MORPHEMIC

Final Data, Cloud Application & Resource Modelling

Modelling and Orchestrating heterogeneous Resources and Polymorphic applications for Holistic Execution and adaptation of Models In the Cloud

H2020-ICT-2018-2020
Leadership in Enabling and Industrial Technologies: Information and Communication Technologies

Grant Agreement Number
871643

Duration
1 January 2020 –
30 June 2023

www.morphemic.cloud

Deliverable reference
D1.3

Date
30 November 2022

Responsible partner
FORTH

Editor(s)
Kyriakos Kritikos

Reviewers
Paweł Skrzypek, Sebastian Geller, Geir Horn

Distribution
Public

Availability
www.morphemic.cloud

Executive summary

CAMEL is a multi-domain-specific language (multi-DSL) that is able to cover, in a rich manner, multiple aspects that are relevant for the management of multi-cloud applications. In fact, it has been demonstrated [1], [2] (see also Section 2.1) that this language is above competition in terms of multi-cloud application modelling.

In order to cover the modelling of polymorphic applications, CAMEL was extended to draft version 3.0 (see Deliverable D1.1 [1]). This change was driven by various requirements, drawn from the MORPHEMIC project. Similarly, the meta-data schema (MDS), a conceptual model for the Cloud and big data domains, was also similarly extended with the capability mainly to cover concepts and relations for various kinds of resources, including hardware-accelerated ones like field-programmable gate arrays (FPGAs) as well as network elements. MDS as a formal vocabulary complements CAMEL in order to enable the enhancement of CAMEL models via feature hierarchies decorated via MDS elements.

While the aforementioned changes were significant, they were deeply studied in the course of the MORPHEMIC project so as to improve them while new requirements came along, mainly from MORPHEMIC features and use-cases. This has led to further enhancing and improving CAMEL, which now comes in the form of a final version 3.0 (v3.0), as well as improving MDS. The goal of this deliverable is to present these new requirements and their rationale as well as the enhancements made to both CAMEL and the MDS. These enhancements are also demonstrated through the exploitation of a use-case originating from a MORPHEMIC partner.

This deliverable can be studied by both technical and use-case partners in MORPHEMIC for realising the MORPHEMIC Preprocessor, improving and enhancing Modelio's CAMEL graphical editor, as well as specifying the corresponding use-cases. It can be quite useful for an external audience that investigates the use of CAMEL and MDS for supporting polymorphic application modelling and subsequently management.

Author(s)

Kyriakos Kritikos, Yiannis Verginadis, Ioannis Patiniotakis, Adeliya Latypova

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871643

Table of Contents

List of figures

List of tables

1 Introduction

1.1 Scope

CAMEL is a multi-domain-specific language (multi-DSL) that is able to cover multiple domains that are relevant for the management of the multi-Cloud application lifecycle, including the deployment, requirement and metric domains. In fact, Achilleos et al. [2] have demonstrated CAMEL as superior to other similar languages in cloud application modelling in terms of richness, domain coverage, DSL integration level, cloud delivery model support as well as support for models@runtime [3]. This was a result originating from the evolution of CAMEL within the auspices of various European projects, including PaaSage, CloudSocket and MELODIC.

The MORPHEMIC project undertook a certain task to further improve CAMEL in order to support the modelling of polymorphic applications. In this respect, by gathering relevant requirements, and even feedback from previous projects, it enhanced CAMEL to reach version 3.0 by giving it the ability to completely support the specification of such applications. The qualitative comparison between CAMEL and other relevant languages in MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling* [1] demonstrated CAMEL's uniqueness in terms of polymorphic application modelling while that deliverable also analysed all the changes made to CAMEL along with their respective cause, i.e., a specific requirement from those collected.

During the course of the MORPHEMIC project, after its first year, CAMEL 3.0 was carefully examined by both technical and use-case partners of the project through its documentation and use as well as the participation in relevant demonstrations. This led to improving some original enhancements made to CAMEL 3.0. Furthermore, the work of some project features unveiled new requirements that were posed to this multi-DSL language in order to support the developments in these features. In this respect, CAMEL 3.0 was further improved and enhanced so as to reach a more final form that is approved by both kinds of partners in the project. The goal of this deliverable is to give insight on these new requirements as well as to analyse the further extensions and improvements that were made to CAMEL 3.0. In addition, it demonstrates CAMEL's further enhancement through a use-case originating from a MORPHEMIC partner.

The Meta-Data Schema (MDS) is a conceptual model that covers well both Cloud and big data domains. This model is used as a vocabulary that complements CAMEL as it enables to extend CAMEL models without affecting CAMEL's abstract and concrete syntax, i.e., changing CAMEL, with arbitrary feature hierarchies encompassing features at different levels and their corresponding attributes that characterise them. This support is quite relevant and important in the context of specifying resource and platform requirements, where the respective domains are large and continuously evolving, making it rather impossible to be fully covered at the conceptual level by any cloud application modelling language.

In order to cover the specification of polymorphic applications, MDS was enriched with new concepts, attributes and relationships focusing mainly on the resource and network domains while some slight enhancements were made at the big data domain. In the resource domain the focus was more on covering additional resource kinds, including hardware-accelerated ones like FPGAs and GPUs. As the network domain was originally not covered at all, the enhancements made to MDS were significant. All these enhancements are deeply analysed in MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling* [3].

Similarly, to CAMEL, MDS enhancement was studied by both technical and use-case partners of the MORPHEMIC project. The outcome from this study was that MDS is quite complete. However, some particular changes were required to be applied to MDS. First, additional artifacts were added to support the modelling of Bring-Your-Own-Node (BYON) in MORPHEMIC deployments. Second, additional hardware accelerator properties were introduced to address the modelling requirements of the MORPHEMIC pilots. Third, a new class/concept was generated to support annotating certain configuration-related metrics in a metric type model.

Last but not least, a final MDS change was applied to reduce its size, especially in terms of the attributes given per concept. In fact, there were multiple forms of the same concept characteristic/attribute that could be reduced to a single one. In this respect, during this second project year, MDS was normalised in order to respect this requirement. The main outcome was that MDS was made more compact and more easily and naturally usable by the project partners, especially the use-case partners. The latter usability result was achieved in concert with a specific change made to CAMEL 3.0 in terms of attribute specification.

1.2 Intended Audience

The content of this deliverable should firstly interest the use-case owners of the project who aim at properly modelling their multi-Cloud, polymorphic applications. Such a modelling can be conducted cooperatively between business experts and DevOps engineers within the use-case organisation. This content should also interest the technical partners of the project (researchers, architects, and developers) who need to rely on CAMEL and MDS features in order to design and implement relevant, and in many cases innovative, features of the MORPHEMIC Preprocessor. This deliverable is public so it is also open to external audience whose respective roles are the same as those mentioned for the internal audience. Similarly, to the case of the internal audience, the interested organisations in terms of the content of this deliverable can be those that aim to exploit CAMEL and MDS to describe their applications and potentially exploit the MORPHEMIC platform to deploy them as well as those that target further enhancing an existing platform with features that are relevant to CAMEL and MDS and polymorphic application management in general.

1.3 Document Structure

The remaining part of this document has been structured as follows:

- Chapter 2 revisits the state-of-the-art in response to the formal review comments with respect to the content of MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling*, i.e., the predecessor of deliverable D1.3.
- Chapter 3 explains the enhancements and improvements that have been performed to the new version (3.0) of CAMEL.
- Chapter 4 analyses the modifications made to MDS so as to better couple it with CAMEL v3.0 in terms of feature attribute specifications.
- Chapter 5 utilises a use-case from the MORPHEMIC project in order to highlight how the new versions of CAMEL and MDS jointly enable the complete modelling of a polymorphic application.
- Chapter 6 concludes this document.

2 State-of-the art revisited

2.1 Cloud Application Modelling Languages

This section will define the criteria for comparing and evaluating the CAMEL language to other DSLs and analyse the superiority of the CAMEL language based on these criteria.

In order to assess all relevant cloud application and service modelling languages that have been developed and proposed in the past, we rely on the criteria framework proposed by Achilleos et al. [2] that we extend in order to also cover the polymorphic modelling aspect. The criteria framework includes the following evaluation criteria, which focus on how well all relevant domains are covered and integrated, which kind of cloud services are supported and whether the models@runtime paradigm is adopted:

- *domain coverage*: this criterion signifies which domains from those relevant to the application lifecycle are covered by a language. We argue that a model-driven approach should be followed by a cloud application management platform as it enables to automate the various management operations in the application lifecycle. As such, central to this approach is the notion of a model, which in the context of the Cloud or multi-Clouds needs to cover well multiple domains, as each of these domains supplies appropriate knowledge and information which is required for proper cloud application management. These domains include the following: deployment, requirement, metric, scalability, security, organisation, location, execution, unit, type, data and provider [2]. In this respect, we use this domain coverage criterion in order to investigate which from these domains are covered by the state-of-the-art cloud application modelling languages. The possible evaluation values for these criteria are the following: (a) Low: if the language covers at most three domains, (b) Medium: if the language covers at most 6 domains and (c) High: of the language covers more than 6 domains.
- *integration level*: this criterion assesses what is the *level of integration*, see [2], between the different domains/sub-languages covered / utilised by a language. A language covering multiple domains might supply

different integration levels, especially when such domains include similar or equivalent concepts. As such, an integration solution to be adopted by a language must: (a) join equivalent concepts and separate similar ones into respective sub-concepts; (b) homogenise the remaining concepts at the same granularity level; (c) enforce a uniform formalism and notation for the abstract and concrete syntaxes; (d) enforce model consistency, correctness, and integrity. Each of these steps is a prerequisite to the following one while it also demands an increasing amount of effort. Based on this analysis, the goal of this criterion is to investigate how many of these steps have been applied by a language. Its evaluation spans the following values: "Low" if only the first step (a) was applied, "Medium" if steps (a) and (b) were applied, "High" if all steps were applied, and "N/A" if none of the steps was actually applied. The last evaluation value maps to the case where a language utilises some domain-specific languages (DSLs) as they are. This leads to the following disadvantages: (a) it raises the DSL complexity, since each DSL has its own abstract and concrete syntax; (b) it steepens the learning curve and increases the modelling effort for the same reason; (c) model duplication for similar or equivalent concepts; (d) manual validation of cross-domain dependencies that is error-prone and costly in effort and time.

- *delivery model support*: this criterion unveils which kinds of cloud services are supported by a language. A cloud application might utilise and integrate different kinds of services, which map to the three main cloud delivery models (IaaS, PaaS, & SaaS) and recently to the fourth one, the serverless or FaaS (Function as a Service). Each from these kinds of services adds a different capability to the cloud application or its management. An IaaS service provides the right, resource-rich environment for hosting application components. A PaaS service provides for a richer environment with tools and runtimes installed for faster and more reliable installation and execution of application components while it can also offer the use of middleware services. A SaaS or FaaS service enables to realise parts of the application functionality, thus reducing its implementation time and cost. In this respect, the more kinds of cloud services are utilised by an application, the better for its provisioning and management. As such, it is essential if a cloud application language can provide support for searching and integrating such services through the ability to specify deployment options and constraints/requirements related to these services. For instance, it could specify the actual runtime needed for installing and executing an application component as a requirement for the discovery of a PaaS service. As another example, it could specify resource constraints that must be supported by the IaaS service to be used for hosting an application component. Based on the above analysis, the goal of this criterion is to investigate which from these cloud service types are supported by a cloud application modelling language. As such, a language has an evaluation value of "IaaS" if it supports the use of IaaS services, of "PaaS the use of PaaS services, "SaaS" if it supports the use of SaaS services and "FaaS" if it supports the use of FaaS services. Obviously, a language can get multiple evaluation values depending on whether it can support one or multiple kinds of cloud services.
- *models@runtime support*: this criterion signifies for which domains is the models@runtime paradigm adopted by a language [3]. This paradigm enables the automatic provisioning of multi-cloud applications while it can be easily implemented using the type-instance pattern [3]. As such, we particularly argue that this type-instance pattern must be implemented at least in the deployment and metric domains. In the deployment aspect, it allows to automatically adapt the components and VM instances in the deployment model based on scalability decisions (e.g., scale out an application service/component and its underlying VM). In the metric aspect, the deployment adaptation is reflected also on the monitoring infrastructure. This criterion investigates the actual domains for which the type-instance pattern has been implemented by a cloud application modelling language. In this respect, the evaluation of a language spans the values of the different domains relevant to the cloud application lifecycle like "Deployment", "Metric", "Data" and "Execution". Obviously, a language can take multiple evaluation values depending on how many from its domains follow this type-instance pattern.

In our view and based on the requirements given in section 3.1.1 of MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling* [3], a cloud language can support polymorphic application modelling when it is able to satisfy the following additional criteria:

- *application architecture variability*: this criterion investigates whether a language is able to capture different forms of application components and thus cover subsequently the different variations that an application architecture can have.
- *component configuration variability*: this criterion examines whether a language is able to capture any kind of configuration that a component might have. This should include script, container, cluster, serverless, PaaS and

accelerated resource configurations. Thus, the higher is the number of the different configuration kinds captured, the better is the language.

- *component complexity*: application components in one form can be single, fine-grained elements and in another form should be split into other simpler components, thus being complex and coarse-grained in nature. This indicates the need for a language to support the specification of both single and complex components, where the latter can be realised through a composition of other components of smaller complexity.

Based on the above, enhanced criteria framework, we have analysed 14 provider-independent, state-of-the-art cloud application / service modelling languages, including CAMEL (v2.0 & v3.0) [2]. As provider-independence is a crucial characteristic in order to support cross- and multi-Cloud deployments. Please note that in comparison to MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling* content, we have added Kubernetes YAML as the "industry reference" to cloud application management. We could also add other languages like the one utilised in Juju[1] and OAM[2] but these languages cover very few domains and thus do not have very good evaluation values. Thus, it was decided to not include them in the evaluation. The evaluation results are depicted in Table 2.

These results map to those that have been already produced for the first 4 evaluation criteria in [2] and have been extended through the assessment of the three polymorphic-modelling related criteria. The latter criteria have been assessed as follows:

- *application architecture variability*: if a language does not support at all the modelling of component forms, it has a "Low" evaluation value. If it indirectly supports multiple component forms, it has a "Medium" evaluation value. Otherwise, it has a "High" evaluation value.
- *component configuration variability*: if a language supports one or two configuration kinds, it has a "Low" evaluation value. If it supports three to four kinds, it has a "Medium" evaluation value. Otherwise, it has a "High" evaluation value.
- *component complexity*: if a language does not make explicit the distinction of single and complex components, it has a "Low" evaluation value. If a language makes this distinction but does not properly model complex components, it has a "Medium" evaluation value. Otherwise, if the language completely models composite components as agglomerations of other components, it has a "High" evaluation value.

