MORPHEMIC

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 –
31 December 2022

www.morphemic.cloud

Deliverable reference
D1.1

Date
31 December 2020

Responsible partner
FORTH

Editor(s)
Kyriakos Kritikos

Reviewers
Paweł Skrzypek, Sebastian Geller, Robert Gdowski

Distribution
Public

Availability
www.morphemic.cloud

Executive summary

The modelling of a polymorphic application is a pre-requisite for its suitable management. Such a modelling bares its own particularities and requirements that must be met. In particular, it needs to cover all domains and metadata related to multi-cloud, polymorphic application management. Such domains include the application's deployment topology, its own requirements as well as its measurement. While the management goals and activities include not only application monitoring but also reactive and proactive application adaptation.
Morphemic seeks to re-use modelling artifacts from the Melodic platform, developed by the Melodic project, to not create everything from scratch and save the respective effort and time. In this respect, CAMEL as the modelling language and the metadata schema (MDS) for metadata annotations have been selected to satisfy this goal.
CAMEL is already a rich cloud application modelling language. However, it lacks the necessary elements to enable the modelling of polymorphic applications, such as the coverage of all kinds of component configurations and the mapping of those configurations to corresponding requirements. To this end, the work conducted in this deliverable made it possible to extend CAMEL in order to support polymorphic application modelling by enhancing it towards particularly focused application management aspects.
MDS is a rich metadata schema covering essential aspects related to application deployment and big data management. However, it lacks conceptual elements related to additional kinds of resources like HPC and network ones as well as hardware accelerated ones. As such, the work conducted in this deliverable took care of extending MDS to suitably cover also such elements at the conceptual level.
This deliverable not only explains what were the enhancements made to CAMEL and MDS but also explicates why they have been performed while it also supplies both implementation details as well as a modelling example showcasing how CAMEL and MDS can be exploited to support the specification of a polymorphic use-case application. This proves that polymorphic application modelling is supported in Morphemic through the CAMEL and MDS frameworks and can be improved in the near future through exploiting relevant (evaluation) feedback so as to produce the final forms of both CAMEL and MDS extended versions.
This deliverable can be studied by both technical and use-case partners in Morphemic for realising the Morphemic Preprocessor as well as specifying the corresponding use-cases. It can be also advocated by an external audience that investigates the use of CAMEL and MDS for supporting polymorphic application modelling and subsequently management.

Author(s)
Kais Chaabouni, Maxime Compastié, Robert Gdowski, Kyriakos Kritikos, Andreas Tsagaropoulos, Yiannis Verginadis, Ioannis Patiniotakis

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

Table of Contents

1 Introduction

1.1 Scope

Polymorphic applications are applications that can have alternative architecture variants, derived from the fact that the application components can have different forms (e.g., function or micro-service) or configuration classes (container, VM, serverless). As such, each architecture variant maps to the selection of one form of each component from those possible. Thus, depending on the respective application requirements (e.g., to cater for a particular customer class), one architecture variant from those available can be selected and deployed in one or across multiple clouds. As such, the main research issues here are: (a) how to automatically select and deploy the right application architecture variant (e.g., choose the most cost-effective variant due to budget restrictions) as well as (b) how to adapt both the application architecture variant and configuration according to the current application requirements and context (e.g., migrate to second optimal variant as the first one always leads to Service Level Objective (SLO) violations). For the second issue, it is also quite interesting to support both the reactive and proactive application adaptation. Reactive adaptation relies on sensing problematic situations like SLO violations and reacting upon them to enable the application to still deliver a specific service level. On the other hand, proactive adaptation relies on predicting such problematic situations, where such a prediction can rely on the development and use of techniques for application quality and deployment utility prediction. The added-value of this latter adaptation form is that it copes with a problem before its actual appearance, thus attempting to either completely address it or restrain its impact to the possible minimum. The benefits of addressing these two research issues are considerable: the application deployment is always optimally satisfying the current application requirements and context, leading to the reliable delivery of service levels, the increase of the application and its provider reputation and the subsequent increase of the provider's net gain. As such, polymorphic applications tend to change the research map of multi-cloud application management [1] by supplying more opportunities for true application (deployment) optimisation through the runtime change of the application architecture (or form) apart from the modification of the kinds and quantities of the resources used.
Based on the above analysis, the Morphemic project aims at advancing the state-of-the-art by providing proper support to polymorphic application management. A major pre-requisite for supporting this management kind is polymorphic application modelling. Without the proper modelling of polymorphic applications, their optimal deployment and reconfiguration cannot be supported by any cloud (application) management platform. Such a modelling not only needs to be rich enough but also deeply cover all necessary management domains, including application deployment, requirements and monitoring.
In order to reach its aforementioned ambitious goal, Morphemic has decided to re-use solutions and tools from the Melodic project. In particular, the Morphemic project aims at producing Morphemic Preprocessor, an extension to the Melodic platform that can enable it to support both polymorphic and proactive adaptation as well as supply self-healing capabilities. For convenience reasons, the result of integrating Morphemic Preprocessor with the Melodic platform is called the Morphemic platform within the Morphemic project, which looks a more typical outcome of an equally-named project.
The re-use of Melodic solutions and tools by Morphemic also holds for the case of the application modelling feature. In particular, to not reinvent the wheel and increase the respective effort and time, Morphemic has decided to re-use the Cloud Application Modelling and Execution Language (CAMEL) [2] and the MetaData Schema (MDS) [3].
CAMEL is a rich modelling language covering all necessary domains for multi-cloud application management, such as the ones mentioned above. However, it currently lacks the necessary conceptual capabilities to support the specification of polymorphic applications as it is not able to model all possible application component forms and configurations while it is missing also network-related concepts and relationships. To this end, the work conducted in the context of this deliverable focused on extending CAMEL in a minimal but sufficient manner to deliver the missing polymorphic application modelling capabilities.
MDS is also a rich metadata schema covering concepts and their relationships across the application deployment, big data management and contextual security domains. As such, it includes concepts and properties related to where (big data) applications should be placed, how big data can be actually managed and how access to both data and resources (including application components) can be restricted. Such a schema is quite handy in the context of extending CAMEL at the model level as well as supplying suitable annotations for CAMEL elements at the same level. In this way, CAMEL is actually enhanced to describe additional features or domains (e.g., requirements for resources like Virtual Machines (VMs) and Graphical Processing Units (GPUs)) without the need to extend it at the conceptual level. However, after carefully examining the application deployment domain, MDS has been found to lack some important concepts related to missing resource kinds (e.g., High-Performance Computing (HPC), hardware-accelerated ones), platform kinds (e.g., serverless) and network elements. Further, some small gaps (e.g., missing trusted execution

environment concepts) were also identified in terms of the other two domains (i.e., big data management and contextual security). As such, in the context of supporting polymorphic application modelling and annotation, MDS has been also extended to cover all missing conceptual elements.

All the aforementioned modelling extensions have been already implemented and are explicated in this deliverable. This research and development work pertains mainly to work-package WP1 of the Morphemic project and tasks T1.1 & T1.2 related to polymorphic application modelling with CAMEL and metadata modelling, respectively. The succession of D1.1 is D1.3, titled as "Final Data, Cloud Application and Resource Modelling", where improvements on both CAMEL and MDS extensions will be incorporated based on the feedback given by use-case and technical project partners.

This modelling-related work impacts all research and development tasks in the project based on the already explained rationale that the modelling is a pre-requisite for successful application management. In essence, any kind of modelling feature, if decided to be supported, has to be reflected in the corresponding platform, i.e., the Morphemic one. It should be stressed here that it is not expected that all modelling features will be realised in the Morphemic platform due to resource restrictions pertaining to this project. In essence, the modelling features will be prioritised based on the respective platform requirements that they map to (see Deliverable D6.1 [4]). As such, modelling features with high priority will be surely realised in the Morphemic platform while features with lower priority might or might not be realised. However, in any case and as section 1.2 indicates, all modelling features are relevant for adoption by an external audience, which could be related to, e.g., providers of other cloud management platforms that desire to adopt CAMEL in order to support polymorphic application modelling and management.

Based on the aforementioned impact of the modelling extensions on the project's research and development tasks, the affected work-packages in the project are WP2-WP5. We should further highlight that the extension of CAMEL and MDS is tightly related to task T5.1, User Interfaces, where a new CAMEL graphical editor will be designed, named as CAMEL Designer. Finally, in the context of use-case definition and preparation, i.e., deliverable D6.3 "Use Cases Definition and Preparation", the aforementioned work is relevant in terms of the actual modelling of the use-cases of the project.

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 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 (where the 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 conducts a state-of-the-art analysis related to cloud application, cloud resource and big data modelling, covering relevant languages, ontologies and metadata schemata.
- Chapter 3 explicates the new version of the enhanced CAMEL language (3.0) along with the changes that have been performed on its previous version (2.0). It also supplies respective implementation details.
- Chapter 4 explicates all MDS extensions while it details the way they have been realised.
- Chapter 5 utilises a use-case from the Morphemic project in order to highlight how CAMEL and MDS jointly enable the complete modelling of a polymorphic application.
- Chapter 6 concludes this document and supplies particular directions for future work related to polymorphic application modelling.

2 State-of-the-Art Analysis

The goal of this section is twofold:

- On one hand, in section 2.1, it attempts to assess whether the current cloud application modelling languages support both multi-cloud and polymorphic application modelling. Its main goal is to highlight that only

CAMEL covers well the modelling of multi-cloud applications and that it already includes some conceptual elements that are necessary for polymorphic application modelling. As such, the Morphemic consortium has taken the right decision to adopt this language and extend it so as to provide support to the latter modelling kind.
- On the other hand, section 2.2 reviews the state-of-the-art in cloud application and services description. Its main goal is to unveil all those modelling approaches that have been consulted in order to extend accordingly MDS so as to provide a better and more complete coverage of resource (including network) and platform kinds. The main outcome is that due to this consulting and the proper MDS extension, this extension seems to advance the state-of-the-art in these modelling domains by providing a deeper and more extensive taxonomy of relevant concepts and properties. However, a more thorough evaluation for proving such an advancement is considered out of scope of this deliverable but could be part of a future publication, within the context of this project, focusing on analysing the MDS extension.

