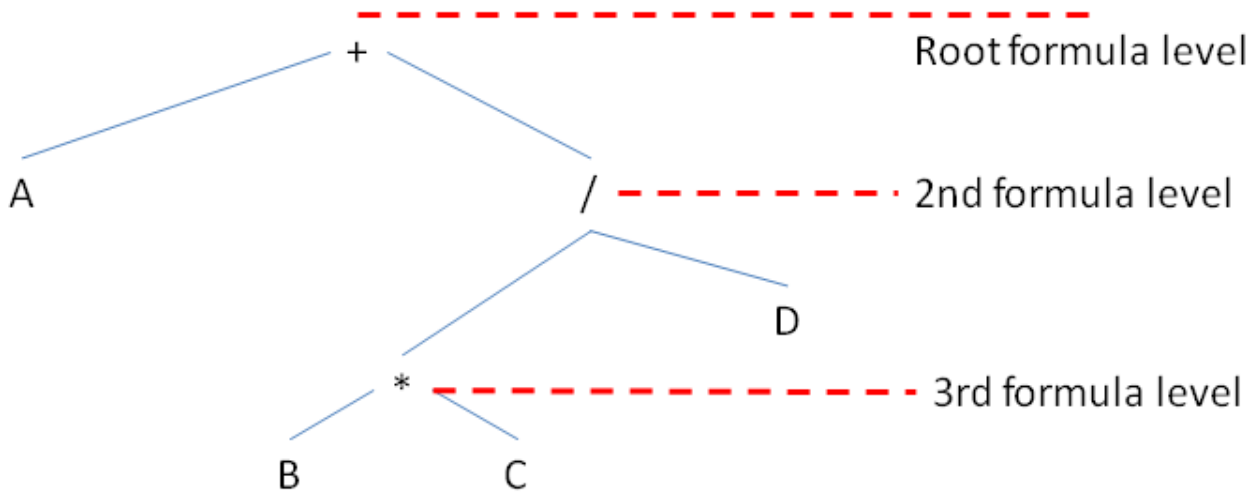
Metric Formula / Utility Function Modelling

In the context of the specification of utility functions and metric formulas (which can be considered as identical as a metric formula can be considered as a utility function) in CAMEL 1.0, there seemed to be a place of optimisation. The main current issue was that metric formulas were specified in a hierarchical manner comprising elements which mapped to sub-formulas that are associated with the invocation of certain functions. This lead to increased modelling effort at the modeller side. We supply a simple example to better clarify this issue. Suppose that we need to compute the following metric formula: A + ((B * C) / D). To specify this formula in CAMEL 1.0, we must specify the following hierarchy of formulas, where the top-most will be actually associated to a composite metric (that could map to the utility function to be optimised).



This maps to introducing three formulas which are correlated to each other:

- formula1: A + formula2 -> this maps to composite metric / utility function
- formula2: formula3 / D
- formula3: B * C

Thus, as it can be seen, instead of one formula, we need to model three. The number of formulas to be introduced actually depends on the amount of operators involved in the overall mathematical formula expression, by considering the fact that each metric formula maps to the application of just one operator.

Another detected issue concerned the fact that the amount of built-in functions that were supported by CAMEL 1.0 was rich but not extensive enough. We should stress here that built-in means here functions which are introduced via an enumeration at the meta-model level. In fact, as it has been proven through the determination of the utility formula of a certain Melodic 1.0 use-case, there exist functions that cannot be modelled in CAMEL like the beta one. This has reduced the applicability of CAMEL 1.0 and, as Melodic platform is tightly coupled to it, also Melodic's applicability. Further, as functions are introduced at the meta-model level, each new function that has to be incorporated leads to updating the CAMEL meta-model and its respective code, which could be considered as not a very good practice.

Based on the above two issues, a new mechanism was introduced in CAMEL, covering both the language and editor level, which enables to specify both in less modelling effort utility functions / metric formulas as well as to incorporate new functions / operators that can be used in these formulas. This has led to the proposition of the following solution to realise such a mechanism which is analysed in the following section.

Solution for CAMEL 2.0

Solution Analysis

This solution lies on the basic principle that one mathematical expression should be given for utility functions / metrics. Further, it should be straightforward to express this expression via a simple String which can be validated on-demand to explore its syntactic correctness. In addition, new operators can be introduced through the introduction of new functions again in a mathematical way, thus not requiring the implementation of these operators built-in in a certain enumeration such that the meta-model of the CAMEL language needs to be updated.