Table 1 Evaluation of Cloud Application Modelling Languages

Language	Domain Coverage	Integration Level	Delivery Model Support	Models@runtime Support	Application Archchitecture Variability	Component Configuration Variability	Component Complexity
Reservoir OVF Extension [4]	Low	N/A	IaaS	N/A	Low	Low	Low
Optimis OVF Extension [5]	Medium	N/A	IaaS	N/A	Low	Low	Low
Vamp [6]	Low	N/A	IaaS	N/A	Low	Low	Low
4CaaSt Blueprint Template [7]	Low	N/A	IaaS, PaaS	N/A	Low	Low	Low
TOSCA [8]	Medium	Medium	IaaS, PaaS	Deployment*	Low	Medium	Low
Provider DSL [9]	Low	Medium	IaaS	N/A	Low	Medium	Low

[1] https://juju.is/docs/olm/model
[2] https://oam.dev

Language	Domain Coverage	Integration Level	Delivery Model Support	Models@runtime Support	Application Archchitecture Variability	Component Configuration Variability	Component Complexity
GENTL [10]	Low	N/A	IaaS	N/A	Low	Low	Low
ModaCloudML [11]	Medium	Low	IaaS, PaaS	Deployment	Low	Medium	Medium
CAML [12]	Medium	Medium	IaaS	N/A	Low	Low	Low
Arcadia Context Model [13]	High	Medium	IaaS	Deployment	Low	Medium	Low
StratusML [14]	Medium	High	IaaS	Deployment	Low	Low	Low
HCL / Terraform[3]	Low	N/A	IaaS, PaaS	N/A	Low	Medium	Low
Kubernets YAML[4]	Low	High	IaaS	Deployment	Low	Low	Medium
CAMEL 2.0	High	High	IaaS, PaaS, FaaS, SaaS**	Deployment, Metric, Data	Medium ***	Medium	Medium
CAMEL 3.0	High	High	IaaS, PaaS, FaaS, SaaS**	Deployment, Metric, Data	Medium	High	High

*: TOSCA [8] has a respective interest group which works on extending TOSCA to include the coverage of the instance level at the deployment domain but the respective outcome is not yet part of the standard. There are also other extensions like TOSCA4QC [15] and TOSCAdata [16]. They will not contribute something more to the evaluation and TOSCA assessment result will not change because of these available extensions.

**: CAMEL has a version equivalent to CAMEL 2.0 which includes support for the SaaS level - conducted in the context of the CloudSocket project [17]

***: CAMEL 2.0 was mapping a component to multiple configurations but only one configuration per component was always supported (and has been realised in the current version of the Melodic platform).

As it can be seen from Table 2 and also derived from the review by Achilleos et al. [2], CAMEL 2.0 was already above competition in terms of its domain coverage, integration level, cloud service type coverage and the models@runtime support. This is due to the following reasons: (a) it supports the models@runtime paradigm in both the deployment, monitoring and data domains; (b) it has tightly integrated the right set of homogeneous DSLs; (c) it covers the PaaS & SaaS levels apart from the IaaS one; (d) it covers with the appropriate expressiveness level all the relevant domains to the cloud application management lifecycle. CAMEL 3.0, the new extension of CAMEL, builds on CAMEL 2.0 in order to enhance it with the polymorphic modelling feature. In this sense, MORPHEMIC has developed an enhancement of an existing language and its respective modelling framework that does provide support for polymorphic application modelling, which is a pre-requisite for polymorphic application deployment and adaptive provisioning. In the next chapter, this new version of CAMEL will be detailed in order to completely comprehend

[3] https://www.terraform.io/docs/configuration/syntax.html
[4] https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/#describing-a-kubernetes-object

how it enables the full specification of polymorphic applications in terms of all relevant application lifecycle management aspects.

To conclude this section, we should stress that a model-driven approach for the adaptive provisioning of cloud applications is not a research hype but a reality. This can be proven by the various cloud service and platform providers who have supplied specific languages in order to provide support for this kind of application provisioning. For instance, just take into account Cloud Formation or ARM DSL. However, the use of such languages favours provider lock-in, especially when they are tight to the respective cloud platforms that offer them. Even if this might not be always the case, as it was assessed in a recent paper of ours [18] all these languages are not yet ready for supporting multi-Cloud applications. So, imagine their inadequacy in also covering polymorphic, multi-Cloud applications.

2.2 MDS is a semantic vocabulary

As indicated in the previous chapter, MDS is a semantic vocabulary / conceptual model that complements CAMEL as it enables to enhance the semantics of CAMEL model elements especially in large domains which are continuously evolving. Such an enhancement is conducted by utilising annotations without affecting the syntax of the CAMEL language or its conceptual model. Thus, the use of MDS is mainly restrained for semantic annotation purposes of CAMEL models.

In this respect, MDS is not actually an ontology and does not play a competitive role with respect to ontologies, domain-specific ones or general, although as a semantic vocabulary it could be utilised in a similar manner with ontologies for semantic annotation purposes, while it can also have a semantic, ontology-based encoding. To this end, the state-of-the-art analysis that was performed in the context of MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling* in terms of MDS-related domains targeted mainly to analyse the relevant literature in those domains. The main analysis goal was to showcase which semantic approaches have inspired MDS in each relevant domain and have been utilised as a basis for the conceptual enhancement of MDS through the re-use of relevant concepts, properties and relationships.

In any case, we do not preclude the semantic enhancement of MDS in the near future in order to support some kind of inferencing tasks that could be relevant for the Cloud (super-)domain. For instance, MDS could be exploited for inferring the consistency between the annotations and the remaining content of a CAMEL model. As an example, consider the case where it is indicated that an application component should be situated in a public Cloud as a provider requirement but, on the other hand, a resource requirement enforces the use of a bring-your-own-node (BYON) node for hosting that application component. This consistency checking that could be performed through the use of semantic rules could enable to bridge the gap between the Unified Modelling Language (UML) based models (like the CAMEL ones) and their semantic annotations.

3 CAMEL Further Enhancement

As explained in the introductory section, CAMEL 3.0 was in a draft version that has been further enhanced and improved in order to become finalised. The goal of this chapter is to give an insight on what were the main requirements that drove these enhancements and to completely analyse them. To this end, this chapter is carefully broken down into three sections. The first section explains the new requirements of CAMEL, how they were generated while it gives an overview of its original requirements for polymorphic application modelling and why some of them have been reconciliated. The second section analyses the changes made to CAMEL by first providing an overview of these changes and then completely presenting them depending on the domain that they belong from those covered by CAMEL. The third section finally supplies some implementation details about CAMEL's current and more complete version.

3.1 Conceptual Analysis

3.1.1 Original Project Requirements

CAMEL's 3.0 draft form was produced by generating, collecting and applying a set of requirements that were grouped into four main categories: (a) polymorphic-modelling related, (b) improvement related, (c) feature-related, (d) use-case related. The following table provides an overview of these requirements. A deep analysis of these requirements can be found in MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling* [1].

Table 2 Original requirements that led to CAMEL 3.0 draft version

Requirement ID	Requirement Short Description	Requirement Group
PM1	Cover new configurations, especially for hardware-accelerated resources	Polymorphic-modelling related
PM2	Make hosting relations non-obligatory for modelling	Polymorphic-modelling related
PM3	Each configuration of a component should come with its own requirement set	Polymorphic-modelling related
PM4	Supporting the modelling of composite application components	Polymorphic-modelling related
IR1	Component re-use across application models	Improvement related
IR2	Introduction of technical communication semantics	Improvement related
FR1	Support the modelling of predicted metrics	Feature related
UR1	Allow the specification of communication requirements	Use-case related

3.1.2 New Project & Reconciliation of Existing Requirements

As indicated in the introductory chapter, CAMEL 3.0 was thoroughly investigated and discussed by all partners of the project. This led to a reconciliation of the CAMEL 3.0 original requirements as well as an improvement on the extensions made to that language during the first project year. The reconciliation was based on two major axes: (a) backward compatibility and (b) MORPHEMIC platform development scheduling.

Backward compatibility enforced that all changes to CAMEL from 2.0 to 3.0 should be extensions that do not change the original structure of CAMEL by removing or re-organising CAMEL meta-model elements, e.g., classes, attributes and properties. In this way, models conforming to CAMEL 2.0 could still conform to CAMEL 3.0 and are thus still valid. Such restriction was deemed important based on the main rationale of MORPHEMIC: it is an extension of the MELODIC platform through the addition of new modules or parts. As such MELODIC platform should not be changed in order to incorporate the MORPHEMIC Preprocessor.

The MORPHEMIC platform development scheduling is a continuous process that prioritises development tasks which focus on adding new features or improving existing ones. In this respect, this process led to carefully examining new CAMEL 3.0 features in order to foresee which ones map to features of the MORPHEMIC processor that will or could be realised in the course of the project. As such, those CAMEL features that are deemed not necessary could be just removed from CAMEL, especially if they do not have any research or academic impact.

According to these two axes, CAMEL 3.0 original requirements were reconciled. The decisions made per requirement are presented in the following table.

Table 3 Original requirements for CAMEL 3.0 reconciled

Requirement ID	Requirement Short Description	Reconciliation
PM1	Cover new configurations, especially for hardware-accelerated resources	Support for such new configurations is important for the MORPHEMIC platform so this requirement was not changed.
PM2	Make hosting relations non-obligatory for modelling	This requirement does not affect backward compatibility while it is important at the research level. Further, it reduces the modelling effort of platform users. So, it is kept as is.
PM3	Each configuration of a component should come with its own requirement set	This requirement is kept as is, especially as it is absolutely correct and critical for polymorphic modelling – each configuration leads to a new component form that comes with its own requirements and restrictions.
PM4	Supporting the modelling of composite application components	This requirement was dropped as it has been decided that component forms that lead to sub-deployment models (for composite components) increase the complexity in deployment reasoning, which is already increased due to the need to support multiple forms per component, with each form having its own requirements. Thus, sub-deployment models for composite components will not be realised at all by MORPHEMIC.
IR1	Component re-use across application models	Due to the development of CAMEL's graphical editor based on Modelio, which can incorporate the ability to drag and drop components from different models, it has been decided that this CAMEL feature is not actually needed and thus component re-use is to be realised at the editor level only.
IR2	Introduction of technical communication semantics	While this is an important research-oriented feature, it is already covered partially in CAMEL 2.0 through the use of constraints involving metric variables. Further, it has been decided by the consortium not to be implemented in the MORPHEMIC platform as it is not needed by any use-case partner and is costly in terms of resources to implement.
IR3	Identification of MDS elements supported by the MORPHEMIC platform	This was an essential improvement feature that needs to be kept as it enables the users of the platform to know which metadata elements are supported and can thus be utilised, e.g., to annotate CAMEL models.
FR1	Support the modelling of predicted metrics	It has been decided that the modeller should not model any prediction metric as prediction is something that is supported by the platform itself, which can decide autonomously when and how to use it. As such, this requirement is rather dropped.
UR1	Allow the specification of communication requirements	This is a feature that could be implemented by the platform in the near future as it belongs to a use-case requirement. Even if this feature is not directly but indirectly realised by the platform through other means, it is considered as an important addition in CAMEL. So, it has been kept.

Based on the content of the above table, it can be easily inferred that 3 out of the 8 original requirements for CAMEL 3.0 have been dropped. In this respect, the respective extensions (mapping to these requirements) that were performed to produce CAMEL's 3.0 draft version have been removed. Further, due to the need to maintain backward

compatibility, it has been decided that the application meta-model needs to be dropped (see MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling* [1]). This led to the need to cover an application polymorphic model through a deployment model at the type level so as to re-use a CAMEL 2.0 feature. Fortunately, deployment type models in CAMEL do support multiple configurations per component so through the newly introduced mapping of each configuration to its own requirement set, the application polymorphism modelling is fully supported by CAMEL 3.0.

During the project's second year, new requirements came along related to further extending CAMEL or improving it. These requirements fit perfectly with the aforementioned requirement categories. One requirement is feature-specific, another is use-case specific and the third one is improvement-specific. These three requirements are summarised in the following table:

Table 4 New requirements for CAMEL 3.0

Requirement ID	Requirement Short Description	Type
IR4	Have the ability to specify attribute constraints apart from attribute values in features	Improvement related
FR2	Enhance metric variables to encapsulate the metric that they predict	Feature related
FR3	Introduce control-flow relationships between application components	Feature-related
UR2	Allow the specification of window pre-processing activities for the computation of composite metrics	Use-case related

Requirement IR4 came from the observation of some technical partners that in resource/platform requirements, the ability to specify non-equality constraints on feature attributes does not exist. This creates issues in expressivity of CAMEL models and the size of the MDS. Requirement FR2 originates from Feature 2: *Proactive Adaptation* of the MORPHEMIC project. It relates to the new characteristic of the MORPHEMIC platform in terms of metric prediction. While CAMEL's expressiveness in specifying metrics is quite rich, there was a need to add one particular property in (composite) metric variables so as to cover the two major possibilities in the production of their values (see MORPHEMIC deliverable D2.3 *Proactive utility: Framework and approach* [19]). On the other hand, requirement FR3 originates from Feature 1: Polymorphic Adaptation feature of the MORPHEMIC project. It relates to the need to capture control-flow relationships between application components as these can further enable to improve the application deployment via the deployment reasoning process. Finally, requirement UR2 originates from the ICON use-case [20], [21] where it is required to perform a new form of window processing, which is not currently supported by CAMEL. All these requirements and the way that they have been implemented are analysed in detail in Section 3.1.4.

3.1.3 CAMEL Enhancement Process

CAMEL 3.0 was produced as a draft version at the end of the first project year. It was then validated by both technical and use-case partners through a series of workshops and telcos. From these telcos, valuable feedback was obtained in the form of the two axes aforementioned, which led to the reconciliation of the respective requirements. Afterwards, the development of CAMEL took the following form:

- First, the reconciliated version of CAMEL was named as CAMEL-3.0-draft and it was incorporated as a (feature) branch[5]
- Next, the respective new requirements were received. Each requirement was treated as a new feature branch that was incrementally developed by the previous one. This led to the following transitions:

[5] https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow

- From CAMEL-3.0-draft to CAMEL-3.0.1 covering FR2
- From CAMEL-3.0.1 to CAMEL-3.0.2 covering FR3
- From CAMEL-3.0.2 to CAMEL-3.0.3 covering UR2
- From CAMEL-3.0.3 to CAMEL-3.0.4 covering IR4

This way of work enabled to validate: (a) the reconciliation of CAMEL and (b) each new requirement as it led to a new minor version which could be immediately studied by both technical and use-case partners. Further, this way of work will be followed also for any new requirement or modification that is needed for CAMEL 3.0 in the near future, either in the context of MORPHEMIC or beyond.

To be noted that each new CAMEL branch incorporates a full implementation of a respective requirement for change or extension. So, this covers both CAMEL abstract and concrete/textual syntax. In this way, the CAMEL editor can be used, e.g., by the use-case/technical partners in order to examine a respective sub-version of CAMEL. In any case, an accompanying documentation for the CAMEL extension/change is produced in order to facilitate the validation task of the project partners. Furthermore, a dedicated telco/workshop is always organised for that purpose.