2.1 Cloud Application Modelling Languages

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 in [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 modelling domains are covered and integrated, which kind of cloud services are supported and whether the models@runtime paradigm is adopted:

- *domain coverage*: which domains from those relevant to the application lifecycle are covered by a language
- *integration level*: what is the level of integration [2] between the different domains/sub-languages covered/utilised by a language
- *delivery model support*: which kinds of cloud services are supported by a language
- *models@runtime support*: for which domains is the models@runtime paradigm [5] adopted by a language

In our view and based on the requirements given in section 3.1.1, a cloud language can support polymorphic application modelling when it is able to satisfy the following additional criteria:

- *application architecture variability*: the 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*: the 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 standalone and in another form should be split into other components, thus being complex 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 12 cloud application / service modelling languages, including CAMEL. These languages are those assessed also in [2] and represent the state-of-the-art in terms of cloud application modelling languages which are not provider-specific. As provider-independence is a crucial characteristic in order to support cross- and multi-cloud deployments. The evaluation results are depicted in Table 1.
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 3 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 not properly model complex components, it has a "Medium" evaluation value. Otherwise, if the language also properly and complete 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**	**App. Arch. Variability**	**Comp. Conf. Variability**	**Comp. Complexity**
Reservoir OVF Extension [6]	Low	N/A	IaaS	N/A	Low	Low	Low
Optimis OVF Extension [7]	Medium	N/A	IaaS	N/A	Low	Low	Low
Vamp [8]	Low	N/A	IaaS	N/A	Low	Low	Low
4CaaSt Blueprint Template [9]	Low	N/A	IaaS, PaaS	N/A	Low	Low	Low
TOSCA [10]	Medium	Medium	IaaS, PaaS	Deployment*	Low	Medium	Low
Provider DSL [11]	Low	Medium	IaaS	N/A	Low	Medium	Low
GENTL [12]	Low	N/A	IaaS	N/A	Low	Low	Low
ModaCloudML [13]	Medium	Low	IaaS, PaaS	Deployment	Low	Medium	Medium
CAML [14]	Medium	Medium	IaaS	N/A	Low	Low	Low
Arcadia Context Model [15]	High	Medium	IaaS	Deployment	Low	Medium	Low
StratusML [16]	Medium	High	IaaS	Deployment	Low	Low	Low
HCL/Terraform [1]	Low	N/A	IaaS, PaaS	N/A	Low	Medium	Low
CAMEL 2.0	High	High	IaaS, PaaS, SaaS**	Deployment, Metric, Data	Medium***	Medium	Medium
CAMEL 3.0	High	High	IaaS, PaaS, SaaS**	Deployment, Metric, Data	Medium	High	High

*: TOSCA [10] 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.
**: 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 the above table, 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 how it enables the full specification of polymorphic applications in terms of all relevant application lifecycle management aspects.

[1] https://www.terraform.io/docs/configuration/syntax.html

2.2 Ontologies and Metadata for Cloud Application & Services Description

As the MDS already covers well the big data domain as well as supplies adequate support for common resource types and services, the focus of the analysis is mainly on the network and hardware-accelerator domains mapping to the two most significant extensions of MDS. For the interested reader, a review of the state-of-the-art in big data and application placement domains can be found in [18].

2.2.1 Network Ontologies, Meta-Models and Languages

The authors in [19] suggest a meta-model for cyber-physical systems that mainly focuses in their communication. The meta-model was produced by first considering and analysing concepts from computing in civil engineering and then determining those to be included by covering both communication-related properties and system components relevant to communication. It was utilised for validation purposes in the modelling of a prototype cyber-physical system related to Building Information Modelling (BIM) and physically implemented in the laboratory.

The technical report in [20] proposes an ontology for computer network categorization aiming to assist in the development of knowledge-base systems for network management. Such systems take the form of a logical reasoner able to provide automation support for management tasks typically realised by human experts. The ontology strictly covers the network domain and especially includes concepts to represent network entities and protocols while also catering for capturing relevant relationships between them as well as their functioning mechanisms.

In [21] the authors study the way to intelligently and efficiently refine and manage a vast amount of network monitoring data sources and come up with a specific solution relying on the use of AI reasoning along with an intuitive user interface. This solution attempts to minimise the user interaction and required knowledge during the network monitoring search process by refining the displayed information based on the user choices. It relies on the use of an ontology for realising a knowledge base of multiple different information aspects like the Internal Management structure, the data sources physical location and network switches.

The network description language NDML+ is presented in [22], which is an extension of NDML (Network Design Markup Language) to cover additional aspects like Traffic mapping, Costs and Geometry analysis as well as to make its design more uniform (in terms of underpinning modelling technologies). This new language has been developed in the context of the CANDY project to support the computer-aided design of networks through the framework produced by that project.

The ToCo (TOuCan Ontology) ontology [23] has been produced in the context of the EPSRC TOUCAN project (Grant No. EP/L020009/1). Such an ontology is able to specify the physical infrastructure as well as the quality of channel, services and users in heterogeneous telecommunication networks spanning multiple domains. At the top-level, the ontology relies on the Device-Interface-Link pattern which is then specialised in more specific concepts and relationships resulting in a very rich telecommunication networks ontology.

The CNMO (Communications Network Modelling Ontology) ontology [24] is able to specify network models that cover multiple aspects related to network development and operations. This ontology covers concepts and aspects like nodes, links, traffic sources and protocols. It has been developed using terminologies and concepts from various network modelling, simulation and topology generation tools.

In [25] the OpenMobileNetwork Resource Description Framework (RDF) dataset for mobile networks and devices is presented, able to describe mobile networks, their structure and topology. This dataset has been constructed from RDF databases like OpenCellID and OpenBMap and has been enriched by exploiting live context semantic data extracted from smartphones or WiFi (Wireless Fidelity) access points. Such a dataset can then be utilized along with interlinked information present in the LOD Cloud in order to realise applications that depend on mobile network and positioning data like semantic location-based services.

The thesis by J. J. van der Ham [26] covers the specification of a network description language that specialises in the description of network topologies for hybrid networks (above the physical layer). This language enables the generation and exchange of network maps so as to allow the automatic correlation of information across domains. In addition, it enables end-users to specify light-path reservation requests while facilitates services providers in terms of validating the feasibility of these requests. Finally, the author claims that their semantic model attains a better trade-off between expressivity and usability with respect to other related languages while it inherits all the benefits from the introduction of semantics.

The paper in [27] suggests a practical domain ontology modelling approach for telecommunication services that enhances the reusability of relevant conceptual domains and greatly reduces the main technical obstacles in domain ontology modelling. Based on this approach, a rich telecommunication ontology has been developed which enables to precisely specify telecommunication services, to easily discover them as well as address the semantic interoperability problem. The ontology modelling covers multiple layers which cover common, domain and application-specific ontology modules. The latter include the Terminal Capability Ontology, the Network Ontology, the Service Role Ontology, the Charging Ontology, the Service Quality Ontology and the Service Category Ontology.

The Web Ontology Language (OWL)-based ontology in [28] covers the network domain, which is approached from various perspectives, including topology, protocols, security, hardware and performance. The ontology seems to be quite deep while it introduces various relations between the modelled concepts. However, the features/attributes of these concepts are scarcely considered.

By examining all the above work, one can understand that there is great variation in terms of network sub-domains coverage, richness, deepness and cloud-relatedness. Further, while not being the goal to achieve and prove in this deliverable, the network extension of MDS could be considered as superior to that work by scoring higher in all these dimensions. This could be regarded as the main outcome of selecting and integrating the best and most relevant parts from each work in the state-of-the-art and structuring them accordingly in the right place within the MDS. Further, it should be stressed that in contrast to the aforementioned and analysed work, MDS does cover multiple cloud-related network elements, making it up-to-the-target with respect to its main application goal, which relates to facilitating the semantic annotation of multi-cloud application models specified in CAMEL.

2.2.2 Accelerated Resource & HPC Ontologies, Meta-Models and Languages

The work in [29] extends the mOSAIC ontology, pillar of the IEEE 2302 — Standard for Intercloud Interoperability and Federation, towards the creation of an ontology named as CloudLightning (CL-Ontology). This ontology facilitates the incorporation of heterogeneous resources and HPC environments in the Cloud. In particular, it provides support for resource management by modelling specific hardware accelerators as well as different resource abstraction methods like virtual machines and containers, by matching service requests to specific heterogeneous infrastructures, and by enabling intelligent resource discovery.

The HPCRO ontology is proposed in [30] covering concepts related to both hardware and software resources as well as relevant properties and relations between them. It also suggests the Wordnet-based Quick Resource Index List (WQRIL) method that supports the efficient, semantics-based, fuzzy discovery of HPC resources.

The work in [31] proposes a modular and extensible XML-based platform language called XPDL, which specializes on the description of heterogeneous multicore systems and clusters. The specifications conforming to XPDL mainly supply information about the hardware and installed systems software in a platform which is relevant for the optimisation of application programs and systems settings targeting improved performance and energy efficiency. They can also become distributed by relying on the use of hyperlinks to reference one part from the other. Apart from the language itself, a toolchain is suggested, able to edit XPDL specifications as well as to produce driver code for microbenchmaking to explore empirical performance and energy models at deployment time.

MARTE (OMG adopted specification) [32] is used for the description of hardware architectures. It includes various meta-models but the most relevant ones in the context of this analysis are the General Resource Model (GRM) and the Hardware Resource Model (HRM) meta-models. The GRM meta-model covers the description of various resource types like storage, communication, timing, synch, concurrency, computing and device. On the other hand, the HRM meta-model spans two views: the logical that classifies each hardware resource according to its functional properties while covering various concrete resources of multiple resource types (e.g., computing or communication) and the physical view focusing on the physical properties of resources.

The work in [33] suggests a novel SoC co-design methodology based on MDE and MARTE which enables to raise the abstraction level and model fine-grain reconfigurable architectures, such as FPGAs. To this end, to achieve the latter goal, the authors have extended MARTE to cover some specific features of FPGAs. In particular, the HwProcessor stereotype has been enhanced in order to cover the implementation technology of an FPGA, which could be hardcore or softcore IP. Further, the HwComponent concept has been extended to include the areatype attribute. This enables to indicate, especially for Partial Dynamic Reconfiguration components, whether their respective regions are static or dynamically reconfigurable.

The above work also varies in terms of the criteria introduced in section 2.2.1 (i.e., sub-domains coverage, richness, deepness and cloud-relatedness) as well as the *standards support* one, which we consider as important in the context of the current domain, i.e., the cloud one. The main outcome of inspecting this work is that we have produced an MDS extension by selecting and integrating the best and most relevant parts of that work and properly structuring them in the right place within the MDS. This outcome relates to an extension of the MARTE standard with additional concepts and properties that enable to cover well the HPC, FPGA and GPU sub-domains (of the resource domain). It is also possible that this outcome, when compared to the related work, could be considered as superior. However, it is not the goal of this deliverable as well as the goal of this project to produce a very rich MDS model that advances the state-of-the-art. The goal is rather to extend MDS in order to properly support the annotation of polymorphic multi-cloud application models in CAMEL.

3 CAMEL Extensions

3.1 Conceptual Analysis

The conceptual analysis of CAMEL is clearly separated into three main sections. Section 3.1.1 explicates what were the main requirements that drove the extension of CAMEL. Section 3.1.2 explains the CAMEL enhancement process as well as the way CAMEL will be exploited by the Morphemic platform. Finally, Section 3.1.3 analyses the new version of CAMEL that has been produced by focusing mainly on those domains that are touched by the devops/modeller.