Based on the above principles, CAMEL was slightly extended in order to support this envisioned solution. First, the expression of metric formulas has been simplified. Instead of having references to sub-metrics, sub-formulas and other kinds of parameters and specifying the formula in a hierarchical manner, a metric formula can now just comprise a simple mathematical expression String that conforms to a certain mathematical language. The implication of supplying such a String comes with the implicit semantics of its content and comes with the following observations/needs: (a) there is a need for a mathematical tool to parse the expression to see if it is first syntactically correct based on the adopted mathematical language semantics; (b) semantic validation (not fully realised in CAMEL via OCL rules) can be also supported through: (i) the checking of the expression term types to see whether they are compatible; (ii) the checking of the expression term types to see whether they

map to terms already defined in CAMEL. Concerning (i), there is a need to check whether there is an expression where, for example, most of the terms are reals but there is one metric which has a string-based value type. In the context of the second semantic validation checking, there can be the case that there is a mispelling or a mentioning of a metric or function which has not been yet defined in CAMEL.

We should stress here that we accompany the specification of String-based formulas with an automatic generation of references to the sub-components (i.e., component metrics) of composite metrics. This has been realised at the editor level. This enables the modeller to have a clear view of what is being specified at the hierarchical level while enables the detection of metric specification cycles which can lead to the semantic invalidity of the respective metric model being specified (which is highlighted via appropriate editor messages).

A second change in CAMEL comes with the introduction of a new concept called *Function*. This concept represents the function or operator that can be used in a metric formula expression. It is actually expressed via a name as well as a String expression that encapsulates the definition of the function according to the language adopted by the mathematical tool supported by the platform. The introduction of this concept means that there is not any more the need for introducing a function enumeration at the meta-model level of CAMEL. On the contrary, functions are now specified at the model level which enables not to update constantly the CAMEL meta-model when the need to introduce a new function is raised. In addition, this maps to a certain flexibility in defining functions in the sense that there is no need to specify all the built-in functions of the mathematical language adopted. Instead, the modeller should concentrate only in the definition of new functions, which can be called as *user-defined*.

By continuing the previous example, suppose that the expression part B*C is replaced with a power operator application over B and C, i.e., POW(B,C). Further, suppose that POW does not belong to the built-in functions of the mathematical tool that is adopted for the parsing of CAMEL's mathematical expressions. This means that POW needs to be defined via a new *Function*. To this end, the modeller in CAMEL 2.0 would have to specify 2 main elements:

- the intended formula via the expression String: "A + (POW(B,C) / D)"
- the new function that is utilised by this formula which will be called POW while its body will be expressed as a String based on the language of the adopted mathematical tool. For instance, the body in Java-language style could be the following:

```
res = B;

for (int i = 2; i <= C; i++) res *= B;
return res;
```

On the other hand, by relying on the textual syntax of CAMEL, the following fragment shows how both the formula and the respective function could be specified in a certain metric model (where not all details in the elements being visualised have been given for space economy reasons):

```
camel model xxx{

metric model yyy{

function pow{

expression "X^Y"

arguments [X, Y]

}

composite metric zzz{

formula: ("A + (pow(B,C) / D)")

}

}

}
```

Technical Details

The above solution seems to be complete and does not require performing many modifications to the CAMEL language/meta-model. At the technical level, apart from the meta-model modifications, there is a need for the existence of a mathematical tool that should take care of mathematical formula parsing and evaluation. The use of such a mathematic tool comes with certain requirements that need to be satisfied, which include:

- capability to validate syntactically and semantically the mathematical expressions
    - the semantic validation would require taking into account terms that have been already specified in CAMEL, such as metrics and functions
    - special case holds when the value type of a metric is not given. In this case, the tool should consider that semantic validation should ignore this metric and concentrate only on terms with known value type. Ideally a warning message could be supplied which would then be communicated to the user/modeller
    - ideally the reasons for the invalidity of the expressions should be supplied - these reasons could then be communicated to the user in order to modify the respective expression to make it valid

- capability to supply all the terms of an expression - this can be beneficial for extra CAMEL validation purposes as the knowledge of the component metrics of a composite metric can be used for validating the context specified for that metric (e.g., that it includes sub-contexts that map to the component metrics of this composite metric)
- capability to evaluate the expression given
    - this requires that the knowledge about the actual values of metrics or other kinds of parameters is given to the tool in order to make the evaluation. This then also demands that the tool has the appropriate interface for supplying additionally this information
- capability to support an extensive list of built-in functions which should include both statistical and normal mathematical operators
- capability to define new functions via the use of a certain mathematical language
    - optional possibility to support the implementation of functions in the tool code could be also appreciated as this could cover the case where the offered language is not expressive enough for specifying the new function