3.1.4 Enhanced CAMEL Version 3.0

3.1.4.1 Overview of Changes/Extensions

In order to accommodate for the reconciliation of CAMEL as well as the addressing of new requirements, various changes have been performed in CAMEL at the abstract and concrete syntax level. The following table summarises these changes by also supplying their origin. All these changes are deeply analysed in the following sections.

Table 5 Changes towards CAMEL 3.0

Domain	Change	Rationale	Origin
Core	Enhancement of attributes to include their minimum and maximum values	Have the ability to specify constraints on the minimum or maximum value of attributes apart from their exact value.	IR4
Deployment	Incorporation of a reference to a respective requirement for a Communication	Ability to refer to constraints on the quality of communication specified via a communication requirement.	UR1
Deployment	Incorporation of a reference from a configuration to its set of requirements	A component configuration should have its own requirement set as different configuration kinds tend to differ with respect to their requirements. This does not preclude the existence of common requirements across all configurations. In that case, such requirements are directly referenced by the respective application component.	PM3
Deployment	Introduction of new configuration kinds	As MORPHEMIC aims at supporting new kinds of resources for hosting application components, such resource kinds (e.g., hardware-accelerated ones) should correspond to new configuration kinds for application components (e.g., image configurations).	PM1
Deployment	Introduction of control-flow relationships between application components	These relationships enable to make better decisions in terms of application placement as they can indicate when application components are actually executed, such that we can save resources and create them only when needed. Such relationships are, of course, relevant mainly for workflow-based applications.	FR3
Requirement	Introduction of communication requirements	Need to introduce a new requirement kind which can include constraints on the quality of communication	UR1

Domain	Change	Rationale	Origin
		between two or more application components.	
Metric	Object context reference to communication	The object that is being measured in an object context within a metric context can be a component, data or now the communication between components, giving the ability to define metrics over the communication quality.	UR1
Metric	Metric variables can now refer and encapsulate metrics	Metric variables can now refer to both formulas and metrics. In the first case, they can be computed by evaluating these formulas at the beginning of the application deployment. In the second case, they can be computed through metric prediction over the current solution alternative examined by a MORPHEMIC Solver once enough data for the prediction are available.	FR2
Metric	Introduction of window processings	These new concepts represent a new ability to be featured by MORPHEMIC in terms of performing certain kinds of window pre-processing, i.e., processing measurements (e.g., filter or group them) before they can enter a window. Only the remaining measurements in the window after this pre-processing will be used for computing the respective composite metric's value.	UR2
Metadata	Introduction of implemented attribute in metadata objects	The MDS is quite large to cover any kind of conceptualization relevant for Cloud services and big data. The MORPHEMIC platform supports only a part of MDS which is signalled to the platform user via this attribute.	IR3

3.1.4.2 Core Domain Extension

Features in CAMEL represent concepts that can shape arbitrary hierarchies of sub-features and attributes. Each feature attribute has been regarded as a characteristic of a feature that can take a specific value. Features and attributes are mainly utilised in CAMEL in order to formulate constraints on resources and platforms in the context of the corresponding resource and platform requirements. Both features and attributes are annotated through the use of MDS elements. For example, one resource requirement could relate to a feature mapping to the *CPU* of a resource (i.e., a certain MDS concept) with an attribute, mapping to the *hasMinNumberofCores* property in MDS, that has a value of 4. In other words, this resource requirement indicates that the CPU of the respective resource should have at least 4 cores.

This particular example actually pinpoints to a certain improvement place for both CAMEL and MDS. The *hasMinNumberofCores* property seems to be a kind of artificial CPU characteristic in the sense that it carries out some additional semantics conveying the use of the *min* operator over the actual value that this characteristic can take in a VM offering. In other words, characteristics like CPU are always exact and cannot vary over time for a specific resource such that we can require to take their minimum or maximum value. As such, it is better to represent them through clear, concise and minimal semantics. This actually indicates the need for CAMEL to support the specification of attribute inequality constraints apart from equality ones. As the handling of such characteristics requires the use of non-equality constraints for them. In the current example, it is more intuitive to express something like the number of CPU cores for a resource should be greater or equal to 4 rather than to say that the minimum number of CPU cores for a resource should be 4, which does not make sense.

In this respect, we can distinguish between two kinds of attributes:

- Absolute ones whose value does not change in an offering like the number of cores. For such attributes, there is a clear need, as stated above, that there should be in place attribute constraints that can enable to restrain the values that these attributes can take in the context of a resource requirement.

- Operator-specific ones whose value can vary in the context of a specific resource like the case of CPU frequency. Such attributes might be already characterised by minimum and maximum values being supplied by the providers of the respective offerings in the form of a (value) range. In this case, it does make sense to have properties in MDS like *hasMaximumFrequency* and *hasMinimumFrequency* to properly cover such attributes and especially their ranges. For such attributes, it might make sense then to specify equality and inequality constraints over one or both limits of the respective attribute range. For instance, we could specify that the minimum CPU frequency of a resource should be greater or equal to 2.0.

Before entering into the details how the above problem was confronted in CAMEL, we must indicate that the aforementioned attribute classification represents an improvement potential of MDS, which was actually employed. As MDS modelled some absolute attributes as operator-specific ones and the opposite. Thus, this change in CAMEL has led also to a respective modification to MDS in order to further improve MDS. In fact, as it will be shown in Section 4.1, MDS size was actually reduced due to having multiple cases where absolute attributes were modelled as operator-specific (i.e., two properties instead of one were always used to cover an absolute attribute).

The extension that has been conducted in CAMEL to address this issue is shown in Figure 1. The extension concerns mainly the *Attribute* class for which four new attributes have been introduced, namely *minValue*, *minInclusive*, *maxValue*, *maxInclusive*. The first two attributes indicate the minimum value that a certain feature attribute is requested to take and when *minInclusive* is true, the semantics is that the feature attribute should be equal or greater than the min value (>= operator), while when it is false that the feature attribute should be greater than the min value (> operator). Symmetrically, the *maxValue* and *maxInclusive* attributes indicate the maximum value that the feature attribute can take and whether it is included (<= operator) or not (< operator). To remind the reader, the existing *value* attribute can be used to cover equality constraints on feature attributes (, i.e., the == operator).

By considering the current example, if someone desires to specify that a CPU should have at least 4 cores and a maximum frequency of 2.0 MHz, then by assuming that the *CPU* concept in MDS has three relevant attributes, namely *hasNumberOfCores*, *hasMinFrequency*, and *hasMaxFrequency*, respectively, then these constraints could be specified in CAMEL through the specification of a feature, annotated via the *CPU* concept, that has two attributes:

- one Attribute, annotated with MDS *hasNumberOfCores* property, having the value of 4 for its *minValue* attribute and the value of true for the *minInclusive* attribute
- one Attribute, annotated with MDS *hasMaxFrequency* property, having the value of 2 for its *value* attribute

The respective textual specification in CAMEL for this example is shown below:

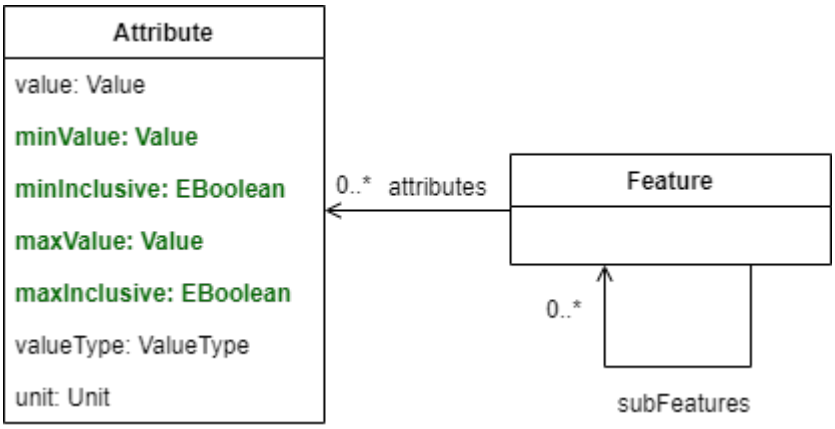

Figure 1 The enhancements (in green colour) to the Attribute class

3.1.4.3 Deployment Domain Extensions/Changes

The deployment domain has been updated with 4 main modifications/enhancements. All these changes are shown in Figure 2, which covers the respective part of CAMEL where these changes are introduced / take place. The first

change is communication-related, analysed in Section 3.1.4.3.1, the next two configuration-related, analysed in Section 3.1.4.3.2, while the last one concerns a new component relationship kind, analysed in Section 3.1.4.3.3.

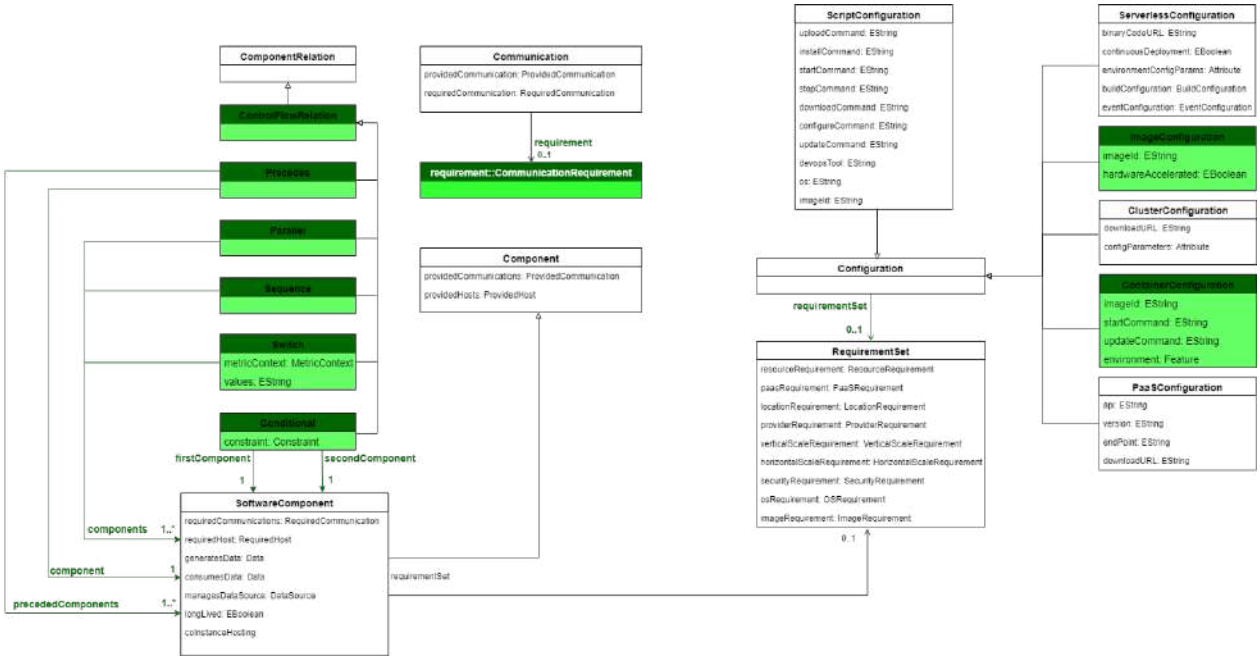

Figure 2 Enhancements in CAMEL's deployment meta-model

3.1.4.3.1 Communication-Related Change

This change corresponds to an enhancement of the *Communication* class, which can now refer to a *CommunicationRequirement*. This change enables to associate a communication between two or more application components with the requirement that this communication must satisfy. Such requirement can be related to a set of constraints on the quality of the communication, covering quality attributes like the latency. These constraints can have an impact on application placement. For instance, if the communication latency is required to be quite small, the MORPHEMIC platform might decide to couple the communicating components in the same host.

3.1.4.3.2 Configuration-Related Changes

The first configuration-related change concerns the application polymorphism, which relates to the component polymorphism in turn. In particular, when an application component is polymorphic, this means that it can have different implementations that map to different configuration kinds. Thus, logically speaking, each such implementation and respective configuration might come with its own unique requirements (e.g., resource ones). For instance, component A might have a VM configuration with the requirement to have 2 CPU cores and 2 GBs of RAM and a serverless configuration with 1 CPU core and 1 GB of RAM. Such differentiation is logical, if we consider that VMs include whole operating systems and thus require additional resources in order to operate correctly. Nevertheless, some requirements might be the same across different configurations of the same component. To this end, the association between an application component and a set of requirements is still kept. As such, both a *SoftwareComponent* and a *Configuration* are associated with a property to a *RequirementSet*.

The second configuration-related change relates to the introduction of two new configuration kinds/types in CAMEL 3.0, namely the *ImageConfiguration* and *ContainerConfiguration*. The *ImageConfiguration*, as indicated in MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling* [1], concerns two main cases: (a) the identification of a VM image which already includes the respective application component. As such, when a VM is produced out of this image, the application component will be up and running; (b) the identification of the application component image from which the respective component can be instantiated in environments which include hardware-

accelerated resources like FPGAs. On the other hand, a *ContainerConfiguration* is a configuration targeting an application component that takes the form of a container. Such a configuration specifies the identifier (ID) of the container image, a feature via which the configuration attributes can be specified as well as the tool/framework-specific specialised commands for starting up or updating a container-based application component. Please note that it is assumed that Docker is the default container management tool/framework. If this is not the case, then the actual container management tool/framework can be specified through an annotation.

3.1.4.3.3 Component Relationship-Related Changes

In workflow-like (Cloud) applications, the application components take the form of (workflow) tasks which map to *ClusteredConfiguration*s. In this respect, the execution of the application tasks is governed by the control and data flow of the application workflow. Implicit data-flow relationships, like one component A produces a data item I which is required as input for Component B, are already captured by CAMEL 2.0. These indicate that one component, e.g., B, should be executed after another component, e.g., A. On the other hand, control-flow (application) component relationships were not covered by CAMEL 2.0. However, both data- and control-flow relationships are quite important as from them the relative order of execution between application components can be inferred. Such an ordering along with a good estimation of component execution time can enable to take improved placement decisions. For instance, if we know that for the beginning, only components A and B are executed but not C, then we can decide to place only these two components and not the third one. Then, if we know the maximum expected execution time of A and B, we can infer when C could execute. Furthermore, we could retain the host of the slowest component from A and B in order to remove that component, when it finishes execution, and install there component C. This can enable to speed up application reconfiguration time as there is no need to create a new host for provisioning component C.