3.1.1 CAMEL Extension Requirements

Apart from the obvious need to support polymorphic application modelling, additional CAMEL extension requirements came into play, which could be either related to:

- the improvement of this language, based on feedback collected from the Melodic and PaaSage projects [34], [35]
- requirements coming from other Morphemic features and especially the proactive adaptation one[2].
- requirements coming from Morphemic use-cases [4]

As such, all CAMEL extension requirements can be summarised as follows:

- Polymorphic-modelling related
    - *PM1*: New configurations for components have to be covered, especially those related to hardware-acceleration-use scenarios (e.g., with respect to FPGA-based resources).
    - *PM2*: Hosting relationships do not have to be expressed by the user but need to be inferred by the (multi-cloud) application management platform. This is essentially necessary in the face of multiple configurations per component where the selection of one can influence the hosting topology of the application. For instance, a script-based configuration leads to a VM-based hosting, so the respective application component should be directly hosted by a VM. On the other hand, a container configuration leads to a container hosting the application component, where the container itself is hosted by a VM.
    - *PM3*: Each component configuration may come with its own requirement set as different configurations might have different requirements, e.g., with respect to resources or the environment on top of them.
    - *PM4*: Components, depending on their form, can be either standalone or complex ones, comprising multiple, other components. For example, a component managing users could be either offered in a micro-service, standalone form or be formulated as a composite component that comprises a set of serverless sub-components/functions.
- Improvement-related:
    - *IR1*: Components should be re-used in the context of different applications that can be modelled via CAMEL (related to feedback from the Melodic project). This enables to: (a) build new applications from existing components, thus reducing both the modelling and development time for these applications; (b) extend existing applications to make them more complex but also increase their added-value. Such an extension would, however, require to view applications as coarse-grained components, i.e., as complex components that include other components inside them. The re-use of such components would then lead to the production of more complex applications comprising components with different granularity.
    - *IR2*: Technical communication semantics should be introduced (related to feedback from the PaaSage project) [35] to govern how the communication coupling of application components can happen at the instance level. In particular, if two application components A and B need to communicate, then how the communication binding of their instances should take place? Should we consider that every instance of A should communicate with every instance of B? Or that there is an

[2] https://confluence.7bulls.eu/display/MOR/Feature%3A+Proactive+adaptation

one-to-one mapping between instances of A and B in terms of communication? All these questions indicate the need to introduce the right, extra elements in CAMEL in order to enable determining the relevant communication semantics.

- Feature-related:
    - *FR1*: Supporting the modelling of prediction-related metrics. In order to predict the quality of an application or its components as well as the overall expected utility of an application's configuration, there is a need for supporting the modelling of prediction-related metrics. Such metrics actually relate to the application of a certain prediction technique/method over the measurements of a normal (single/raw or composite) metric based on a specific probability and time horizon. Based on this modelling, the platform can then be able to realise such metrics and thus support the production of the needed predictions.
- Use-case related:
    - UR1: Communication requirements (based on IS Wireless use-case requirements identified as UC-C-UF.1 and UC-1-UF.2 in D6.1 [4]) should be expressed in order to extend the portfolio of scenarios that can be covered by CAMEL so as to include network-/communication-aware ones. Such communication requirements should include constraints over the quality of the communication (e.g., latency) between two application components. As such, this kind of constraints can restrain the deployment of both components but still maintaining the flexibility of cloud service selection. In other words, such constraints do not restrict the deployment of one of the two components in terms of a specific location but actually restrain where the second component should be placed once the location of the first one has been selected.

Please note that the first category of requirements, i.e., the polymorphic-modelling-related, include requirements that have been also identified in D6.1 [4]. This indicates the alignment and common vision of the consortium in terms of what is polymorphic application modelling and how it can be supported. Further, many other requirements coming from that deliverable were already supported in CAMEL 2.0 so this is the main reason why they have not been identified as new. In the following, we supply a table that highlights which relevant requirements from D6.1 have been already met by CAMEL 2.0 and which requirements are now met by CAMEL 3.0. The table also provides two additional columns which unveil: (a) whether a specific requirement is met/supported through a specific CAMEL version alone or also the MDS; (b) which requirements are clearly modelling-related and which are not such that the goal of CAMEL should have been or must be (depending on its version) to provide support for their achievement through the supply of respective modelling constructs.

Requirement ID	**Requirement Name**	**CAMEL Version**	**MDS**	**Comment**
MOR-SE.1, 2, 3 and 6	Polymorphic Environments: Cloud, Hybrid Clouds, Multi-Cloud, Bare metal, HPC,	2.0	Original version	CAMEL via MDS annotations enables to pose constraints on features related to the resources of these environments enabling their selection during deployment reasoning
MOR-SE.4, 5, 7, 8 and 9	Polymorhipc Environments: Fog, Edge, Hardware Accelerators, FPGA	2.0	New version	Explanation same as previous requirement. In addition, with respect to MOR-SE.9, it should be noted that a new environment can be supported via a further extension of MDS. There is no need to change CAMEL for that.
MOR-SA.1 to 4	Polymorphic application forms: VM, containers, serverless and more forms	2.0	Both versions	CAMEL 2.0 already supports more than three configuration kinds from those needed. The additional ones are cluster and PaaS. Please note that CAMEL 3.0 identifies container configuration as a distinct configuration kind while in CAMEL 2.0 this was modelled via a script configuration. The mentioning of MDS relates to the fact that apart from the configurations, there is a need for imposing

				constraints on respective resources (e.g., VM) or platforms (e.g., serverless).
MOR-CON.1	Preconfigure multiple deployment configurations	3.0	-	CAMEL 2.0 was able to map a component to multiple configurations. However, only one was taken into consideration in the original Melodic platform. Further, CAMEL 3.0 enables to map each configuration to a different requirement set. Thus, CAMEL 3.0 caters for the full modelling of this feature.
MOR-SH.1 to 2	Real time infrastructure and application monitoring	2.0	-	CAMEL 2.0 enables the complete modelling of all those infrastructure and application metrics related to such a monitoring. Which is a pre-requisite for the achievement of these two requirements.
MOR-AD.1 to 3	Proactive Adaptation, Prediction capabilities on applications, prediction capabilities on infrastructures	3.0	-	CAMEL 3.0 introduces prediction-based composite metrics which enable to conduct the prediction of application and infrastructure metrics. Through such metrics a suitable utilify function can be also modelled. In this respect, CAMEL 3.0 provides proper support to all three requirements.
UC-C-SE.2	Multi-site deployment	2.0	Original version	CAMEL 2.0 introduces the right flebility in terms of different requirement kinds in order to support the deployment of applications in multiple sites.
UC-C-UF.1	Targeted deployment: network capability	3.0	New version	Network/Communication requirements can be specified in CAMEL 3.0. These include constraints on network/communication features and properties annotated through the new version of MDS.
UC-C-UF.2 to 5	Targeted deployment: price, packaging, geographical awareness, computing power	2.0	Original version	CAMEL 2.0 enables to pose constraints/requirements on all those elements through the assistance of MDS.
UC-C-SEC.2	Support for secure communications	3.0	New version	It is possible through CAMEL 3.0 to pose a communication requirement that indicates via a respective MDS annotation that there is a need to employ a secure private network.
UC-C-SEC.3	Support for security-related applications	3.0	New version	Explanation similar to previous one. MDS now has the capability to specify specialised security components like firewalls and IDSs which can be introduced in the application architecture.
UC-1-SE.2	Platform awareness - CPU pinning	2.0	New version	There is a need for specifying a resource constraint that highlights through MDS the need for a CPU pinning capability. CPU pinning has been introduced in new version of MDS while CAMEL 2.0 is already able to model resource constraints/requirements.
UC-1-UF.1, UC-3-UF.2	Support for low latency in terms of deployment time, Targeted deployment: deployment time	2.0	Original version	The utility-based approach followed in Melodic introduced the ability to include in the utility function a factor related to deployment cost which was assessed based on deployment time. As such, the devops, through CAMEL 2.0 and MDS original

				version, can include with the most suitable weight this factor in order to guarantee the low deployment latency for the respective application. Obviously, the user can also specify SLOs that can restrain the deployment time that an application can have.
UC-1-UF.2	Targeted deployment: latency between the deployed components	3.0	New version	CAMEL 3.0 enables the specification of communication requirements between application components which can include constraints on latency through the use of the new MDS version.
UC-C-SE.3	Support for GPU	2.0	Original version	CAMEL enables through the use of MDS annotations the specification of constraints on GPU features in resource requirements. This enables to select matching resources with GPUs that exhibit the required capabilities.
UC-2-SE.3	Support for Master orchestrator components on public environment	2.0	-	In Melodic, the approach taken was that one Master orchestrator component was indirectly deployed in case of the use of at least one slave/task-based component in an application. Morphemic aims to follow a direct approach to handle this capability. This means that the orchestrator component should be directly specified in CAMEL along with its configuration and requirements. This will be also facilitated by a template CAMEL model which will include a basic but standardised architecture of a big data application that can then be customised based on the requirements of the current application at hand.
UC-3-UF.1	Targeted deployment: memory	2.0	Original version	CAMEL through the use of MDS annotation can specify memory constraints in resource requirements. Obviously, memory-related metric and properties can be also included in the specification of a utility function.
UC-3-SH1	Track worker velocity	2.0	-	This seems to be a composite application metric that can be modelled in CAMEL and then monitored by the platform.

From the above table, it can be seen that 35 requirements are met or supported by CAMEL. This highlights the proper design of this language to be able to cover both multi-cloud and polymorphic features. Further, it can be seen that CAMEL 2.0 has been already a well-designed CAMEL version as it is able to cover 27 from these requirements, i.e., more than 3 quarters of them. This indicates that CAMEL 3.0 satisfies in addition 8 from the remaining requirements. This does not mean that there was not a lot of development effort with respect to producing this CAMEL version as each requirement might lead to a different amount of work to be realised in CAMEL. However, this also evidences that CAMEL 3.0 is able to meet or support all these requirements.
We should also mention that many of the requirements require the use of MDS. Indeed, by carefully observing the above table, it seems that this is the case for 27 out of the 35 requirements. This outlines well the need for the introduction and integration of this metadata schema with CAMEL. From these 27 requirements, 11 are based on the new version of MDS and 16 on the old one. This highlights that the decision to extend MDS has been correct and that such an extension contributes to the satisfaction or proper support to many of the requirements identified in D6.1 [4]. They way such an extension has been performed to meet or support these requirements is explained in section 4.1.

3.1.2 CAMEL Enhancement Process & Exploitation Flow

CAMEL 3.0 is a heavily extended version of CAMEL, which provides full support for polymorphic application modelling while it covers all other requirements related to CAMEL improvement feedback and Morphemic features. CAMEL 3.0 is rather a draft version. This means that, while developed in the project’s first year, will be validated and

continuously evolved until the end of the second year, when it will take its final form. The validation will involve a series of workshops and telcos with both technical and use-case partners of the project. The discussion with the technical partners will facilitate the determination of the impact of CAMEL changes on the Morphemic preprocessor and will enable also the changes prioritisation in terms of their realisation in that module/component. The discussion with the use-case partners will enable the full comprehension of this CAMEL version as well as its ability to fully model the respective use-cases. In both cases, valuable feedback will be also obtained from both kinds of partners in terms of CAMEL changes suitability and could enable to further evolve CAMEL 3.0 towards its final form. Such final form will be then the major CAMEL version that will drive the future versions of the Morphemic Preprocessor. This final CAMEL form will be also analysed in the context of the D1.3 deliverable due in M24.
The following table enables to compare CAMEL 3.0 with respect to the changes that it has performed on CAMEL 2.0, i.e., the current CAMEL version, according to the main (management) domains covered by CAMEL. As it can be seen from this table, CAMEL 3.0 enhances 8 domains and introduces a new one where the extensions in 3 out of the 8 enhanced domains are substantial. All these changes will be well-detailed in the following section.