The mathematical tool could be developed from scratch. Alternatively, we could adopt an existing tool and then adjust it according to our needs, if this is required. Obviously, such a tool should support most of the above requirements in order not to spend much effort in updating it to cover the satisfaction of the remaining ones. After conducting a small survey, such a tool has been discovered. It is named as MathParser (www.mathmarser.org). It not only satisfies all the above requirements but it is also supports different programming languages. It has also a quite rich set of mathematical operators/functions which could reduce the need to introduce new used-defined functions. The specification of the latter seems quite simplified and user-intuitive – this is actually a feature of the whole mathematical language that is adopted. Performance-wise, the tool seems to be quite fast but no thorough evaluation of its performance has been conducted.

Another important technical detail that was already mentioned relies on the fact that formulas are no longer checked for validity at the model but the editor level. This was a certain implementation choice which was imposed for the following reasons: (a) the OCL validation of CAMEL models would then be rather heavy; (b) there would be a need to couple the CAMEL code with the use of a certain tool while it is our desire to have CAMEL independent of any other tool. In this respect, the validation of models at the editor level was rather imperative. We now supply some technical details how this has been done in both CAMEL 2.0 editors and the respective reflection of this in the way the metric models can be specified.

In the textual editor, the checking of metric formulas is performed once the user attempts to save his/her model. This actually enables the user to have the freedom to specify composite metrics and respective formulas / utility functions in any order and not to be bothered by continuous validation error messages which are spawned while he/she is typing these formulas. Thus, once the user is confident that the metric model is complete, then he/she can press CTRL-S to save the model and then respective formula validation errors can be showcased. In our opinion, this is a more user-intuitive way of specifying models which also highlights why the respective validation functionality should not have been embedded directly in OCL constraints. Otherwise, the same distractive effect would have occurred where the user is annoyed by various formula validation messages.

In the web-based editor, the modelling is performed element-wise. This means that each composite metric needs to be defined in turn and the insertion of this metric comes directly with the validation of the respective metric model. In this respect, the user cannot specify the metrics in a random order but a specific one. This means that if there is a metric C which depends on metrics A and B based on its formula (e.g., C = A + B), then the metrics A & B need to be defined first before C can be defined. However, the user could still perform a nice trick here. He/she could still define C but leave its formula empty or provide an expression like "1 + 2". Then, he/she could define A & B and go back to the definition of C to change its formula (e.g., "A + B" instead of "1 + 2"). We should also mention that due to the way a composite metric is visualised, the web editor offers two nice capabilities: (a) it shows all the metrics that can be used to specify the formula of a composite metric – this could enable the user to understand that A & B have not been defined yet; (b) in case that the user supplies the formula of C without defining A & B metrics, then he/she also has the ability to validate the formula before saving the metric. So, whatever is specified in the metric formula can be immediately checked for syntactic and semantic correctness. The same is also done during attempting to save the metric which creates a double layer of guarantee over metric formula validity.

Thus, the editor functionality differs with respect to how metrics and formulas can be specified enabling the user/modeller to choose the editor which is more close to the preferred way such elements can be specified. This is another added-value of having the luxury of two editors accompanying the same modelling language.

Benefits

We should distinguish now between modelling benefits and Melodic platform benefits and we start with the latter. The mathematical tool to be adopted for mathematical expression parsing and evaluation will be exploited both by the CAMEL editor as well as by other components of the platform. For example, the Utility Generator could exploit it in order to evaluate utility functions based on parameter/metric values associated with a candidate solution. Further, the monitoring/Esper agents could exploit such tool to evaluate metric formulas in case of measurement aggregation – especially if there is a mechanism to include new operators that could enable the evaluation of advanced mathematical expressions. .

Concerning the modelling side, there is a clear benefit of simplicity, more user-intuitive modelling and less modelling effort through introducing this solution. Further, this solution achieves meta-model independence in terms of function specification, thus resolving one of the two main issues that raised the need to extend CAMEL. In addition, we should clarify that this solution does not prescribe the use of a certain mathematical language. In fact, the use of String-based expressions in formulas and functions caters for mathematical language independence. This means that it is left to the editor implementer (or the whole respective platform) to adopt any mathematical language of his/her choice. The language independence also leads to mathematical tool independence. In particular, each tool might have different built-in functions that are supported and different ways via which mathematical expressions can be supplied. Thus, by specifying formulas and functions at the instance level, we have the ability to exchange the use of a mathematical parsing/evaluation tool with another one.