Based on the above analysis, as CAMEL 2.0 already captured implicit data-flow relationships, it has been decided that CAMEL 3.0 should cover control-flow relationships. However, such a coverage should not be as complete as possible in terms of modelling as CAMEL is not intended to replace a workflow language. On the contrary, what needs to be covered are just the most basic control-flow relationships [22], which can then be utilised in order to build more composite ones. Such basic relationships are considered as adequate for the purposes of application placement because: (a) they are enough for inferring the relative execution order between components; (b) do not lead to increased modelling effort; (c) the complete workflow of the application is not duplicated.

Control-flow relationships (see *ControlFlowRelation* class) are kinds of *ComponentRelation*s, so they relate two or more application components together. They can be categorised as follows:

- *Precedes*: this relationship is associated with one component that precedes in execution order with respect to a set of other components. For instance, we can indicate that component A is executed before components B and C
- *Sequence*: this relationship indicates a sequence of components, which are executed one after the other
- *Parallel*: this relationship indicates that some components are executed in parallel
- *Conditional*: expresses a conditional execution relationship when the execution of one component over the other depends on the evaluation of a specific constraint. If the constraint is not violated, the first component is executed; otherwise, the second. This resembles if-then-else statements in programming languages
- *Switch*: in this relationship one from multiple components will be executed depending on the evaluation result of a specific metric (see associated metric context in this relationship). For instance, if we have a metric that can take 3 values: 1-3, then if the current metric measurement equals to 1, the first from the component will be selected for execution; if it equals to 2, the second; and if it equals to 3, the third. Obviously, the set of possible metric values should have the same size as the set of components.

3.1.4.4 Requirement Domain Extensions/Changes

In this domain, only one addition was incorporated (see Figure 3), relating to the introduction of communication requirements (see *CommunicationRequirement* class). They represent a hard requirement kind that needs to be satisfied at all costs by the platform. Such a requirement kind should be related to a set of constraints over the quality of a communication (between two application components). As any kind of requirement is a Feature, it has been decided that the content of the *CommunicationRequirement* class should be empty. This is a right decision as a feature can have multiple attributes and can form a hierarchy of sub-features. In this sense, we can specify constraints for

communications in the same way we formulate constraints for resources and platforms. For instance, we can indicate that the communication latency between communicating components A and B is 100 ms by specifying an attribute within a *CommunicationRequirement* element that is annotated with the respective (latency) concept in MDS and has the value of 100 and the unit of ms.

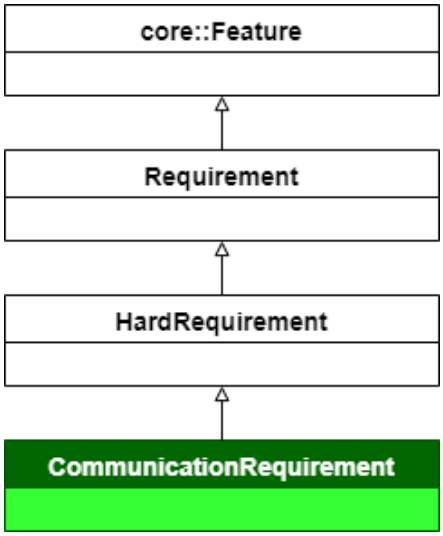

Figure 3 Enhancement of CAMEL's requirement meta-model

3.1.4.5 Metric Domain Extensions/Changes

Three changes have been performed in this domain which relate to three different requirements for CAMEL enhancement, namely UR1, UR2 and FR2. These three changes are analysed in the following three sub-sections.

3.1.4.5.1 Communication-Related Change

In order to specify communication-related SLOs as well as optimisation functions, there is a need to specify communication metrics. In this respect, while CAMEL 2.0 was already rich enough to specify any kind of metric, it was observed that only metrics related to components and data could be specified, thus the object context of a metric was restricted not to include any kind of communication-related elements. To this end, in order to provide full support for communication metrics, the metric domain in CAMEL was enhanced (see Figure 4) through the addition of a reference from an *ObjectContext* to a *Communication*. In this way, the communication puzzle in CAMEL has been solved as: (a) a *CommunicationRequirement* could cover constraints on static properties for communications; (b) SLOs and optimisation functions can cover constraints and utilities related to communication (quality) metrics; (c) additionally, technical communication semantics now govern the coupling of communicating components at the instance level. As such, requirement UR1 can be indicated to be completely satisfied by the enhancements made to CAMEL.

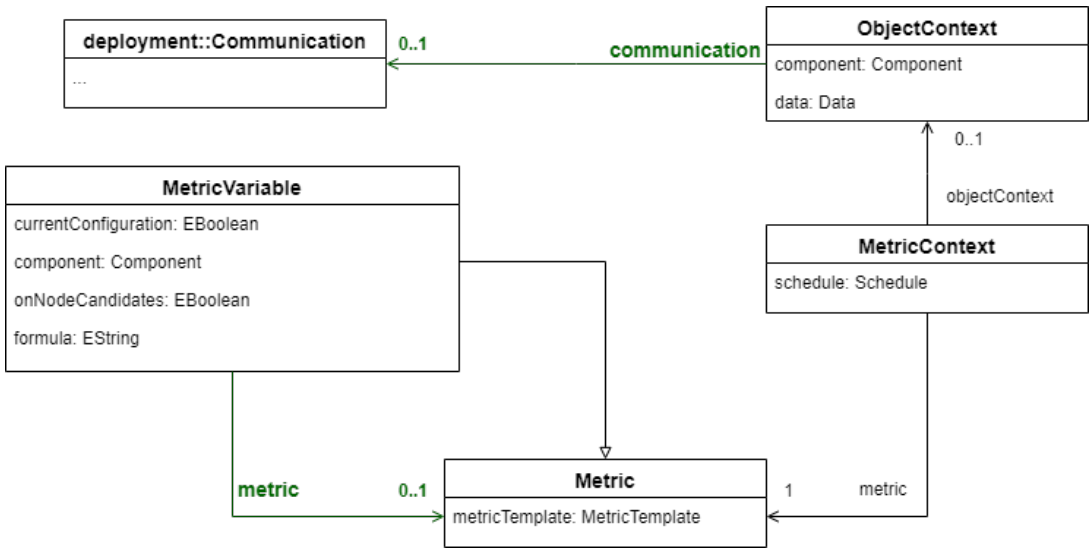

Figure 4 Communication & prediction-related enhancements to CAMEL's metric meta-model

3.1.4.5.2 Metric Prediction-Related Change

As indicated in detail in MORPHEMIC deliverable D2.3 *Proactive utility: Framework and approach* [19], metric forecasting is a novel feature of the MORPHEMIC platform which can be exploited after some period since application execution due to the well-known accuracy issue of inadequate historical input. In particular, during application initial deployment as well as during the initial attempts for application reconfiguration, it would not be possible to forecast the performance of application components and in consequence of the whole application when these components are placed in specific resources, where such a placement decision is considered during application deployment reasoning. In this respect, the workaround, also applied in the context of previous European projects like MELODIC, was to provide a specific formula for the calculation of the respective metric variable[6] which involved other metrics or metric variables. Please note here that a metric variable is required for this purpose as we deal with an optimisation-related variable whose value changes when different placement candidate solutions are examined by a Solver.

CAMEL 2.0 covered well the modelling of the aforementioned mathematical formula for the production of the values of the respective metric variable. However, due to the implementation of the forecasting feature, this formula is applied only initially; when forecasting accuracy is adequate, a forecasted performance value can be computed instead. In this respect, CAMEL 2.0 was enhanced through the following extension: associating a metric variable with the metric whose values can be forecasted through that metric's context. As such, the existing mathematical formula of a metric variable is applied initially while the reference to the metric is an indication to the platform that the forecasting feature needs to be applied in order to forecast the metric variable values, whenever this is possible. This CAMEL extension is depicted in Figure 4 (see *MetricVariable* class).

For example, suppose that we need to compute the completion time (CT) for an application. By applying the workaround, we create a metric variable named as *CT_var* and associate it with the following formula: $\theta_1 \cdot \theta_2/(c_1 \cdot c_2) + \theta_3$, where θ_i are metrics: θ_1 represents number of trainings left to do, θ_2 the percentile bound on task execution and θ_3 the elapsed time. and c_i are metric (decision) variables with c_1 representing the number of instances and c_2 the number of cores. This formula intuitively enables to select the more powerful application configurations when the application performance is bad and less powerful application configurations when the application performance is too high. Through the MORPHEMIC platform prediction feature and CAMEL's respective extension, the metric variable is also associated with the context *CT_Metric_Context* of metric *CT* that measures the application's completion time.

[6] Please note that a metric variable is a decision variable whose value is assessed by a Solver. On the other hand, a metric is something that is monitored (either by the platform or the application itself).

As mentioned above, this is an indication that the *CT* metric measurements will be exploited for forecasting purposes when adequate historical input exists. The respective CAMEL metric model sub-part that shows the definition of the metric variable *CT_var* is given below (see more details in MORPHEMIC deliverable D2.3 *Proactive utility: Framework and approach* [19]).

```
variable CT_var{

template MetricTemplateCamelModel.MetricTemplateModel.Completi
onTimeTemplate
formula: ('(Theta_1 * Theta_2) / (C1 * C2) + Theta_3')

  context CT_Metric_Context
}
```

3.1.4.5.3 Window Processing-Related Change

3.1.4.5.3.1 Introduction

One of the most critical and important features of the metric domain in CAMEL is the separation of concerns in terms of application measurement. In particular, sensors and metric formulas cover the measurement computation part, depending on which kind of metric is being measured, while schedules cover how often the triggering of computations occurs and windows define the exact data (e.g., raw or composite measurements), collected in between computation triggerings, to be used for the measurement computation. The following Figure 5 showcases this separation of concerns while it highlights how these different conceptualisations come together. As it can be seen, contexts are the ways to group the relevant concepts whose content varies depending on the metric type to be computed. Raw metrics are sampled according to a schedule through the use of sensors. On the other hand, composite metrics might be computed from other metrics (either raw or composite) through those metrics' measurements, which are collected in well-defined windows while their computation is still triggered via a schedule.

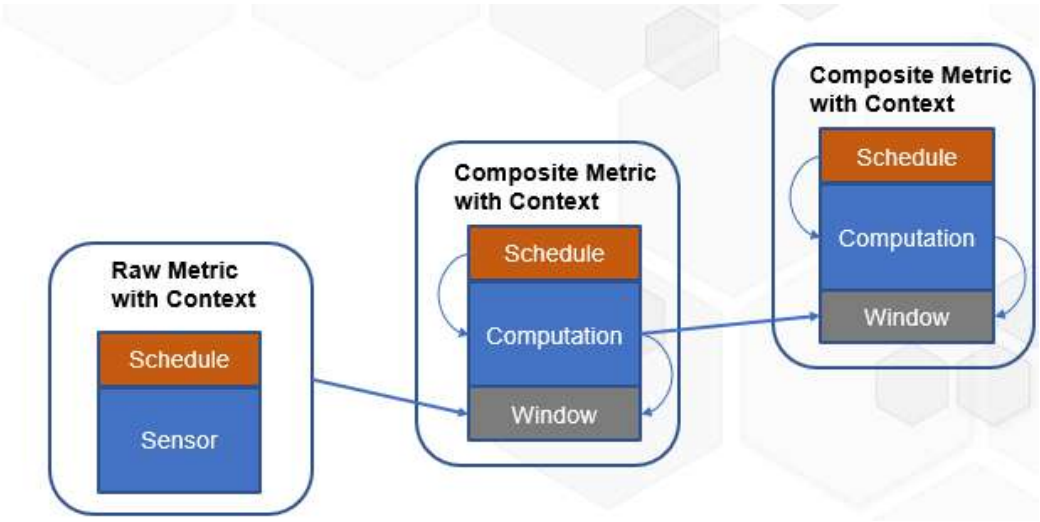

Figure 5 The definition of a computation chain covering metrics, contexts, schedules, windows and sensors

Another interesting feature covered in the metric domain concerns the fact that measurements can be grouped according to a specific criterion and computations can be performed in each group. The criterion currently covered for the measurement grouping concerns the actual Cloud level being concerned. As such, there are 6 Cloud/grouping levels:

- *PER_INSTANCE*: the measurements are collected for each instance of an application component
- *PER_HOST*: the measurements are collected from multiple instances of an application component situated in the same host (i.e., virtual or physical machine)
- *PER_ZONE*: the application/component measurements are collected per each (availability) zone
- *PER_REGION*: the application/component measurements are collected per each region, covering multiple zones
- *PER_CLOUD*: the application/component measurements are collected per each Cloud
- *GLOBAL*: this is the default level where all measurements are collected for the application or one of its components, irrespectively from its placement.

As for the definition of a target composite metric (e.g., participating in an SLO), a chain of computations is formulated (e.g., from a raw metric towards that target composite metric), a grouping can be propagated up until the highest possible level or might be aggregated to single values at a specific higher level from the current one. This can be highlighted through two examples. In the first example, shown in Figure 6, the average response time for an application is computed per host (*PER_HOST*) and all these host-based measurements are then aggregated at the highest level to compute a global average (*GLOBAL*). In the second example, shown in Figure 7, a more complicated scenario is covered where the measurements at the highest level of computation are still host-based. As it can be seen, raw measurement values are grouped per host and lead to computing aggregated measurements for two different composite metrics *CM1* and *CM2*, respectively. Then, the measurements of these two composite metrics are then aggregated at the highest level again per host in order to compute the values of the highest composite metric, *CM3*, which is, in fact, exploited to formulate a certain SLO.