Table 2 - High-Level view of CAMEL 3.0 changes on CAMEL 2.0

Domain	**CAMEL 3.0**
Core	Introduction of a component containment reference covering the whole application. Removal of application reference.
Application	New domain covered by this version. Resembles a deployment type model without hosting relationships
Deployment	Introduction of nodes, groupings of communications per component pair, technical communication semantics, mapping of configurations to requirements, mapping of components to application models, introduction of container and image configurations
Requirement	Introduction of *LinkRequirement* (equivalent to *CommunicationRequirement* in v2.5)
Metric	Same as in v2.5 plus introduction of *PredictedMetric(Instance)* concepts and deletion of *currentConfiguration* attribute from *MetricVariable* class
Data	*DataSource* now refers to *Component*, *DataSourceInstance* now refers to *NodeInstance*
Execution	Modification of *CommunicationMeasurement*, introduction of *LinkMeasurement*, *NodeMeasurement* plus deletion of other measurement kinds)
Metadata	Introduction of *implemented* attribute in *MmsObject*
Security	Introduction of predicted security metrics
Other domains	No change

The following figure explains the way CAMEL will be exploited by the Morphemic platform. The CAMEL Designer is the main CAMEL model producer where CAMEL models can then be uploaded in the Morphemic platform through the UI for deployment. Such CAMEL models will touch mainly four domains: (a) the application domain to describe the application architecture and its variants; (b) the requirement domain to explicate the main requirements of the application and its components; (c) the metric and constraint domains to cover the specification of relevant metrics and attributes as well as constraint for supporting application deployment reasoning and monitoring. These models are named as *architecture models*.
An *architecture model* is then fed to the Profiler in order to construct and maintain a profile of the respective application which is critical for application deployment reasoning support. Based on the architecture model and the non-functional parts of the application profile, the Architecture Optimiser will then choose the best architecture variant of the application. This will result in the production of an application deployment type sub-model that will be incorporated in the respective CAMEL model. The resulting CAMEL model is called a *provider-independent model*.
From that moment and on, the CAMEL model has the right content to be processed by the Melodic core. In particular, the CP Generator will produce a CP model that will be used as input to the Reasoner (a composite component comprising the Meta-Solver, Solver and Utility Generator) for deployment reasoning process. The Reasoner will then produce a specific deployment solution for the application at hand. Such a solution will result in enhancing the application deployment type model as well as generating a respective deployment instance model. The overall CAMEL model extended in this way is called a *provider-specific model*.
Finally, the provider-specific model is taken as input by the Adapter in order to orchestrate the application's deployment, including its monitoring infrastructure, through a derived deployment plan based on the cooperation with the Executionware (Proactive Scheduler). The execution-related feedback is then continuously injected into the CAMEL model in order to produce an execution sub-model, i.e., a model that covers the current execution state of the application. The enhanced CAMEL model is called an *execution model* and drives the subsequent adaptive

provisioning of the respective application to close the global optimisation loop. Please note here that the monitoring infrastructure deployed has the capability to produce measurements as well as predictions in order to drive both the reactive and proactive adaptation of the application, respectively.

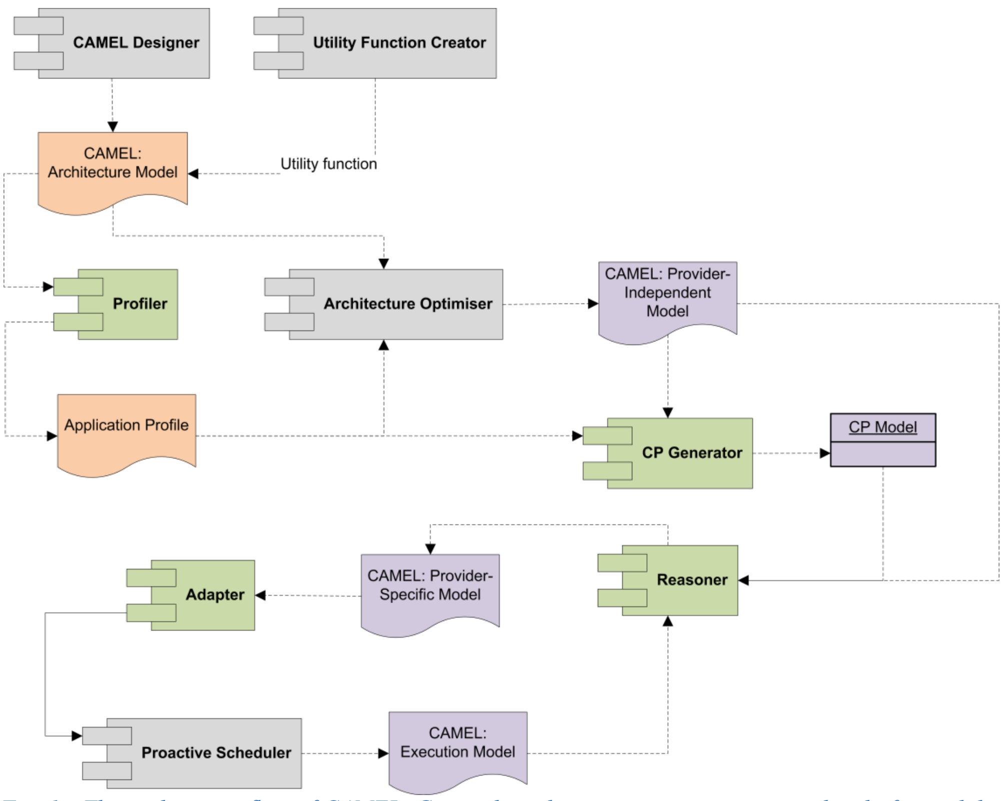


Fig. 1 - The exploitation flow of CAMEL. Grey colour denotes new components in the platform while green existing ones, orange denotes new models in the platform while purple existing ones

3.1.3 CAMEL Version 3.0

CAMEL 3.0 is a new version of CAMEL which completely addresses all requirements specified in section 3.1.1. The current content of this version is analysed in this section while any kind of adjustment or modification conducted on it based on the obtained partner feedback will be supplied in deliverable D1.3 “Final Data, Cloud Application and Resource Modelling”. In the following, the analysis of CAMEL 3.0 is incarnated as follows:

- first, we explicate the main structure of a CAMEL model
- then, we explain in a step-wise manner each part (i.e., meta-model/package) of this structure. The explanation includes also the changes made to CAMEL 2.0 to reach the desired part content. The focus of the analysis is mainly on those domains that have been updated in this CAMEL version and have been already implemented in the Melodic platform (with an exception of a newly introduced domain), which happen to include all those domains that are expected to be covered in a devops-supplied CAMEL model. More details about the untouched domains can be found in [31] and CAMEL 2.0 documentation (https://confluence.7bulls.eu/display/MEL/11+Modelling).

3.1.3.1 CAMEL Model Structure

Any CAMEL model contains a set of sub-models which map to respective CAMEL meta-models or packages focusing on a particular application management domain. Initially, only specific kinds of models are produced through any kind of editor that conforms to the CAMEL syntax, mapping to the kind of information that has to be supplied by a devops in the context of a specific application. Then, while the multi-cloud application management platform is on operation and manages this application, new models are produced by this platform that are included in the current structure of a CAMEL model. These models are exclusively managed by the platform (for application management purposes) and should not be touched by a devops user. The following table showcases all these models by providing a short description of their coverage as well as the responsible party to produce and maintain them along with the point in time where this production takes place. It also indicates whether a domain is obligatory to be covered in a CAMEL model supplied by a devops/modeller. Finally, it highlights whether a specific domain is implemented or not in the current version of the Melodic platform.

Table 3 - Details about the domains/aspects covered by CAMEL

Name	**Coverage**	**Modeller/ Editor**	**Design/ Runtime**	**Obligatory for Devops**	**Implement ed**
Core	Top model, container of other models, includes containment reference to application as a composite component	Devops / System	Both	Yes	Yes
Application	Covers the architecture of an application, i.e., the components, the different forms/configurations of these components as well as the components relationships	Devops	Design	Yes	To be implemente d
Deployment	The deployment topology of the application at the type and instance level, covering the selected configuration of application components as well as their hosting and communication relationships.	System	Both	No	Yes
Requirement	Any kind of requirement (e.g., resource, provider, QoS/SLO, optimisation) that can be associated with one or more configurations of application components or the whole application	Devops	Design	Yes	Yes
Constraint	Any kind of constraint (single or composite as logical combination of other constraints) that can be posed on either/both variables and metrics	Devops	Design	Conditional ly	Yes
Metric	All needed information to support application measurement, including the specification of (metric) variables, metrics and their contexts	Devops / System	Both	Conditional ly	Yes
Scalability	Coverage of scalability rules and their constituting parts, i.e., events and scaling actions. This aspect/domain is deprecated for the Melodic platform	Devops / System	Both	No	No
Location	Coverage of both physical and cloud-specific locations	Devops	Design	No	Yes
Unit	Coverage of different kinds of units of measurement	Devops	Design	No	Yes
Types	Coverage of value types and single values	Devops	Design	No	Yes
Security	Coverage of security controls, domains, attributes and metrics	Devops / System	Both	No	No
Organisation	Coverage of organisations, users, roles, cloud credentials and access control rules/policies	Admin	Design	No	No
Execution	All details concerning the application execution like measurements, violations of SLOs and adaptation actions performed	System	Runtime	No	Yes
Data	Data and data sources at both the type and instance level	Devops / System	Both	Conditional ly	Yes
Metadata	Conceptual schema covering concepts, relationships and properties that can be used for annotation and CAMEL model enhancement purposes	Admin	Design	No	Yes

In the above table, an underlined aspect/domain/model kind denotes a respective update of this domain or the introduction of a new one. In conformance to what was stated in section 3.1.2 and Table 2, it can be easily seen that 1 new domain has been introduced (application) and 8 others have been modified. In Table 3, we can also observe that there are 10 domains that can be touched by a devops, 5 by the system and 2 by the administrator (admin for short) (with the exception of the core domain that is not accounted for as it is always present in a CAMEL model at the top level).

The 10 domains covered by the devops concern mainly the design phase in the application lifecycle. However, please note that not all domains are necessary to be included within a CAMEL model. Three out of all these domains are not currently supported by the Melodic platform, i.e., the scalability, organisation and security ones, respectively. Three domains play an auxiliary role and might not need to be touched or already exists a model for them that can be just re-used in the context of another covered domain. In particular, the unit and type domains are usually used for enhancing the description of metrics (i.e., metric domain), where the first one already maps to a model of existing, well-known units. Such an enhancement is optional as it is not currently exploited by the platform. The third domain is location where again an existing (physical) location model exists that can be exploited for expressing location requirements.