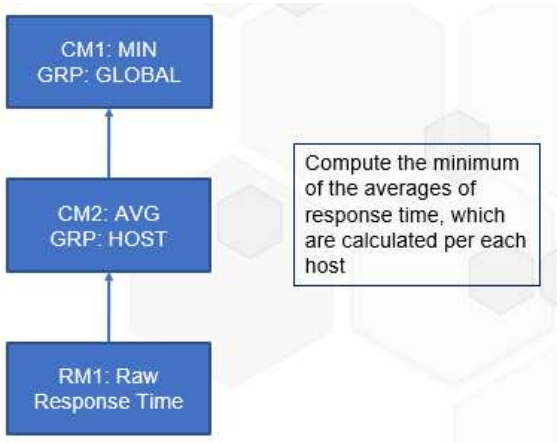

Figure 6 Aggregations of Response Time with different groupings

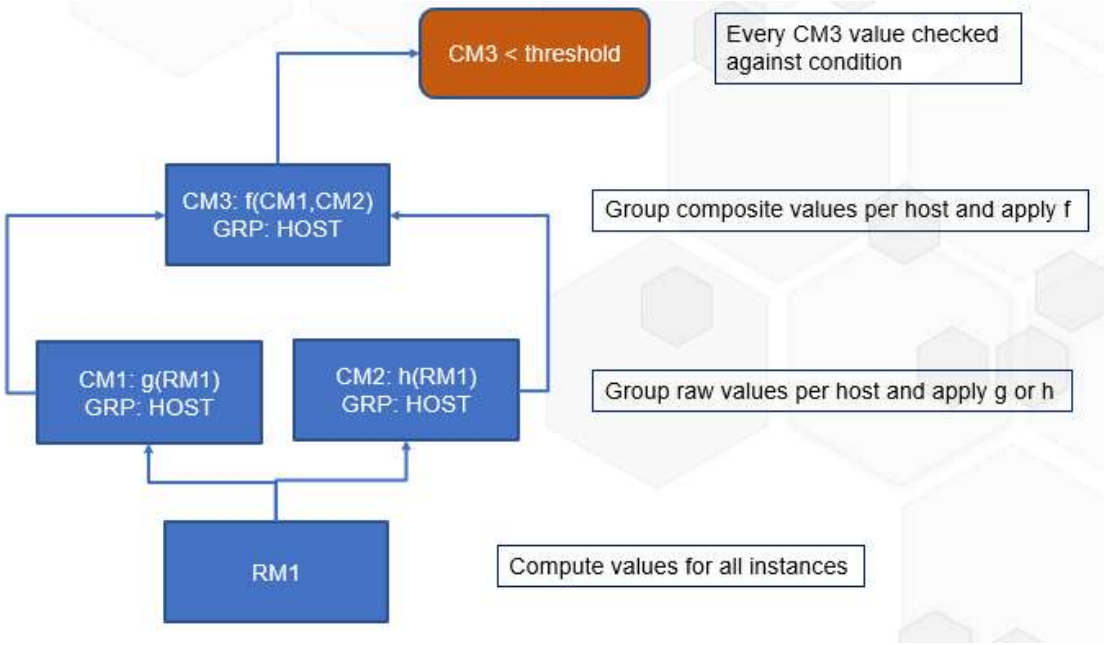

Figure 7 Host-based grouping all the way up to most complex metric

3.1.4.5.3.2 Issues

As CAMEL metric meta-model is partially derived from Complex Event Processing (CEP) languages, there is one potential issue which hampers its further adoption. In particular, a rich set of window pre-processing operators is not offered which leads to two main problems: (a) not all possible use-cases can be properly covered, something that will be showcased shortly afterwards, (b) in order to compensate for this, some partial solutions can be applied, which increase the number of hops in the (metric) computation chain by also specifying non-meaningful metrics so as to just cover the above gap by applying functions in composite metric formulas that map to these window operators.

In order to exemplify this situation, we will rely on a certain use-case from the MORPHEMIC project, the ICON one. In this use case, there is a need to compute a composite metric by calculating the minimum from the latest raw worker efficiency measurements, once these measurements are grouped per host, i.e., per each host in which the workers are deployed. As such, apart from the composite (application) metric, which we call *MinimumWorkerEfficiency*, we need to take into account also a raw metric one, called *WorkerEfficiency*. However, there is the above issue of the gap between these two metrics which comes with the pre-processing of the raw measurements: we need to first group these measurements per host before we can compute their minimum by taking the latest raw measurement from each group. To be noted that both metrics have computations triggered every 30 seconds and that for the composite metric we have a time-based window size of 30 seconds.

In order to address this gap, various solutions have been inspected but only one has been adopted, which nicely solves the problem and is still backwards compatible in terms of the metric domain in CAMEL.

First Solution. By considering the current version of CAMEL's metric meta-model, a first solution that could be applied would map to creating an intermediate (composite) metric between the other two, placed in the middle of the computation chain, that we call *UniqueWorkerEfficiency*. This metric is computed per host every 30 seconds and has a time-based window of 30 seconds. However, while being a composite metric, it does not have a computation formula (i.e., it is empty). The rationale is that once a raw measurement is sensed, it is the sole fitting its window for the respective host and that measurement would propagate to the *MinimumWorkerEfficiency* metric's window for the highest-level computation.

This first solution, while it could solve the metric meta-model's shortcoming, suffers from three major drawbacks. First, it introduces a new metric, the *UniqueWorkerEfficiency* one, thus enlarging the computation chain. Second, it is not proper and elegant to define a composite metric with no effective computation formula. Third, there is a need for

perfect synchronisation between the different computations at the different levels so as to compute the final, highest-level measurement properly without any delay or imprecision. On the other hand, this solution does not require changing the metric meta-model in CAMEL.

Second Solution. The second solution is similar with the previous one. Its sole difference is that it incorporates a new function that is utilised in the *UniqueWorkerEfficiency* metric's computation formula called LATEST/UNIQUE. The semantics of this function is that it computes the latest value in each window partition/grouping of the respective (composite) metric. While this solution solves the second from the above drawbacks, the other two still hold.

Third Solution. The rationale for the third solution is that we could incorporate in the metric computation formula functions which perform some kind of window pre-processing. In the current use-case examined, this would mean that we do not define any intermediate metric while the computation formula for the *MinimumWorkerEfficiency* metric becomes MIN(UNIQUE(*WorkerEfficiencyMetric*, *PER_HOST*)). The semantics is that first all the collected measurements are grouped per host and we take the latest from each group and then we compute the minimum over the filtered/selected measurements.

This third solution solves all the drawbacks of the first solution. It does not introduce any intermediate metric, as it only contains meaningful computation formulas and does not require perfect synchronisation. However, it does have other drawbacks. First, it requires extending the CAMEL editors (textual & web-based) in order to support the specification of the pre-processing operators/functions and their grouping-related parameters. Second, as the grouping criterion is now incorporated inside the window grouping/pre-processing function call, the explicit grouping constructs in CAMEL are more or less useless while this also means that there are two ways to achieve the same modelling goal which is not desirable in modelling languages (redundancy issue). Third, the semantics of these functions are not so clear to the modeller, who needs to carefully study CAMEL's documentation in order to know how to precisely use them. For instance, for the UNIQUE function, it is not clear whether it returns a single or a set of values. Finally, and more importantly, as the window pre-processing is applied according to a specific schedule, this means that the actual, overall window for a composite metric can become quite long between two different, sequential computation triggers of that metric measurements.

3.1.4.5.3.3 Adopted Solution

Inspired by the third solution and its drawbacks, especially the one that indicates that window pre-processing should not be mixed with metric computation, the final and fourth solution that has been actually adopted and enforced, attempts to achieve a clear separation between these two aspects. This final solution slightly enhances CAMEL's metric meta-model in order to apply such a separation. Its main rationale is that grouping as well as other window pre-processing operators need to be supported. Simple grouping stays, for backward compatibility reasons, in the composite metric context (so as it is right now) while more advanced forms of grouping as well as other major pre-processing operators become part of the window definition.

The modifications made in this CAMEL's meta-model are depicted in Figure 8, where existing classes are coloured with white colour, removed classes and attributes with red and new classes, enumerations, enumeration members, attributes and references with green. As it can be seen, a *Window* is associated with zero or more window processings (mapping to the *WindowProcessing* concept), which need to be applied in order (i.e., the first processing in the list is applied first, the second is applied second and so on). As it will be shown later on, the ability to apply multiple window processings can enable to simulate various operators in CEP languages. Any *WindowProcessing* has a specific type and comprises: zero or more grouping criteria as well as zero or more ranking criteria. There are three types of window processing that are envisioned:

- *GROUP*: in this type, we cluster measurements in groups according to the grouping criteria defined. We need to stress here that groupings formulated via a *WindowProcessing* are more advanced with respect to those simple ones that are supported by CAMEL 2.0 for two main reasons: (a) the grouping can be performed based on multiple criteria and not just one; (b) as indicated later on, the modeller can also utilise custom criteria instead of the basic ones

- *SORT*: in this type, the measurements are ranked according to the ranking criteria
- *RANK*: in this type, in each group formulated via the grouping criteria only the latest measurement is kept and then all measurements retained are sorted/ranked according to the ranking criteria

A criterion, either grouping or ranking, is represented via the *WindowCriterion* concept. Such a criterion concerns a specific metric, for which measurements are to be collected in a window and require some sort of pre-processing, and a specific type. There are 6 fixed types of a criterion (see *CriterionType* enumeration) mapping to the 5 original grouping levels covered in CAMEL plus the timestamp one, indicating that measurements can be grouped or ranked based on their timestamp (i.e., the time moment they were produced). In addition, there is one additional type mapping to a custom criterion type (see *CUSTOM* enumeration member). In this case, the modeller needs to specify the actual criterion via another attribute called *custom* mapping to a String. This gives the freedom to add special metadata to measurements that can then be utilised in order to specify grouping/sorting criteria over them. For instance, measurements could include a tag named *user* mapping to the related end-user of the multi-Cloud application. As such, such measurements could then be grouped into different clusters by utilising a grouping-based window processing having the *CUSTOM* criterion with the value "user" for the *custom* attribute. Finally, in case of a ranking criterion, it is required to also specify whether the ranking will be performed in an ascending or descending order according to the criterion being modelled.

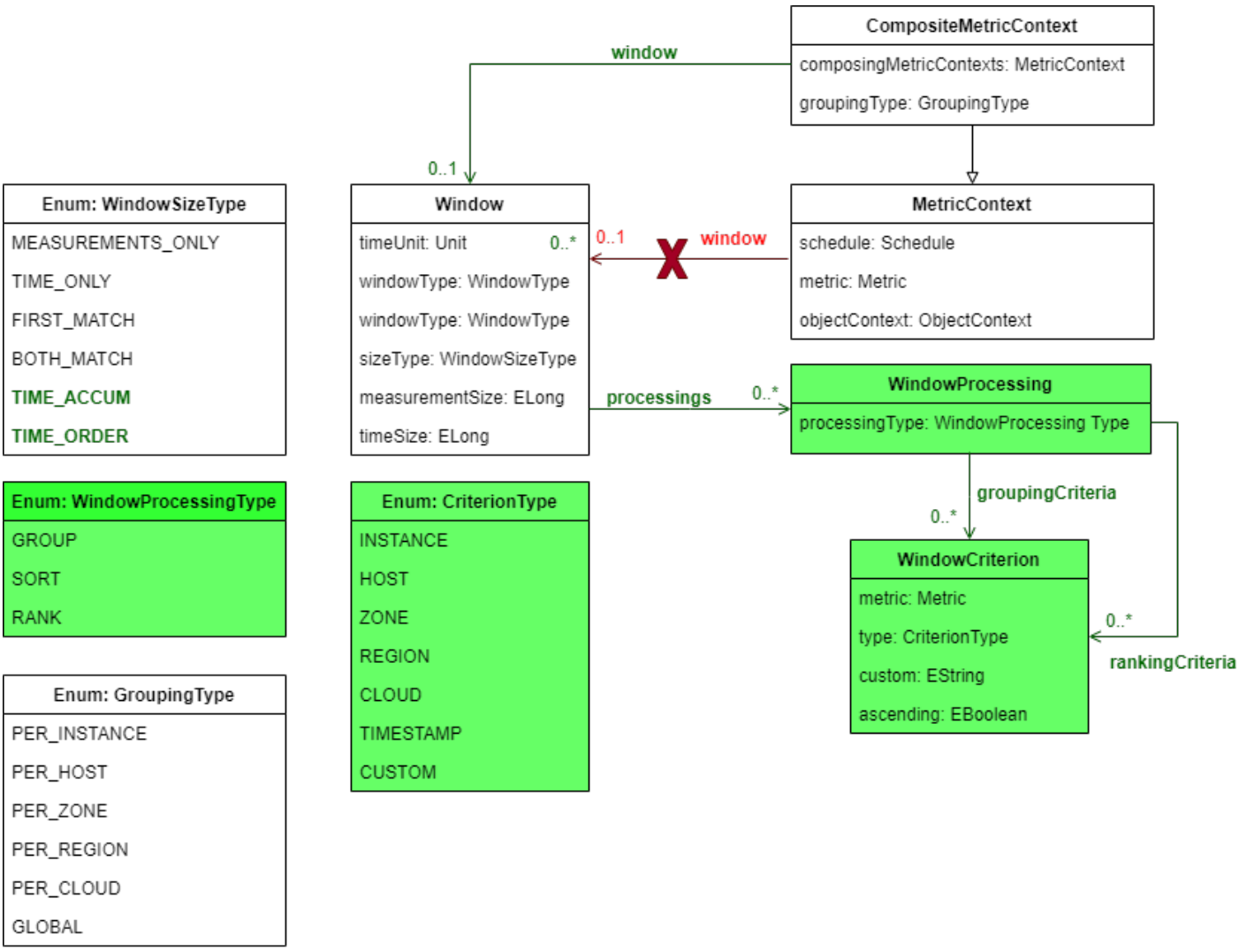

Figure 8 Window processing-related enhancement to CAMEL's metric meta-model

Another related extension in the metric meta-model concerned the *WindowSizeType* enumeration where two new window size types have been introduced:

- *TIME_ACCUM*: represents a special window that accumulates events until no event comes for a specific time period
- *TIME_ORDER*: a (sliding) window that keeps and ranks only some measurements for a specific time period relative to the measurements of the arrival time

Such an extension makes CAMEL even more complete with respect to the window size types that it can support. Furthermore, these new window size types could be regarded as special pre-processing operators based on their semantics.

The last change made to the CAMEL's metric meta-model concerned moving the *window* reference from *MetricContext* to *CompositeMetricContext* with the rationale that it is only meaningful to specify windows of measurements for composite metrics. This change is still backward compatible as there is no real CAMEL model structure change because the use of windows for raw metrics is obviously not realistically applicable in any possible or existing CAMEL model.

This new solution, while slightly extending CAMEL's metric meta-model, is quite elegant and addresses all the drawbacks of the previous solutions, which actually become its own merits. First, it does not lead to any increase in the length of the metric computation chain. On the contrary, it tends to decrease it. Second, it does not mix metric computation with window pre-processing. Third, it does not require any kind of synchronisation so as to achieve precise metric measurements. Finally, and more importantly, it strengthens CAMEL's metric meta-model by allowing to apply multiple different window pre-processing operations, thus enabling to cover further more use-cases in multi-cloud application modelling and monitoring.