Another class of domains includes those that might be conditionally touched depending on the nature of the application and its requirements. These domains are the data, metric and constraint ones. The first should be touched only if big data are manipulated by an application. The other two are necessary only when SLO and optimisation requirements are to be supplied by the devops. This leaves us with only two obligatory domains which are the application and requirement ones. The first describes the architecture of the application and the second its requirements (which might not include SLO & optimisation requirements such that the application deployment is always optimised based on cost). Thus, to conclude, a devops might need to include in a CAMEL model always 2 sub-models and conditionally other 3, making up 5 necessary sub-models in total, which is half of the original number that can be observed in the above table. This is demonstrated in the fifth column of the above table. It is also witnessed in the following figure that shows a CAMEL model containing 5 sub-models. Please note one important detail: the application model is part of a component which represents the whole application. That is the main reason that it does not hang directly from the Camel model element (i.e., the top-level container).

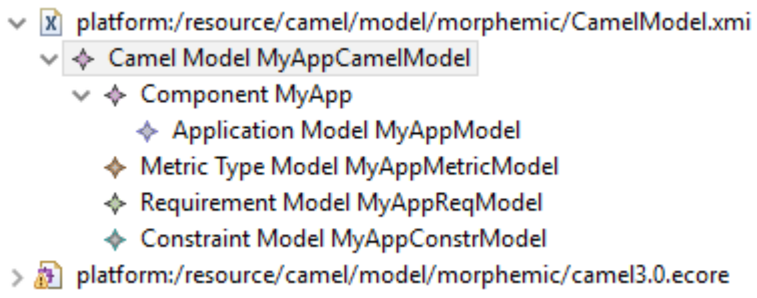

```
platform:/resource/camel/model/morphemic/CamelModel.xmi
  Camel Model MyAppCamelModel
    Component MyApp
      Application Model MyAppModel
    Metric Type Model MyAppMetricModel
    Requirement Model MyAppReqModel
    Constraint Model MyAppConstrModel
platform:/resource/camel/model/morphemic/camel3.0.ecore
```

Fig. 2 - Simple Camel Model of an application that comprises the 5 most important and necessary CAMEL sub-models

The multi-cloud application management platform needs to deal with and maintain 4 domains (the security domain is not counted as not implemented in the platform). The maintenance for most of these domains occurs at runtime, which is logical if we consider that a platform deals mainly with the runtime execution and adaptation of an application. The sole exception is the deployment domain where the deployment model of an application at the type level is produced before the runtime phase, when deployment reasoning is conducted by the platform.

For three out of the four domains previously mentioned, the instance level is mostly touched. This includes the metric and data domains, where the platform, based on the models@runtime approach, needs to monitor and maintain the runtime state of both the monitoring infrastructure and the data instances produced and manipulated by the application. This also includes the deployment domain where the models@runtime approach obviously covers the current deployment topology of the application at runtime. The final domain covered by the platform is the execution one, which is maintained for analysis and optimisation purposes (e.g., avoid bad deployments).

The admin is the role that has just one domain to touch in a CAMEL model. This includes the metadata domain covering the metadata schema of the platform. This domain can be touched even before an application is modelled via CAMEL. The model mapping to that domain is actually replicated across every CAMEL application model due to the need to annotate such a model or enhance it with arbitrary feature models. The reason why this domain is touched by the administrator lies also on the fact that this role is in charge of operating and enhancing the platform, so it has the complete knowledge of which part of the metadata schema is currently supported by this platform.

3.1.3.2 Core Domain

The core domain of CAMEL covers either general, abstract concepts or top-level concepts that can be re-used across different domains. The most remarkable concepts are *Model*, which represents any kind of model and *CamelModel*, which represents a CAMEL model. To be noted that a *CamelModel* is a *Model* and that any other sub-model kind in CAMEL (e.g., *DeploymentModel*) is also a *Model*. A *CamelModel* is a top-level container of sub-models of different kinds as well as of the main application at hand, which is represented by a coarse-grained *Component*. Other important concepts that are captured in this domain include the *NamedElement* and the *Feature* ones. The first concept, i.e., *NamedElement*, represents any kind of CAMEL element that has a specific name and textual description while it can be annotated through using the metadata schema (i.e., by referring to elements of a metadata model). This concept is a direct or indirect super-concept/class of any other concept/class in CAMEL. On the other hand, the second concept, i.e., *Feature*, represents a named element that can contain other feature elements, reaching an arbitrary level of nesting. This concept enables a CAMEL model to be arbitrary extended with an additional structure without the need to change the syntax of this language. As such, it is a super-concept of all concepts that require or allow such an extension. A feature element can also have a set of *Attribute*s that characterise it. The latter map to a generic concept which represents a named feature characteristic with a specific value, which can be taken from a specific domain of values (see valueType reference) and can map to a specific unit of measurement. Attributes can be either generic or can be further categorised into *QualityAttribute*s, i.e., attributes that characterise the quality of a specific feature/object. The latter can be further categorised themselves into *MeasurableAttribute*s, i.e., attributes that can be measured through the use of one or more sensors.
The following figure depicts a graphical representation of the abstract syntax of the core domain. In comparison to CAMEL 2.0, very slight changes have been conducted. These map to:

- the removal of the *Application* concept. This is not needed any more as an application can be represented as a coarse-grained software *Component*.
- the containment reference to the whole application as a *Component* and not as an *Application* element in the *CamelModel*.
- the removal of the containment reference to *Action*s, representing generic actions that can be performed by the platform, in the *CamelModel*. Actions can be either scaling actions (where the scalability domain is deprecated in Melodic) or platform actions which are actually captured in a different way in the execution domain. Thus, this is an unnecessary reference that is not needed and used at all by the Melodic platform.

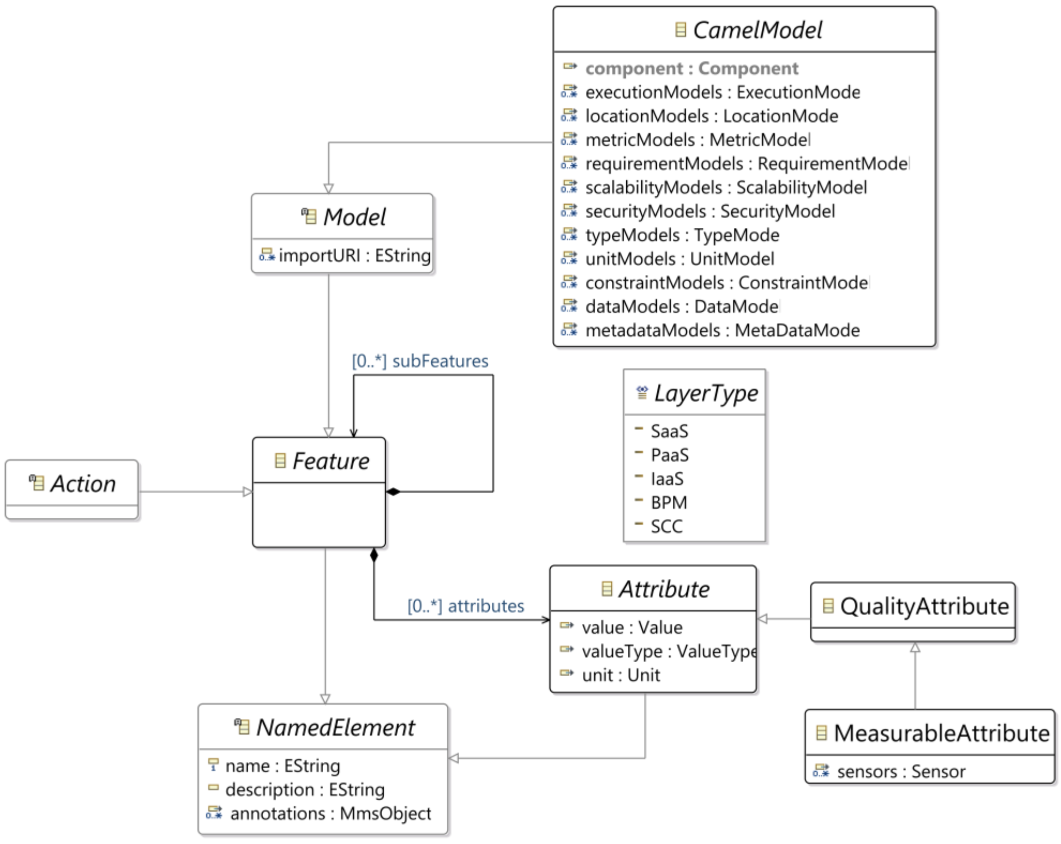


Fig. 3 - The core domain of CAMEL

3.1.3.3 Application Domain

This is a new domain in CAMEL that covers the specification of an application model, which is approximately similar to a deployment type model in CAMEL 2.0. Such an application model covers the architecture of an application, including all its possible variants in terms of the different forms that the application components can take. The resemblance to a deployment type model in CAMEL concerns the fact that both the structure and content of these models is similar. However, there exist particular differences, like the non-existence of hosting relationships that involve the hosted application components in an application model as such relationships do not characterise the architecture of an application. Another major difference concerns the fact that an application model includes all component configurations/forms, while a deployment type model contains one configuration/form per component. A final, important difference concerns the fact that in an application model each configurations of a component maps to a set of requirements while in a deployment type model, such a requirement set was exclusively associated with the component itself.

An application model is a kind of a component graph which contains components and their communication link ports as nodes and component relationships as arcs (between either components or communication port links). Each component is associated with a set of requirements that it might impose over its deployment as well as with additional information, concerning its current version, alternative configurations, the data that it might produce or consume, the data sources that it might manipulate, its provided and required communication links, whether it is long-lived or not, whether its instances are allowed to be situated on the same host, and finally the application models that can realise its functionality.

Through the latter containment association, it is possible to create a hierarchy of application models/components, where the root of a hierarchy represents the most complex component while going down further the hierarchy one can find components with an increasingly reduced complexity until he/she reaches the leaves of the trees. Such a hierarchy also represents decision points in terms of component/functionality realisation. In particular, when being in a particular place within the hierarchy, it could be decided whether the current complex component can be used as it is with its standalone/individual form (e.g., in case it is a micro-service), or the hierarchy should be descended to select simpler components that constitute this component and realise its functionality (e.g., functions that realise a different part of the functionality of the parent component).

Further, another novelty in terms of application models is that they can include references to external components, i.e., components that are not defined in the current CAMEL model. This, thus, enables to re-use existing, already-modelled components, as much as possible.

Two kinds of component relationships are currently covered by this domain. The first one is the communication relationship which is represented through the (communication) *Link* concept. The latter concept represents a communication link between two components through their provided and required (communication) link ports, respectively. A communication link does not only relate two components but can also pose requirements over the quality of the communication between these components which can affect the actual placement of these components in the multi-cloud landscape. To cover this, a *Link* in CAMEL 3.0 is associated with a *LinkRequirement*, which is a container of communication quality constraints.

Links are also associated to the configuration of their ports as well as with particular, technical semantics that govern the communication binding of components at the (deployment) instance level. In the latter case, the semantics indicate what should be the mapping of the paired components in terms of their instances during runtime, i.e., application deployment and execution. Such a mapping determines the low and upper multiplicity of the instances of both the source and target component in the communication. For instance, if the low and upper multiplicity of both components is one, then there should be a 1-to-1 mapping between the instances of these components. On the other hand, if the source component multiplicity is [0,1] and [1,1] for the target component, then this means that each source component instance should be mapped to exactly one target component instance but a target component instance is not certainly that will be bound to a source component instance.

The second component relationship captured concerns the location coupling between the related components. In particular, such a relationship specifies that a set of (application) components should be placed in the same location, where the location granularity can vary between such relationships. The variation highlights that the interpretation of location similarity can be different: it can mean the same host, the same zone, the same region or the same cloud. Further, the enforcement of such a relationship can be relaxed or obligatory. Relaxed means that the (cloud application management) platform should examine whether it is possible to host all the related components in the same location. If it is not, then the application deployment solution found can be followed irrespectively of that fact that it violates this relationship. On the other hand, in case that this relationship is obligatory, this means that it needs to be satisfied at all costs. Thus, if no possible deployment of the application can satisfy it, then the platform will fail to find a deployment solution for this application.

As it was indicated above, a component is associated with a set of requirements, which includes resources, PaaS, location, security, OS and image requirements. Further, it is possible to define a global set of requirements that cover all application components. Obviously, it is possible that there can be overlap between local and global requirements. In this case, the local requirements prevail as they have higher priority than the global ones. A novelty introduced in CAMEL 3.0 (in contrast to previous CAMEL version) concerns the fact that such a set of requirements can be also associated with the configuration of a component, something which is sensible if we consider that different application forms can come with their own, differentiated requirements. This signifies that fact that now we have three levels of prioritisation of requirement sets: global, component and configuration from the lower to the highest one.

A component configuration is represented in CAMEL via the *Configuration* concept. This is a super-concept which is further classified with specific and concrete configuration kinds. CAMEL 2.0 included 4 concrete configuration concepts named as *ScriptConfiguration*, *ClusterConfiguration*, *ServerlessConfiguration* and *PaaSConfiguration*. CAMEL 3.0 adds two new configuration kinds, named as *ImageConfiguration* and *ContainerConfiguration*. The semantics of such configuration kinds is now shortly explained below. More details along with specific, real-world examples can be found in: https://confluence.7bulls.eu/display/MOR/CAMEL+Configuration+Alternatives

- *ScriptConfiguration*: Enables to manage the lifecycle of an application component through respective OS-specific commands. The modeller could also specify the id of the image that can be used for the deployment of the application component, thus making the specified configuration as image-specific. Further, although still not supported by the Melodic platform, it is possible to rely on a devops tool and utilise tool-specific commands to manage the component lifecycle. In that case, the name of the tool should be specified. This configuration is more suitable for deploying components directly to VMs.
- *ClusterConfiguration*: enables to configure a task-based component so as to be properly deployed in a big data processing framework like Spark. The name of the framework needs to be specified in the form of an MDS annotation. In addition, the modeller should supply the URL of the binary code of the component and optionally the name of the class to be executed (in case of Java code). Finally, additional configuration details can be supplied like configuration parameters for the framework as well as runtime arguments for both the framework and the application component.
- *ServerlessConfiguration*: enables to properly configure a component that takes the form of a (serverless) function. Such a configuration is always serverless-platform-specific. For this configuration, the modeller should supply either the URL of the binary code of the component or a build configuration that clarifies how the component binary can be built from the source. He/she can also determine whether the platform should sense changes on the component binary or source code in order to redeploy it in the corresponding serverless platform used. Finally, some particular details can be specified in the form of an *EventConfiguration*, which highlight the name of the function endpoint and its bound HTTP method as well as the frequency of (platform-originating) calls that should be made to this component/function in order to keep warm its container.
- *PaaSConfiguration*: enables to configure a component in an underlying VM through the use of a standardised PaaS API. In this sense, the modeller should specify all necessary details, which include: (a) the endpoint of the API; (b) its version; (c) the name of the API; (d) the place from which the API-specific configuration of the component can be downloaded.
- *ImageConfiguration*: it plays a dual role. On one hand, it enables to just re-use and instantiate a (VM) image which already includes the application component (installed and running). On the other hand, it enables to instantiate a component's image within an environment that allows the exploitation of specific types of hardware-accelerated resources like FPGAs. To distinguish between these two cases, there is a boolean attribute which determines whether the (component) image is hardware-accelerated. Further, in the hardware-acceleration case, the modeller can also specify the kind of the hardware-accelerated resource(s) through an MDS annotation. This is a provider-specific configuration in the sense that each environment that can host a component image instance tailored to a specific cloud. As there can be multiple (VM) images that can enable such an environment within a specific cloud, it is recommended to associate such an image configuration with an (VM) image requirement.
- *ContainerConfiguration*: This is actually a split of the original *ScriptConfiguration* in CAMEL 2.0 in the sense that it is tailored specifically for container-based components, where the issuing of script-based commands is actually not necessary. In such a configuration, it is obligatory to specify the id of the component container image. Optionally, through an annotation, the actual container type (e.g., Docker) can be specified (essentially, this could be handy in case that the cloud application management platform supports multiple container kinds). Further, the modeller can also specify, when needed, some

environment configuration parameters through a specific feature as well as particular, container-tool-specific commands for starting the component container or updating it. Please note that such a configuration is container-tool-specific and not provider-specific, thus catering for multi-cloud application deployments.

The following two images showcase, on one hand, the components and their relationships, and, on the other hand, the component configuration alternatives. As this domain is new to CAMEL, there is no need to clarify changes with respect to CAMEL 2.0, although some comparisons and differentiation has been made clear in this section to clarify the change of focus and the optimisations/enhancements that have been performed over the specification of application architectures/deployment topologies.

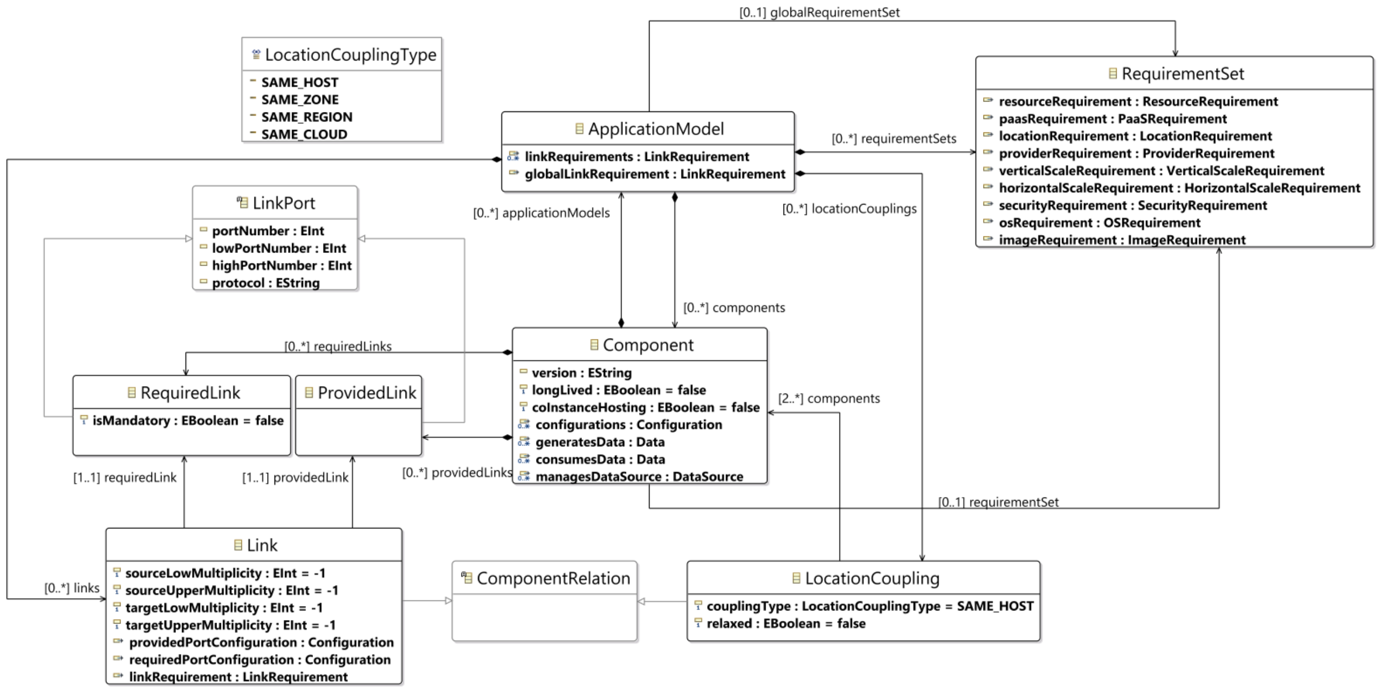

[0..1] globalRequirementSet
LocationCouplingType
SAME_HOST
SAME_ZONE
SAME_REGION
SAME_CLOUD
RequirementSet
resourceRequirement : ResourceRequirement
paasRequirement : PaaSRequirement
locationRequirement : LocationRequirement
providerRequirement : ProviderRequirement
verticalScaleRequirement : VerticalScaleRequirement
horizontalScaleRequirement : HorizontalScaleRequirement
securityRequirement : SecurityRequirement
osRequirement : OSRequirement
imageRequirement : ImageRequirement
ApplicationModel
linkRequirements : LinkRequirement
globalLinkRequirement : LinkRequirement
[0..*] requirementSets
[0..*] locationCouplings
[0..*] applicationModels
LinkPort
portNumber : EInt
lowPortNumber : EInt
highPortNumber : EInt
protocol : EString
[0..*] components
[0..*] requiredLinks
Component
version : EString
longLived : EBoolean = false
coInstanceHosting : EBoolean = false
configurations : Configuration
generatesData : Data
consumesData : Data
managesDataSource : DataSource
RequiredLink
isMandatory : EBoolean = false
ProvidedLink
[2..*] components
[1..1] requiredLink
[1..1] providedLink
[0..*] providedLinks
[0..1] requirementSet
Link
sourceLowMultiplicity : EInt = -1
sourceUpperMultiplicity : EInt = -1
targetLowMultiplicity : EInt = -1
targetUpperMultiplicity : EInt = -1
providedPortConfiguration : Configuration
requiredPortConfiguration : Configuration
linkRequirement : LinkRequirement
[0..*] links
ComponentRelation
LocationCoupling
couplingType : LocationCouplingType = SAME_HOST
relaxed : EBoolean = false

Fig. 4 - First part of application meta-model pertaining to components and their relationships