The following table showcases how well-known window pre-processing constructs (or construct combinations) in the Esper's Event Processing Language (EPL) for CEP[7] can be covered via the use of CAMEL's extended metric meta-model:

Table 6 Way new CAMEL extension can cover some EPL's Window Pre-Processing Constructs

ESPER's EPL Window Pre-Processing Construct(s)	**Construct Semantics**	**CAMEL Coverage**
rank(groupCriteria, topMeasurementNum, rankingCriteria)	Only the latest measurement per group is kept, then all measurements are ranked and only a specific top number from them is finally kept	Apply a RANK window processing with all necessary grouping and ranking criteria in a window with a specific size (topMeasurementNum)
groupwin(groupCriteria)	Cluster the measurements according to the groupCriteria	Apply a GROUP window processing with all necessary grouping criteria
groupwin(groupCriteria)#length(num)	Cluster the measurements according to the groupCriteria and retain latest num measurements per group	Apply a GROUP window processing with all necessary grouping criteria on a sliding window with a specific size
unique(groupCriteria)	Cluster the measurements according to the groupCriteria and retain only the latest measurement per group	Apply a GROUP window processing with all necessary grouping criteria on a sliding window with a size of 1

[7] http://www.esper.espertech.com/release-5.2.0/esper-reference/html/epl_clauses.html

ESPER's EPL Window Pre-Processing Construct(s)	Construct Semantics	CAMEL Coverage
sort(topNum, sortCriteria)	Sort the measurements according to the sortCriteria and then keep the highest topNum ones	Apply a SORT window processing with all necessary ranking criteria on a sliding window with a size of topNum
groupwin(groupCriteria)#sort(topNum, sortCriteria)	Cluster the measurements according to the groupCriteria, then sort them in each group according to the sortCriteria and then keep the topNum elements in each group	Apply first a GROUP window processing with all grouping criteria and then a SORT window processing with all necessary sorting criteria on a sliding window with a specific size (topNum)
length(size)	A sliding window with size measurements	Define a sliding window with a specific size
length_batch(size)	A fixed window with size measurements	Define a fixed window with a specific size
time(timePeriod)	A sliding window with a time-based size	Define a sliding window with a specific timePeriod as its size
time_batch(timePeriod)	A fixed window with a time-based size	Define a fixed window with a specific timePeriod as its size
time_length_batch(timePeriod,size)	A fixed window where its size becomes fixed when either the timePeriod passes or a specific size of measurements arrives	Define a first-match window with both a time-based and a measurement-based size
time_accum(timePeriod)	A sliding window that accumulates events until no event is reached within a specific time period	Define a time-accumulating window with a time-based size
lastevent	Keep only the latest measurement	Just apply a sliding window with a size of 1
firstevent	Keep only the first measurement	Just apply a fixed window with a size of 1
firstunique(criteria)	Keep only the first from all measurements having the same value on the given criteria	Apply the GROUP window processing with all necessary grouping criteria on a fixed window with a size of 1
timeorder(timePeriod)	Order events that come out of order where each event is kept for a specific time period relative to its arrival time	Apply a sliding window with a TIME_ORDER size type
#length(num)#time(period)	A sliding window that retains at most num measurements that come within a specific period only	Apply a sliding window with a BOTH_MATCH size type with num as measurement size and period as the time size

ICON Use Case. By considering, now, the ICON use-case, i.e., our running example, we indicate how this CAMEL extension can lead to its proper modelling (see also the figure below for an excerpt of ICON use-cases CAMEL model focusing on the metric domain and especially this CAMEL extension). The *WorkerEfficiency*, as being a raw metric, does not require any kind of different treatment. So, the sole metric being affected is the *MinimumWorkerEfficiency*.

For this metric, we leave untouched its schedule, which should be 30 seconds. Its metric computation formula is also the same: MIN(*WorkerEfficiency*). However, its window has been altered. In particular, it now incorporates a grouping-based window processing (i.e., its type is GROUP). This processing includes one grouping criterion, which has as its type the value of PER_GROUP and applies to the *WorkerEfficiency* metric. Further, this window is sliding with a size of 1, which means that only the latest measurement is kept per each group formulated (per host). All these changes now enforce that the window of the *MinimumWorkerEfficiency* metric should be pre-processed such that its measurements, mapping to the *WorkerEfficiencyMetric*, are first grouped per host and only one measurement is always kept per group. This, now, precisely covers the original requirements of the ICON use case, which is an indication that this CAMEL extension is quite proper and sufficient to cover this as well as additional use cases.

```
metric model WPRM{
    measurable attribute Efficiency

    template EfficiencyTemplate{
        attribute Efficiency
        unit UnitTemplateCamelModel.UnitTemplateModel.Seconds
    }

    raw metric WorkerEfficiency{
        template EfficiencyTemplate
    }

    schedule MinEfficiencySched {
        interval 30
        time unit UnitTemplateCamelModel.UnitTemplateModel.Seconds
    }

    window MinWorkerWindow{
        type sliding
        size type measurements-only
        measurement size 1
        processing WorkerWinProc{
            type group
            grouping criterion Host{
                type host
            }
        }
    }

    composite metric MinWorkerEfficiency{
        template EfficiencyTemplate
        formula 'min(WorkerEfficiency)'
    }

    object context WorkerObjContext { component WPDM.Worker }

    composite metric context MinWorkEffContext{
        metric MinWorkerEfficiency
        schedule MinEfficiencySched
        window MinWorkerWindow
        object context WorkerObjContext
    }
}
```

Figure 9 Excerpt of ICON use-case's CAMEL model showcasing the window pre-processing CAMEL extension

3.2 Language Implementation

As already indicated in section 3.1.3, the new minor versions of CAMEL 3.0 were implemented by modifying and extending both the abstract and textual syntax of the very first draft version of CAMEL 3.0. The abstract syntax was enhanced by modifying CAMEL's meta-model in ECORE[8]. Such a modification included the incorporation of new classes, attributes and properties, the modification of existing ones plus the migration of the latter in different places without changing CAMEL's main structure with respect to CAMEL v2.0. It also involved the updating of the OCL rules that govern the semantic cross- and intra-model validation of CAMEL models. This updating took place inside the ECORE model of CAMEL through the use of the OCL[9] Editor[10] of the Eclipse Environment[11]. Out of the ECORE model of CAMEL, its respective domain code has been automatically produced by exploiting the automatic code generation facilities of the Eclipse Environment. Such code can then be exploited for the management of CAMEL models, where such a management involves tasks like CAMEL model creation, validation, storage and reading/parsing. Please also note that the (enhanced) CAMEL framework supports two encodings of CAMEL models: XML-based (XMI) and textual (conforming to CAMEL's textual syntax - see paragraph below). This means that models in any of these two encodings can be written or read by a computer program.

The textual syntax updating relied mainly on modifying the Xtext[12] model of CAMEL via the use of the Xtext Editor of the Eclipse Environment[13]. Please note that such an updating was deemed more suitable in comparison to the re-generation of the whole textual syntax (Xtext) model from scratch due to the effort required in the latter case to modify this model according to specific textual/formatting patterns that have been followed from the very first version of CAMEL. Apart from modifying the Xtext model, additional, lightweight modifications were performed also in those places related to the documentation of CAMEL where information about CAMEL classes is displayed when the user hovers over a specific CAMEL model element.

In the following, we supply relevant implementation links related to the latest minor version of CAMEL 3.0, i.e., v3.0.4:

- Source-code: https://gitlab.ow2.org/melodic/camel/-/tree/camel-3.0.4
- Meta-model: https://gitlab.ow2.org/melodic/camel/-/blob/camel-3.0.4/camel/camel/model/camel.ecore
- Documentation: https://confluence.7bulls.eu/display/MOR/CAMEL+3.0
- Textual Editor Installation Instructions: https://confluence.7bulls.eu/display/MEL/%5BCAMEL%5D+Camel+2.0+Eclipse+%28oxygen%29+editor+installation

4 Metadata Schema Extensions

4.1 Conceptual Analysis

In MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling* [1], we provided an extensive update of the vocabulary entitled *Metadata Schema* (MDS) which was initially introduced in the frame of the Melodic project [23]. This schema is quite important for the modelling framework used in MORPHEMIC for driving, updating, and maintaining the deployment of multi-component applications that exploit multi-Cloud and edge resources. Specifically, MDS provides a number of classes and data and object properties that correspond to semantics used for describing requirements, constraints, and offerings' characteristics in multi-Cloud placement decisions. This kind of semantic description constitutes the formal means for extending the CAMEL language with appropriate concepts related to big data management, the optimisation of the placement of processing jobs, and access control in multi-Cloud environments. In this way, CAMEL does not have to incorporate any hardcoded terms for expressing

[8] https://www.eclipse.org/modeling/emf/
[9] https://www.omg.org/spec/OCL/
[10] https://projects.eclipse.org/projects/modeling.mdt.ocl
[11] www.eclipse.org
[12] https://www.eclipse.org/Xtext/
[13] https://www.eclipse.org/Xtext/documentation/308_emf_integration.html

e.g., placement constraints but instead it exploits the MDS easily extensible and reusable vocabulary. As it was detailed in MORPHEMIC deliverable D1.1 *Data, Cloud Application & Resource Modelling*, MDS comprises the *Application Placement*, *Big Data* and *Context Aware Security* models that group a number of classes and properties to be used for defining where a certain big data application should be placed; what are the unique characteristics of the data artefacts that needs to be processed; and what are the contextual aspects that may be used for restricting the access to the sensitive data.

As part of the WP1 work, we supported the process of application modelling of the use case providers based on which we provided additional artefacts or changes required according to their modelling needs. Here, we briefly discuss four important changes to MDS. The first change involved the simplification and reduction of certain MDS properties in order to avoid using concepts or properties variants that can more easily be expressed with the use of operators in CAMEL 3.0. For example, instead of having two different data properties for denoting integers that capture the minimum (*hasMinNumberofCores*) and maximum number (*hasMaxNumberofCores*) of CPU cores available or requested, respectively, we use just one i.e., *hasNumberofCores* and let CAMEL with proper operators express the maximum or minimum thresholds. In the following table, we provide the changes made in such properties. We mention only the relevant classes (with their parent classes as a path) and the properties affected by the change.

Table 7 MDS Updates (removing properties' thresholds)

Class Taxonomy Levels	Properties	Description
IaaS/Processing/CPU		This class refers to IaaS resources that use Central Processing Units (CPUs) for carrying out software instructions that specify the basic arithmetic, logical, control and input/output (I/O) operations.
	hasNumberofCores	This property denotes an integer that captures the number of CPU cores available or requested.
	hasMinNumberofCores	Removed
	hasMaxNumberofCores	Removed
IaaS/Processing/Memory/ ProcessingMemory/RAM/ TotalMemory		This subclass captures the desired or offered value of the virtualised memory storage dedicated for frequent program instructions.
	hasSize	This property associates the Total Memory class with an integer that represents the amount of memory capacity required or offered.
	hasMin	Removed
	hasMax	Removed
IaaS/Processing/ Accelerator/GPU		This class refers to IaaS resources that use graphics processing units (GPUs), i.e. specialized electronic circuits initially designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer.
	hasConcurrentWorkgroups	This property denotes an integer that represents the work-groups that may be simultaneously executed on compute units supported by a certain GPU.
	hasMaxConcurrentWorkgroups	Removed
	hasNumberofCores	This property denotes an integer that captures the number of GPU cores available or requested.

Class Levels Taxonomy	Properties	Description
	hasMinNumberofCores	Removed
	hasMaxNumberofCores	Removed
IaaS/NetworkEntity /HardwareNetworkEntity /NetworkNode /Interface		This class models a point of interconnection between a device and a private or public network associated to an IaaS resource that may involve Cloud or Edge nodes.
	hasNumberOfInterfaces	This property expresses the amount of network interfaces an IaaS offering is requested to have or already has.
	hasMinNumberOfInterfaces	Removed
	hasMaxNumberOfInterfaces	Removed
IaaS/Storage/Capacity		Removed
	hasMin	Removed
	hasMax	Removed
IaaS/Storage		
	hasCapacityUnit	The hasUnit property was moved and renamed as a data property of the Storage class. It refers to a string that indicates the unit for measuring the storage capacity offered or required.
	hasCapacity	This data property is introduced to capture the required or offered storage capacity provided in an IaaS offering. This property can be used with the appropriate CAMEL 3.0 operator to define maximum and minimum threshold for the storage capacity.

The second change involved the necessary additional artefacts to support the modelling of Bring-Your-Own-Node (BYON) in MORPHEMIC deployments. This addition is related to the IS-Wireless use case where it was important to describe capabilities or requirements of BYON devices in the pilot demonstration. Therefore, we have extended the IaaS class of the MDS *Application Placement Model* as it is presented in the table below.

Table 8 MDS Updates (BYON-related)

Class Levels Taxonomy	Properties	Description
IaaS/BYON		This class refers to certain IaaS resources that can be introduced in a processing topology by the operator or the user of a certain cloud application. Bring-Your-Own-Node (BYON) can be considered as a «mobile» hosting resource with limited but not negligible processing capacity that can be easily installed, at any given time, near other important resources in an use case (e.g. 5G antenna, IoT device etc.)
	hasBYONName	This data property expresses as a string the name of the BYON to be used in a cloud application deployment.
	hasBYONStatus	This Boolean property indicates whether or not a certain BYON is active, connected and ready to host cloud application components.

Class Taxonomy Levels	Properties	Description
	hasCPU	This object property associates BYON class with the CPU class (i.e., range) in order to extend the properties of the Central Processing Units (CPUs) that BYON is equipped with (e.g., hasNumberofCores).
	hasGPU	This object property associates BYON class with the GPU class (i.e., range) to refer to properties of the graphics processing units (GPUs) that BYON can bring in a hosting topology (e.g., hasNumberofCores, hasGPUtype)
	hasMemory	This object property associates BYON class with the Memory class (i.e., range) in order to use the properties of this class to express the memory capacity of the BYON at hand.
	hasStorage	This object property associates BYON class with the Storage class (i.e., range) in order to use the properties of this class to express the persistence capacity of the BYON at hand.

The third change involved the introduction of additional properties required under the *Accelerator* class to address all the modelling requirements of the MORPHEMIC pilots. Therefore, in the table below we present the new properties added for *FPGA* and *GPU* classes.

Table 9 MDS Updates (Accelerator-related)

Class Taxonomy Levels	Properties	Description
IaaS/Processing/Accelerator		This class refers to application specific hardware designed or programmed to compute operations faster than a general-purpose computer processor.
IaaS/Processing/ Accelerator/GPU		This class refers to IaaS resources that use graphics processing units (GPUs), i.e. specialized electronic circuits initially designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer.
	hasGPUNumber	This data property indicates an integer that reveals the number of GPUs that are required in a specific use case.
IaaS/Processing/ Accelerator/FPGA		This class refers to IaaS resources that use field programmable gate arrays (FPGAs), as integrated circuits made to be configured by the user after manufacturing.
	hasFPGANumber	This data property indicates an integer that reveals the number of FPGAs that are required in a specific use case.

The fourth change involved the introduction of an additional class used to annotate certain configuration related metrics in a CAMEL metric type model, which convey for example the nodes' busy statuses (i.e., working or idle). In the same context small additions were made in other parts of the IaaS class. Therefore, in the table below we present the new classes.