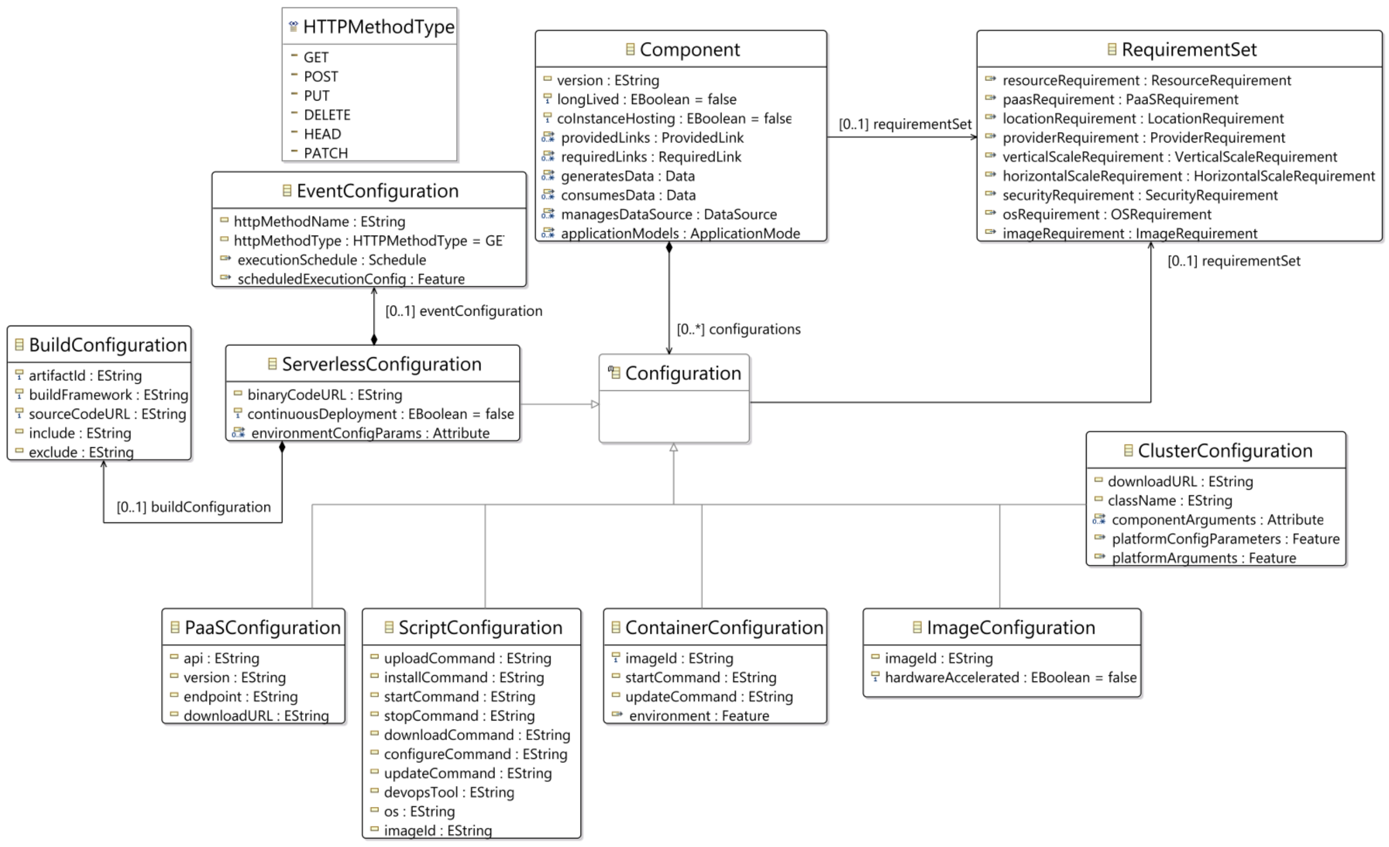

HTTPMethodType
GET
POST
PUT
DELETE
HEAD
PATCH
Component
version : EString
longLived : EBoolean = false
coInstanceHosting : EBoolean = false
providedLinks : ProvidedLink
requiredLinks : RequiredLink
generatesData : Data
consumesData : Data
managesDataSource : DataSource
applicationModels : ApplicationMode
RequirementSet
resourceRequirement : ResourceRequirement
paasRequirement : PaaSRequirement
locationRequirement : LocationRequirement
providerRequirement : ProviderRequirement
verticalScaleRequirement : VerticalScaleRequirement
horizontalScaleRequirement : HorizontalScaleRequirement
securityRequirement : SecurityRequirement
osRequirement : OSRequirement
imageRequirement : ImageRequirement
[0..1] requirementSet
[0..1] requirementSet
EventConfiguration
httpMethodName : EString
httpMethodType : HTTPMethodType = GE
executionSchedule : Schedule
scheduledExecutionConfig : Feature
[0..1] eventConfiguration
[0..*] configurations
BuildConfiguration
artifactId : EString
buildFramework : EString
sourceCodeURL : EString
include : EString
exclude : EString
ServerlessConfiguration
binaryCodeURL : EString
continuousDeployment : EBoolean = false
environmentConfigParams : Attribute
Configuration
[0..1] buildConfiguration
ClusterConfiguration
downloadURL : EString
className : EString
componentArguments : Attribute
platformConfigParameters : Feature
platformArguments : Feature
PaaSConfiguration
api : EString
version : EString
endpoint : EString
downloadURL : EString
ScriptConfiguration
uploadCommand : EString
installCommand : EString
startCommand : EString
stopCommand : EString
downloadCommand : EString
configureCommand : EString
updateCommand : EString
devopsTool : EString
os : EString
imageId : EString
ContainerConfiguration
imageId : EString
startCommand : EString
updateCommand : EString
environment : Feature
ImageConfiguration
imageId : EString
hardwareAccelerated : EBoolean = false

Fig. 5 - Second part of application meta-model pertaining to component configurations

3.1.3.4 Deployment Domain

The deployment domain in CAMEL 3.0 has been significantly modified. In particular, as indicated in the previous section, the variation in the component configuration is now covered in the application model while the deployment type model has been restrained to represent the application topology at the type level where the configuration of each application component is fixed.

As a deployment topology of an application can be considered as a graph, a new concept has been created which is named as a *Node*. This is a concept that is common also in other languages (e.g., TOSCA [10]). This concept can represent any kind of component, not necessarily a software component of the current application at hand. Thus, components like VMs, containers, PaaS platforms which can host application components can be also represented and modelled. In case that a node does represent an application component, then that component should be referenced along with its fixed configuration/form. In essence, a deployment type model is an outcome of the reasoning process, where the configuration of each application component is selected along with the resources that will be utilised to host it.

A node can also have a set of provided hosting ports as well as a required hosting port. In this way, we can also distinguish between components which can only host other components (e.g., VMs), components that can be only hosted (e.g., software components) and components that can host other components as well as be hosted by others (e.g., containers). In fact, hosting is one of the main relationship kind between nodes. While the main change with respect to CAMEL 2.0 is the fact that arbitrary hosting relationships can be formed (e.g., a software component is hosted in a container which is hosted in turn in a VM). This is a powerful modelling feature that is well-suited to the need to support different kinds of component forms that require different hosting topologies to be formulated. Please note that always a hosting relationship, represented by the *Hosting* concept, enables to connect one provided hosting port with all the required hosting ports that it hosts, and thus maps to a 1-N relationship between the host and the hosted component. Further, a hosting relationship enables to configure properly the different ends through the use of respective configuration elements.

In this node graph that is represented by a deployment type model, another core node relationship that is captured is the *Communication* one, which represents a communication relationship between nodes/application components. Such a relationship covers any kind of communication that happens between two components/nodes, irrespectively of the communication direction. In this respect, such a relationship, apart from referring to the two related nodes, it also

provides references to all the (communication) links between these two nodes. Where each link also includes a configuration point at each end of the communication (as indicated in the previous section).
The following figure showcases the content of the deployment type model at the conceptual level by hiding the actual concept of a *DeploymentTypeModel* in order to reduce the complexity of the concept relationships and thus enhance the readability of the figure. In order to keep the size of the deliverable appropriate, we have also neglected the analysis of a deployment model at the instance level also with the rationale that instance-based concepts are more or less similar and have counterparts at the type level while a full analysis of this deployment domain part will be supplied in the full CAMEL 3.0 documentation. Finally, due to the fact that a comparison between a deployment type model in CAMEL 2.0 and an application model in CAMEL 3.0 has been already supplied (in the previous section), there is no reason of repeating the changes that have been conducted in CAMEL 2.0 to produce CAMEL 3.0 for the deployment domain.

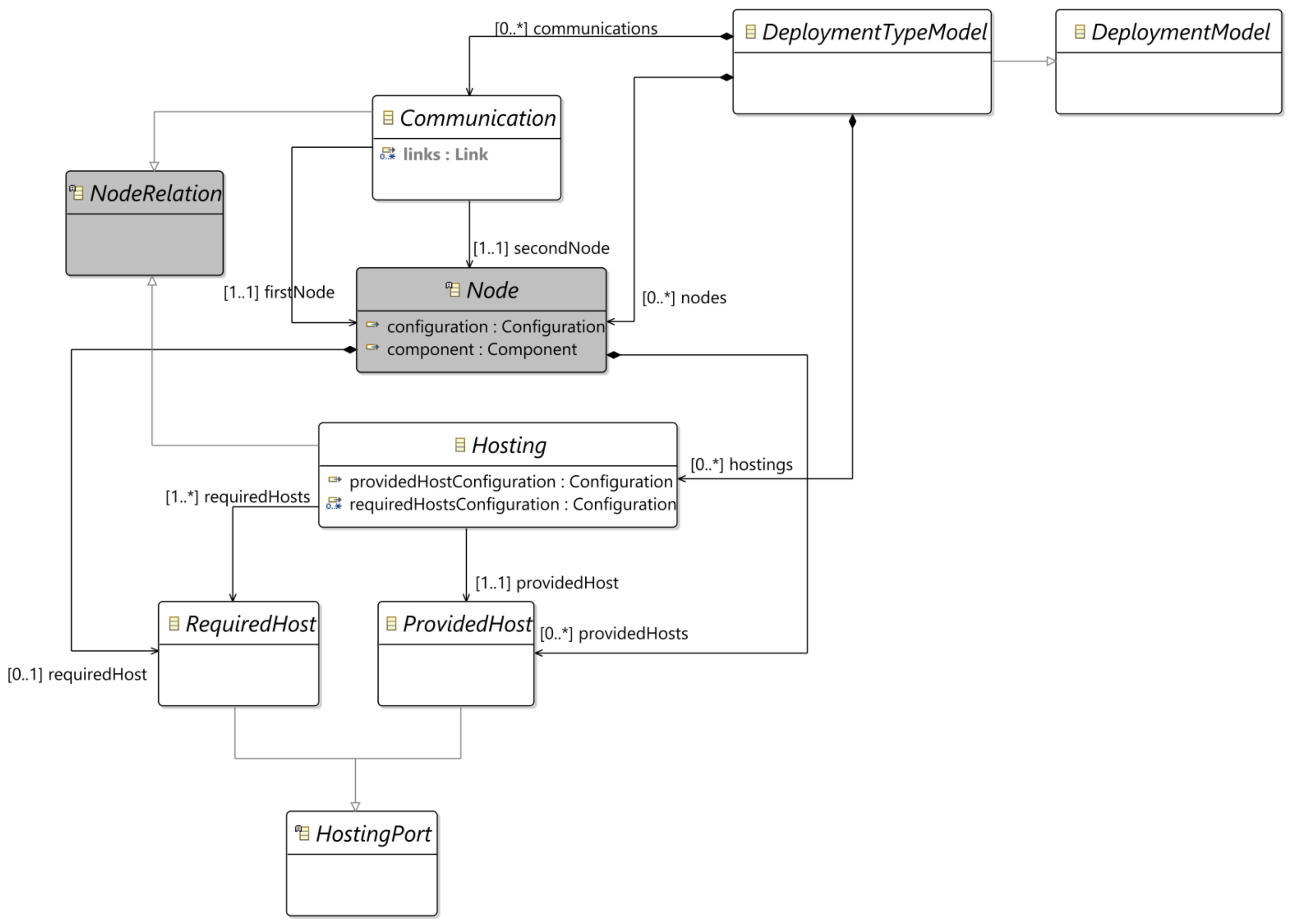


Fig. 6 - The deployment meta-model of CAMEL (type level)