Table 10 MDS Metrics-related Updates

Class Taxonomy Levels	Properties	Description
Application placement model / UtilityNotions		This class is used for accommodating properties that express configuration related metrics that are useful for the appropriate application deployment management.
	BusyInstanceMetric	This boolean property is used to annotate the Metrics in CAMEL metric type model, which are responsible for conveying the busy status of the application nodes.
	Cardinality	This boolean property is used to define the number of resources used for certain application or application component
	Unmovable	This boolean property is used to define a constraint on certain application component or in its hosting resource on whether or not is allowed to be moved during a reconfiguration.
	IaaS/Processing/Memory/ProcessingMemory/RAM/hasUsedMemoryPercentage	This property associates the RAM class with a value expressed in percentage that denotes the amount of used memory in the virtualised resource.
	IaaS/Processing/Memory/ProcessingMemory/RAM/hasFreeMemoryPercentage	This property associates the RAM class with a value expressed in percentage that denotes the amount of unused memory currently available by the virtualised resource.
ContextAwareSecurityModel/SecurityContextElement/Object/SoftwareArtefact	*softwareCategory*	String property describing the variant or the hardware of the application component.
	derivedCategory	This string property is similar to the softwareCategory; however, the *derivedCategory* is inserted by MORPHEMIC profiling process.
	language	String property representing the main programming language of the application component
	repositoryLocation	It indicates the link/URL from where the application component code can be downloaded
	softwareDescription	String property providing general information on the application component.

4.2 Implementation

The MDS was developed and extended in iterations, starting with an analysis of the available vocabularies and ontologies related to data-aware multi-Cloud computing [24] and continued with an investigation of the advanced requirements of the MORPHEMIC use cases [4]. For the representation of a comprehensible overview of MDS, we used a free, HTML5-compliant mind mapping webapp[14] with Cloud support. The detailed mind map produced for the MDS can be used for an easier walkthrough of the Schema's main aspects and extensions and can be found here[15]. MDS was also serialized in XMI[16]. The serialization used was decided based on the fact that this vocabulary should be properly specified in one Ecore-based language encoding form so as to enable the re-use of its elements for annotating

[14] https://app.mindmapmaker.org/
[15] https://www.morphemic.cloud/mds2022.png
[16] http://www.omg.org/spec/XMI/

CAMEL models. We note that the reader may find the serialization of the complete model[17]. This serialization took place by using the Metadata Schema editor which a graphical web-based tool that has been developed in the context of the Melodic project, in order to enable the creation, modification and management of MDS. This editor has been implemented in Java and is publicly available[18].

5 Use-Case Modelling

In order to showcase the main extensions made to CAMEL and MDS, we will rely on an existing use-case of the MORPHEMIC project. This use-case concerns the E-Brain Science, offered by CHUV. In this use-case, the focus is on the image pre-processing pipeline which involves executing a workflow for pre-processing neuroimaging data. The purpose of such a workflow is to convert neuroimaging data into the appropriate format, segment the formatted data accordingly and extract the main brain features.

In order to support the execution of the pre-processing workflows, CHUV relies on the Proactive Workflow Scheduler, a workflow engine able to deploy and execute workflows. As such, the architecture of the use-case follows the master-slave pattern where the master is the scheduler and the workers are components which realise the functionality of the workflow tasks. To enable flexibility in task realisation, CHUV has decided to realise one generic slave component, which is able to fulfil the functionality of all needed tasks. This component originally had a container-based form and was exploiting only the CPU of the respective underlying machine. However, through the cooperation of CHUV and InAccel, a new component form was created, relying on the existence of FPGA-based underlying resources. As such, the whole, container-based application of CHUV is polymorphic, thanks to this development, which enabled to have two forms of the same application component, the slave one.

The master component, the ProActive Scheduler one, has a VM-based form, it is deployed via script-based configuration and has the following requirements:

- An ubuntu v18 image should be utilised for the component deployment
- There should be two to 4 cores available in the underlying VM
- There should be 8100 to 10072 MB of main memory in the underlying VM
- The VM should be situated in a public Cloud
- There should be always one instance of this component which should never be migrated from one VM to another one (e.g., due to application reconfiguration reasons)

The slave component, as indicated previously, has two forms, one simple, container-based and another both container- and FPGA-based. Irrespectively of the form, the following requirements apply to this component:

- It should be deployed on a public Cloud
- It can be horizontally scaled from 1 to 10 instances at most

Further, there exist requirements which are specific to a certain form. For the simple, container-based form, the following requirements apply:

- An ubuntu v18 image should be utilised for the component deployment, same as the one for the Pro-Active Scheduler/master component
- The same resource requirements as for the master component also apply to this form of the slave component (number of cores between 2 and 4 and main memory size between 8100 and 10072 MB)

On the other hand, the following requirements apply to the container- and FPGA-based form of the slave component:

- A different ubuntu v18 image should be used for the component form's deployment
- There should be exactly 8 cores available to the container
- There should be 124928 MB of main memory available to the container

[17] https://gitlab.ow2.org/melodic/camel/-/tree/morphemic-rc2.0/metadata-schema/current
[18] https://bitbucket.7bulls.eu/projects/MEL/repos/metadata-schema/browse/muse

- There should be one FPGA available in the underlying VM

Thanks to the new version of CAMEL, it is not only possible to specify component-based requirements that apply to all component configuration forms but also specific requirements that apply to a certain component form. The respective deployment sub-model of the CAMEL model of this CHUV use-case is depicted in the following figure.

CHUV.camel

```
camel model ApplicationCHUVBrainTwoCompUtily13_gr {
    application ApplicationCHUVBrainTwoCompUtily12_gr { version '0.4' }

    // Deployment model
    deployment model SchedulerAndWorkersDeployment{
        software Component_Scheduler {
            [MetadataModel.MELODICMetadataSchema.UtilityNotions.UtilityRelatedProperties.Unmoveable]
            requirements SchedulerRequirementSet required host WM_SchedulerHostReq
            script configuration BrainProcConfiguration {
                download 'x' install 'x' configure 'x'
            }
            attribute category_variant_vm [MetadataModel.MELODICMetadataSchema.ContextAwareSecurityModel.SecurityContextElement.Object.SoftwareArtefact.softwareCategory] : string "VM"
            attribute category_hardware_cpu [MetadataModel.MELODICMetadataSchema.ContextAwareSecurityModel.SecurityContextElement.Object.SoftwareArtefact.softwareCategory] : string "CPU"
            provided communication ComponentSchedulerUI port 80 provided communication ComponentSchedulerNode port 64738
        }

        software Worker{
            requirements WorkerRequirementSet required host WM_WorkerHostReq
            script configuration WorkerT1Configuration{
                download 'x' install 'x' configure 'x' start 'x'
                requirements CPU_Reqs {
                    resource SAW_Requirement.WorkerReqs image SAW_Requirement.Ubuntu18
                }
                attribute category_t1_docker [MetadataModel.MELODICMetadataSchema.ContextAwareSecurityModel.SecurityContextElement.Object.SoftwareArtefact.softwareCategory] : string "DOCKER"
                attribute category_t1_hardware [MetadataModel.MELODICMetadataSchema.ContextAwareSecurityModel.SecurityContextElement.Object.SoftwareArtefact.softwareCategory] : string "CPU"
            }
            script configuration WorkerT2Configuration{
                download 'x' install 'x' configure 'x' start 'x'
                requirements FPGA_Reqs{
                    resource SAW_Requirement.WorkerReqs_FPGA image SAW_Requirement.Ubuntu18_FPGA
                }
                attribute category_t2_docker [MetadataModel.MELODICMetadataSchema.ContextAwareSecurityModel.SecurityContextElement.Object.SoftwareArtefact.softwareCategory] : string "DOCKER"
                attribute category_t2_hardware [MetadataModel.MELODICMetadataSchema.ContextAwareSecurityModel.SecurityContextElement.Object.SoftwareArtefact.softwareCategory] : string "FPGA"
            }
            required communication WorkerPort port 64738 mandatory
        }

        communication SchedulerToWorker from  Worker.WorkerPort to Component_Scheduler.ComponentSchedulerNode

        requirements SchedulerRequirementSet{
            resource SAW_Requirement.SchedulerReqs horizontal scale SAW_Requirement.HorizontalScaleScheduler provider SAW_Requirement.PublicRequirement image SAW_Requirement.Ubuntu18
        }

        requirements WorkerRequirementSet{
            horizontal scale SAW_Requirement.HorizontalScaleWorker provider SAW_Requirement.PublicRequirement
        }
    }
```

Figure 10 The deployment model of the CHUV use-case

Please notice that apart from this new CAMEL feature, CAMEL's extension to apply inequality constraints on resource/platform attributes annotated by MDS has been also applied. Further, for this application, some MDS properties have been consolidated to have a singular form instead of max and min-based forms. For example, in the case of the CPU-based resource requirements for the master component, we have the definition of a single attribute, *coresScheduler*, which has now a range of values that restrain it while it is semantically annotated with the *hasNumberOfCores* property of the CPU concept in the MDS. Thus, instead of having to define two attributes in the resource requirement mapping to the MDS properties *hasMinNumberofCores* and *hasMaxNumberofCores*, we define just one. This leads also to a smaller CAMEL model content and thus reduces the modelling effort required. Mathematically, we have that now we can specify: 2 <= *hasNumberOfCores* <= 4 while previously we had to define: *hasMinNumberofCores* = 2 and *hasMaxNumberofCores* = 4. This CAMEL & MDS extension along with all the application requirements are shown in the following figure, depicting the respective requirement sub-model of the CAMEL model of the CHUV use-case.

CHUV.camel

```
requirement model SAW_Requirement{
    resource requirement SchedulerReqs{
        feature coresScheduler{
        [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU]
            attribute coresScheduler [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU.hasNumberofCores]: [int 2, int 4]
        }
        feature ramScheduler{
        [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.Memory.ProcessingMemory.RAM]
            attribute ramScheduler [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.Memory.ProcessingMemory.RAM.TotalMemory]: [int 8100, int 10072]
        }
    }

    resource requirement WorkerReqs{
        feature coresWorkerSmall{
        [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU]
            attribute minCoresWorkerSmall [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU.hasNumberofCores]: [int 2, int 4]
        }
        feature ramWorkerSmall{
        [ MetadataModel.MELODICMetadataSchema.Application_Placement_Model.IaaS.Processing_.Memory.ProcessingMemory.RAM]
            attribute ramWorkerSmall [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.Memory.ProcessingMemory.RAM.TotalMemory]: [int 8100, int 10072]
        }
    }
    resource requirement WorkerReqs_FPGA{
        feature cpu_worker_fpga{
        [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU]
            attribute coresWorkerSmall_FPGA [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU.hasNumberofCores]: int 8
        }
        feature ram_worker_fpga{
        [ MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.Memory.ProcessingMemory.RAM]
            attribute ram_worker_size_FPGA [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.Memory.ProcessingMemory.RAM.TotalMemory]: int 124928
        }
        feature fpga_worker_fpga{
        [ MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.Accelerator.FPGA ]
            attribute fpga_worker_size_FPGA [MetadataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.Accelerator.FPGA.hasFPGANumber]: int 1
        }
    }

    horizontal scale requirement HorizontalScaleWorker [1,10] horizontal scale requirement HorizontalScaleScheduler [1,1]

    image requirement Ubuntu18 ['chuv-pa-scheduler-upd12-ny'] image requirement Ubuntu18_FPGA [ 'MORPH-FPGA' ]

    slo ReconfigurationRule1 constraint SAWConstraintModel.ReconfigureIfPendingTasks slo ReconfigurationRule2 constraint SAWConstraintModel.ReconfigureIfAvgRunTaskDurLong

    optimisation requirement CHUV_maxUtility{ variable SAWMetricModel.CHUV_Utility }

    provider requirement PublicRequirement{ cloud type public }
}
```

Figure 11 The requirement model of the CHUV use-case

Apart from resource, image and horizontal scale requirements, this CHUV application has two SLOs and one optimisation requirement. The first SLO indicates that there should be no pending tasks to execute. Thus, if such tasks exist, then we should reconfigure the application in order to have the computing ability to rapidly execute them. The second SLO indicates that the average running tasks' estimated finish time should be less or equal to 10 seconds. Otherwise, the application would need to be reconfigured to provide additional resources to the slave component instances in order to speed up the execution of their tasks. Finally, the utility to be maximised is given by the following formula:

$$\frac{1}{1 + e^{AvgPTEstDur * ceil\left(\frac{PTNumber - WorkerCores * WorkerCardinality}{WorkerCores * WorkerCardinality}\right) - BufferTime}}$$

where *AvgPTEstDur* is the estimated average duration of pending tasks, *PTNumber* is the number of pending tasks, *WorkerCores* is the number of cores to be given for the slave component and *WorkerCardinality* is the number of instances of the slave component to create and finally the *BufferTime*, related to the margin between the current duration and the overall workflow deadline, is a composite metric calculated as follows: (*Deadline* - *CurrentDur*) * 0.5 where *Deadline* is the overall workflow deadline and *CurrentDur* is the current duration in the workflow execution.

While the above SLOs and optimisation requirements could be already captured by CAMEL 2.0, a specific measurement need applied requiring also the usage of the new CAMEL extension related to window pre-processing.

The respective CAMEL model part covering the aforementioned SLOs and optimisation requirements as well as all the other CAMEL elements needed to specify them is depicted in the following figures. The last figure shows the constraint model which is related to metric contexts defined in the metric sub-model as well as the SLOs as being defined in the requirement model.

CHUV.camel

```
metric model SAWMetricModel{
    //DEADLINES: reference workflow duration and (time to) deadline adapted to current time
    measurable attribute Ref_workflow_duration sensors [SAWMetricModel.Ref_workflow_duration_Sensor]
    measurable attribute Deadline sensors [SAWMetricModel.Deadline_Sensor]
    measurable attribute BufferTime

    //RUNNING TASK
    measurable attribute avgRUNtaskEstFinishTime sensors [SAWMetricModel.avgRUNtaskEstFinishTime_Sensor]

    //PENDING TASK
    measurable attribute PENDtaskNumber  sensors [SAWMetricModel.PENDtaskNumber_Sensor]
    measurable attribute avgPENDtaskEstDuration sensors [SAWMetricModel.avgPENDtaskEstDuration_Sensor]
    measurable attribute DeltaTime_actual

    //NODES
    measurable attribute cores  [MetadataModel.MELODICMetadataSchema.Application_Placement_Model.IaaS.Processing_.CPU.hasNumberofCores]
    measurable attribute cardinality

    sensor Ref_workflow_duration_Sensor { config '' }

    sensor Deadline_Sensor { config '' }

    sensor avgRUNtaskEstFinishTime_Sensor { config '' }

    sensor sumPENDtaskEstDuration_Sensor { config '' }

    sensor avgPENDtaskEstDuration_Sensor { config '' }

    sensor PENDtaskNumber_Sensor { config ''}

    template Ref_workflow_durationTemplate{
        attribute Ref_workflow_duration
        unit UnitTemplateCamelModel.UnitTemplateModel.Seconds
        value type TypeTemplateCamelModel.TypeTemplateModel.ZeroToPositiveInfinityInteger
    }

    template DeadlineTemplate{
        attribute Deadline
        unit UnitTemplateCamelModel.UnitTemplateModel.Seconds
        value type TypeTemplateCamelModel.TypeTemplateModel.ZeroToPositiveInfinityInteger
    }

    template avgRUNtaskEstFinishTimeTemplate{
        attribute avgRUNtaskEstFinishTime
        unit UnitTemplateCamelModel.UnitTemplateModel.Seconds
        value type TypeTemplateCamelModel.TypeTemplateModel.ZeroToPositiveInfinityInteger
    }
```

Figure 12 CHUV's metric model - PART I

CHUV.camel

```
template avgPENDtaskEstDurationTemplate{
    attribute avgPENDtaskEstDuration
    unit UnitTemplateCamelModel.UnitTemplateModel.Seconds
    value type TypeTemplateCamelModel.TypeTemplateModel.ZeroToPositiveInfinityInteger
}

template PENDtaskNumberTemplate{
    attribute PENDtaskNumber
    unit UnitTemplateCamelModel.UnitTemplateModel.Simulations
    value type TypeTemplateCamelModel.TypeTemplateModel.ZeroToPositiveInfinityInteger
}

raw metric Ref_workflow_durationRawMetric{ template Ref_workflow_durationTemplate }

raw metric DeadlineRawMetric{ template DeadlineTemplate }

raw metric avgRUNtaskEstFinishTimeRawMetric{ template avgRUNtaskEstFinishTimeTemplate }

raw metric avgPENDtaskEstDurationRawMetric{ template avgPENDtaskEstDurationTemplate }

raw metric PENDtaskNumberRawMetric{ template PENDtaskNumberTemplate }

schedule SAWTimeSchedule { interval 120 time unit UnitTemplateCamelModel.UnitTemplateModel.Seconds }

window SAWWindow {
    type fixed
    size type time-only
    time size 120 time unit UnitTemplateCamelModel.UnitTemplateModel.Seconds
}

object context SchedulerMetricContext{ component SchedulerAndWorkersDeployment.Component_Scheduler }

object context WorkerMetricContext{ component SchedulerAndWorkersDeployment.Worker }

raw metric context Ref_workflow_durationContext{
    metric Ref_workflow_durationRawMetric
    sensor SAWMetricModel.Ref_workflow_duration_Sensor
    schedule SAWTimeSchedule
    object context WorkerMetricContext
}

raw metric context DeadlineContext{
    metric DeadlineRawMetric
    sensor SAWMetricModel.Deadline_Sensor
    schedule SAWTimeSchedule
    object context WorkerMetricContext
}
```

Figure 13 CHUV's metric model - PART II

CHUV.camel

```
raw metric context avgRUNtaskEstFinishTimeContext{
    metric avgRUNtaskEstFinishTimeRawMetric
    sensor SAWMetricModel.avgRUNtaskEstFinishTime_Sensor
    schedule SAWTimeSchedule
    object context WorkerMetricContext
}

raw metric context avgPENDtaskEstDurationContext{
    metric avgPENDtaskEstDurationRawMetric
    sensor SAWMetricModel.avgPENDtaskEstDuration_Sensor
    schedule SAWTimeSchedule
    object context WorkerMetricContext
}

raw metric context PENDtaskNumberContext{
    metric PENDtaskNumberRawMetric
    sensor SAWMetricModel.PENDtaskNumber_Sensor
    schedule SAWTimeSchedule
    object context WorkerMetricContext
}

template CoresTemplate{
    attribute cores
    unit UnitTemplateCamelModel.UnitTemplateModel.Cores
    value type TypeTemplateCamelModel.TypeTemplateModel.ZeroToPositiveInfinityInteger
}

variable WorkerCores{
    template CoresTemplate component SchedulerAndWorkersDeployment.Worker
}

variable ActWorkerCores{
    template CoresTemplate component SchedulerAndWorkersDeployment.Worker
    current-config
}

template CardinalityTemplate{
    attribute cardinality
    unit UnitTemplateCamelModel.UnitTemplateModel.Instances
    value type TypeTemplateCamelModel.TypeTemplateModel.ZeroToPositiveInfinityInteger
}

variable WorkerSmallCardinality{
    template CardinalityTemplate
    component SchedulerAndWorkersDeployment.Worker
}
```

Figure 14 CHUV's metric model - PART III

CHUV.camel

```
        variable WorkerSmallActualCardinality{
            template CardinalityTemplate component SchedulerAndWorkersDeployment.Worker
            current-config
        }

          // Buffer time (time between reference_duration time and deadline)
        template BufferTimeTemplate{
            attribute BufferTime unit UnitTemplateCamelModel.UnitTemplateModel.Seconds
            value type TypeTemplateCamelModel.TypeTemplateModel.ZeroToPositiveInfinityInteger
        }

        composite metric BufferTimeMetric{
            template BufferTimeTemplate
            formula "(DeadlineRawMetric - Ref_workflow_durationRawMetric)*0.5"
        }

        composite metric context  BufferTimeContext{
            metric BufferTimeMetric grouping global window SAWWindow
            schedule SAWTimeSchedule
            composing contexts [DeadlineContext,Ref_workflow_durationContext]
        }

        // DELTA_TIME_ACTUAL
        template DeltaTimeActTemplate{
            attribute DeltaTime_actual unit UnitTemplateCamelModel.UnitTemplateModel.Seconds
            value type TypeTemplateCamelModel.TypeTemplateModel.ZeroToPositiveInfinityInteger
        }

        composite metric DeltaTimeActMetric{
            template DeltaTimeActTemplate
            formula 'avgPENDtaskEstDurationRawMetric*ceil(PENDtaskNumberRawMetric/(ActWorkerCores*WorkerSmallActualCardinality))'
        }

        composite metric context DeltaTimeActContext{
            metric DeltaTimeActMetric grouping global window SAWWindow
            schedule SAWTimeSchedule
            composing contexts [avgPENDtaskEstDurationContext,PENDtaskNumberContext]
        }

    //  utility function
        variable CHUV_Utility{
            template MetricTemplateCamelModel.MetricTemplateModel.UtilityTemplate
            formula '1/((1+exp(-BufferTimeMetric + avgPENDtaskEstDurationRawMetric*ceil((PENDtaskNumberRawMetric-WorkerCores*WorkerSmallCardinality)/(WorkerCores*WorkerSmallCardinality)))))'
        }
    }
```

Figure 15 CHUV's metric model - PART IV

```
    constraint model SAWConstraintModel{
        metric constraint ReconfigureIfPendingTasks : [SAWMetricModel.DeltaTimeActContext] > 0.0
        metric constraint ReconfigureIfAvgRunTaskDurLong : [SAWMetricModel.avgRUNtaskEstFinishTimeContext] > 10.0
        logical constraint Reconfigallconstrains : and (ReconfigureIfPendingTasks,ReconfigureIfAvgRunTaskDurLong)
    }
}
```

Figure 16 The constraint model of the CHUV use-case

6 Conclusions & Future Work

This deliverable explained how CAMEL has evolved towards its new version 3.0. It indicated which changes and additions were conducted on CAMEL in order to support polymorphic application modelling & adaptation as well as other features of the MORPHEMIC platform. One of the most important change requirements that was accommodated is backwards compatibility. In this sense, any CAMEL model conforming to its previous version (2.0) also conforms to CAMEL 3.0. In this way, any use-case/commercial partner can easily migrate his/her models to this new CAMEL version by just adding new elements in them such that his/her applications become polymorphic and can thus be properly managed by the MORPHEMIC platform. Another interesting extension of CAMEL related to window-preprocessing, a capability to group, filter and potentially sort measurements on the fly before they are processed in terms of some statistical function (so as to compute measurements for a respective composite metric). In this way, now CAMEL supports even more advanced scenarios for (multi-)Cloud application monitoring.

The deliverable also shed light to changes and updates made to MDS, the metadata schema of the MORPHEMIC platform. Such changes related to CAMEL extensions (as MDS plays complementary role to CAMEL), to supporting platform features as well as the MORPHEMIC use-cases. One important new extension (with respect to the ones reported in MORPHEMIC D1.1 *Data, Cloud Application & Resource Modelling* [1]) related the coverage of features of BYON nodes which enables to utilise such resources in cloud application provisioning by the MORPHEMIC platform. In particular, resource requirements can now be posed in CAMEL models which can enable to filter the available BYON nodes and select the one that best fits the current Cloud application and its requirements.

As we march towards the end of the MORPHEMIC project, it is not expected that new changes will be performed on CAMEL and MDS as both artefacts have been already revised in accordance to the MORPHEMIC platform features and their scheduling. However, we foresee that it is now the time to work on the exploitability of these artefacts in terms of organisations that might desire to exploit the MORPHEMIC platform. In this respect, in the last period of the project, there is a plan to work on CAMEL templates for different types of applications. Such templates could provide assistance to a great variety of use-cases by supplying to them template-based CAMEL models and sub-models which convey already sufficient information and require few additions and changes in order to accommodate the actual content and requirements of the real application at hand. For instance, the deployment architecture for workflow-based applications could be supplied and thus the owners of these applications could just modify the configuration of the application components to guarantee their proper deployment according to this tailor-made architecture. This templating work is undergoing and is expected to be reported in the last deliverable of WP1, D1.4 *Final Component Specification Collection & Enrichment Mechanisms*. Further, the CAMEL Designer will be extended in order to enable the specification of CAMEL models that comply to the new versions of CAMEL and MDS. This will be covered in deliverable D5.2 *User Interface Guidelines*.

7 References

[1] Kais Chaabouni *et al.*, "D1.1 Data, Cloud Application & Resource Modelling," MORPHEMIC Project Deliverable, Dec. 2020.

[2] A. P. Achilleos *et al.*, "The cloud application modelling and execution language," *J Cloud Comp*, vol. 8, no. 1, p. 20, Dec. 2019, doi: 10.1186/s13677-019-0138-7.

[3] Gordon Blair, Nelly Bencomo, and Robert B. France, "Models@run.time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009, doi: 10.1109/MC.2009.326.

[4] B. Rochwerger *et al.*, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 535–545, Jul. 2009, doi: 10.1147/JRD.2009.5429058.

[5] A. J. Ferrer *et al.*, "OPTIMIS: A holistic approach to cloud service provisioning," *Future Generation Comp. Syst.*, vol. 28, no. 1, pp. 66–77, 2012, doi: 10.1016/j.future.2011.05.022.

[6] X. Etchevers, T. Coupaye, F. Boyer, and N. de Palma, "Self-Configuration of Distributed Applications in the Cloud," in *2011 IEEE 4th International Conference on Cloud Computing*, Washington, DC, USA, Jul. 2011, pp. 668–675. doi: 10.1109/CLOUD.2011.65.

[7] D. K. Nguyen, F. Lelli, M. P. Papazoglou, and W.-J. van den Heuvel, "Blueprinting Approach in Support of Cloud Computing," *Future Internet*, vol. 4, no. 1, pp. 322–346, Mar. 2012, doi: 10.3390/fi4010322.

[8] Tobias Binz, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann, "TOSCA: Portable Automated Deployment and Management of Cloud Applications," in *Advanced Web Services*, Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel, Eds. Springer, New York, NY, 2014, pp. 527–549. doi: 10.1007/978-1-4614-7535-4_22.

[9] G. C. Silva, L. M. Rose, and R. Calinescu, "Cloud DSL: A language for supporting cloud portability by describing cloud entities," *CEUR Workshop Proceedings*, vol. 1242, pp. 36–45, 2014.

[10] V. Andrikopoulos, A. Reuter, S. Gómez Sáez, and F. Leymann, "A GENTL Approach for Cloud Application Topologies," in *Advanced Information Systems Engineering*, vol. 7908, C. Salinesi, M. C. Norrie, and Ó. Pastor, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 148–159. doi: 10.1007/978-3-662-44879-3_11.

[11] D. Ardagna *et al.*, "MODACLOUDS, A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds," in *ICSE MiSE: International Workshop on Modelling in Software Engineering*, 2012, pp. 50–56.

[12] A. Bergmayr, U. Breitenbücher, O. Kopp, M. Wimmer, G. Kappel, and F. Leymann, "From Architecture Modeling to Application Provisioning for the Cloud by Combining UML and TOSCA:," in *Proceedings of the 6th International Conference on Cloud Computing and Services Science*, Rome, Italy, 2016, pp. 97–108. doi: 10.5220/0005806900970108.

[13] F. Chauvel *et al.*, "Definition of the ARCADIA context model," Arcadia project deliverable D2.2, Jul. 2015.

[14] M. Hamdaqa and L. Tahvildari, "Stratus ML: A Layered Cloud Modeling Framework," in *2015 IEEE International Conference on Cloud Engineering*, Tempe, AZ, USA, Mar. 2015, pp. 96–105. doi: 10.1109/IC2E.2015.42.

[15] K. Wild, U. Breitenbucher, L. Harzenetter, F. Leymann, D. Vietz, and M. Zimmermann, "TOSCA4QC: Two Modeling Styles for TOSCA to Automate the Deployment and Orchestration of Quantum Applications," in *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, Eindhoven, Netherlands, Oct. 2020, pp. 125–134. doi: 10.1109/EDOC49727.2020.00024.

[16] C. K. Dehury, P. Jakovits, S. N. Srirama, G. Giotis, and G. Garg, "TOSCAdata: Modeling data pipeline applications in TOSCA," *Journal of Systems and Software*, vol. 186, p. 111164, Apr. 2022, doi: 10.1016/j.jss.2021.111164.

[17] K. Kritikos *et al.*, "Multi-cloud provisioning of business processes," *J Cloud Comp*, vol. 8, no. 1, p. 18, Dec. 2019, doi: 10.1186/s13677-019-0143-x.

[18] K. Kritikos and P. Skrzypek, “Are Cloud Modelling Languages Ready for Multi-Cloud?,” in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion - UCC ’19 Companion*, Auckland, New Zealand, 2019, pp. 51–58. doi: 10.1145/3368235.3368840.

[19] Geir Horn *et al.*, “D2.3 Proactive utility: Framework and approach,” MORPHEMIC Project Deliverable, Jun. 2021.

[20] Ciro Formisano, Robert Gdowski, Adeliya Latypova, Ferath Kherif, and Sebastian Geller, “D6.1 Industrial requirements analysis,” Morphemic Project Deliverable, 2020.

[21] Ferath Kherif, Robert Gdowski, Sebastian Geller, Ciro Formisano, and Adeliya Latypova, “D6.3 Use cases definition and preparation,” MORPHEMIC Project Deliverable, Dec. 2020.

[22] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, “Workflow Patterns,” *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003, doi: 10.1023/A:1022883727209.

[23] Geir Horn and Paweł Skrzypek, “MELODIC: Utility Based Cross Cloud Deployment Optimisation,” in *Proceedings of the 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Conference Location: Krakow, Poland, May 2018, pp. 360–367. doi: 10.1109/WAINA.2018.00112.

[24] Yiannis Verginadis, Ioannis Patiniotakis, Christos Chalaris, Gregoris Mentzas, Kyriakos Kritikos, and Keith Jeffery, “D2.4 Metadata schema,” Melodic Project Deliverable, Nov. 2017.