3.1.3.5 Requirement Domain

This is an important domain in CAMEL as it enables to specify all application and component requirements that need to be met for a specific application by its management platform. In comparison to CAMEL 2.0, CAMEL 3.0 introduces only a very slight change, which concerns the addition of the *LinkRequirement* concept. Thus, in the analysis that follows, we supply a short summary of what CAMEL 2.0 already captured and we focus on the semantics of the newly introduced concept.
Any requirement in CAMEL, represented by the *Requirement* concept, is a kind of a feature; this means that it can be extended through MDS annotations and arbitrary feature models that specify constraints on MDS-specific attributes and concepts. Requirements can be hard or soft. Hard requirements need to be met at any cost by the application management platform. On the other hand, the platform will attempt to satisfy soft requirements in a best-effort basis.
Hard requirements can be further distinguished into provider, image, resource, PaaS, OS, Service Level Objective (SLO), location, security, horizontal/vertical scale and link requirements. Provider requirements restrain the

deployment space into a set of specific cloud providers. Image requirements specify the ids of the images that can be used to instantiate the VMs on which application components can be hosted. Resource requirements enable to specify constraints over the characteristics of resources through the use of feature models. Similarly, PaaS requirements enable to specify constraints over the characteristics of a PaaS environment again through the use of feature models. OS requirements enable to constrain the OS on which a component can be deployed. Service Level Objectives are constraints that can be posed over the performance of an application component or the whole application at hand. Location requirements restrain the deployment of an application component over a specific set of locations. Security requirements restrain the deployment of application component(s) over only those cloud providers that satisfy and implement a set of security controls. Horizontal scale requirements restrain the number of instances that an application component can have in a certain range. Vertical scale requirements can again be mapped to feature models but are not currently supported by the Melodic platform.
The new kind of hard requirement introduced in CAMEL 3.0 is the *LinkRequirement*, i.e., a requirement over the communication between two application components. Such a requirement can be posed over the quality characteristics of the components communication through the use of an arbitrary feature model. As such, the *LinkRequirement* concept is empty in content but is an indirect sub-concept of *Feature*.
Only one specific soft requirement sub-kind has been modelled in CAMEL which is named as *OptimisationRequirement*. In such a requirement, the modeller can specify the optimisation goal, i.e., to maximise or minimise, as well as the (mathematical) expression to optimise which maps to a metric variable. This requirement kind is quite critical in order to produce optimal application deployments as it enables the evaluation of the utility of such deployments through the use of the metric variable. Thus, it really caters for the utility-based and thus optimal deployment of multi-cloud applications.
The following figure showcases the core content of this domain for all the requirement kinds analysed. We utilise the gray colour to showcase the sole change/addition conducted over CAMEL 2.0.

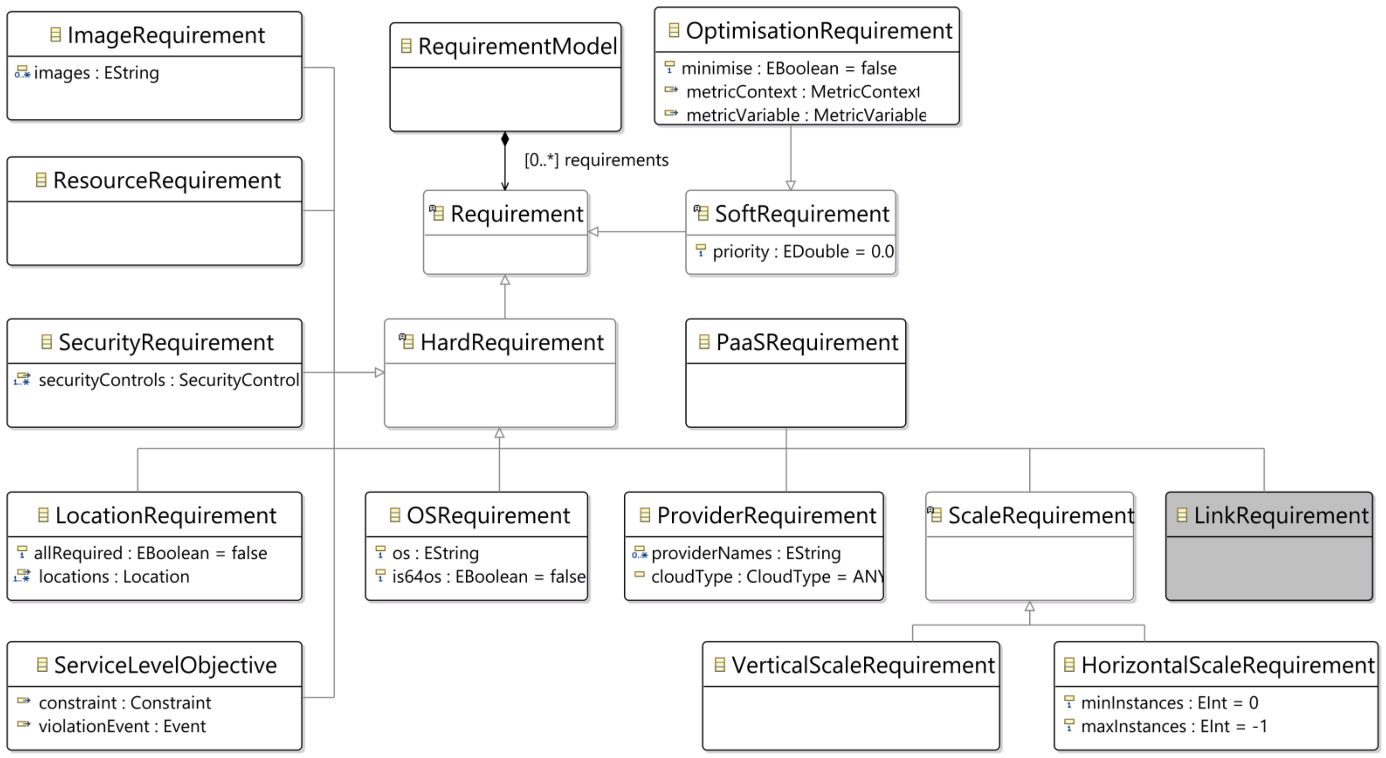


Fig. 7 - The requirement meta-model of CAMEL

3.1.3.6 Metric Domain

This domain concerns the way the quality of an application can be measured, thus tends to supply all details in order to guarantee the measurability of multi-cloud applications. Similarly to the deployment domain, the models@runtime [5] approach is followed with the rationale that the platform needs to maintain the state also for the application's monitoring infrastructure, especially as the application adaptation directly influences it. Due to keeping the size of this deliverable suitable as well as focusing mainly on devops users, we do not provide details about the instance level of

this domain. Essentially, also for the reason that this level has not been modified in CAMEL 3.0. Thus, the analysis focus is mainly on the type level.
The core concept of this domain is *Metric*, which represents any kind of metric. A metric references a metric template, which supplies some common information among similar metrics, like the (quality) attribute being measured, the unit of measurement and the value type. Similar metrics can be considered as those which can be computed from each other. For instance, a metric of *raw response time* and a composite metric of *average response time* have as common all such information (e.g., the value type can be from 0 to positive infinity and the measurement unit can be seconds while both measure the quality attribute of *response time*).
A metric can be single/raw, composite or a metric variable. A single metric can be computed from sensors. A composite metric is computed from other metrics through the use of a (mathematical) formula. A metric variable can be considered as a special kind of metric which is computed during deployment reasoning. In other words, it takes a value that can rely on the (optimal) deployment solution computed by the platform or on the node candidates that match the requirements of a certain application component. This is its main actual distinction from the other kinds of metrics which are computed directly or indirectly from sensors. A metric variable can concern a specific application component and might be computed over the node candidates of that component. It can be single or composite. A single metric variable runs over each node candidate of a component. For instance, a metric variable of cost runs over the cost of each node candidate. In other words, its actual value can depend on the deployment solution derived (and thus the actual node candidate selected to support the deployment of the application component at hand). A composite metric variable applies a certain (mathematical) expression over other metric variables. For instance, maximum cost over the node candidates of a certain component can be computed by applying the max function over the single metric variable of cost per node candidate. In this case, such a cost is more or less irrespective of the deployment solution in the sense that all node candidates of a component have to be computed before the actual deployment reasoning problem is solved.
CAMEL 3.0 introduces a new metric kind which is named as *PredictedMetric*. This is considered as a (special) composite metric which can be computed out of the measurements of a certain metric by applying a specific prediction method as a mathematical function (so the mathematical formula will be the application of the prediction method on the metric to be predicted). However, there are some additional details that need to be supplied for such a metric including the prediction horizon, i.e., for how long the prediction accuracy is guaranteed in the future, and the prediction probability, i.e., the probability of occurrence of the predicted metric value. The prediction horizon is also associated with the actual time-based (measurement) unit concerned (e.g., seconds or minutes). Through this new metric kind, it is now possible to affect the deployment solution of an application as predicted metrics can participate in SLOs and utility formulas.
The metrics, but not the metric variables, are associated with a metric context whose complexity depends on the metric complexity. A metric context carries important details that affect how a metric is measured like the window (of measurement), the (measurement) schedule (i.e., the measurement frequency) and the object context. The latter reflects the actual object that is being measured which can be a (application) component, a data (item) or, now in CAMEL 3.0, a (communication) link. A metric window can be time-based (it is considered full when a certain time period has passed), size-based (it is considered full when a set of measurements with a specific size have been collected) or mixed (both time- and size-based). It can be also a sliding (current measurements stored slide to accommodate space for a new measurement to be inserted) or normal window (full windows become empty to accommodate space for new measurements). A metric schedule determines the frequency of measurement, i.e., how often a metric can be measured by specifying the time period of this frequency and additional, optional details like when the measurement can be started and ended and how many times it can be repeated.
Single/raw metrics are associated with single/raw metric contexts, which also determine the actual sensors that will be used for measuring these metrics. On the other hand, composite metrics (including predicted ones) are associated with composite metric contexts, which determine the way the component metric measurements should be grouped and what are the component metric contexts. As such, the composite metric contexts dictate the exact way composite metrics can be measured from other metrics based on both their (mathematical) formula and the way the measurements of the component metrics can be computed and grouped before being applied in that formula.
Finally, a *Sensor* can be considered as a component which is responsible for producing the measurement values of single/raw metrics. Such components are either supplied by the platform or they are realised by the devops and become part of the application deployment architecture/topology. Sensors can be either pull or push based. When sensors are offered by the platform, we need to supply particular configuration details for them (e.g., what is the implemented class for the sensor and which metric from those that can be measured by the sensor should be computed - all in one configuration string). Please also note that sensors can be also part of an application or can be external (to the platform). In that case, we do not need to supply any kind of detail about the sensor apart from the fact that it can

be pull or push based. It will be then the responsibility of the sensor to properly supply its computed measurements to the right components of the platform.
The following figures show the metric and context parts of this domain where the grey colour is used to denote new concepts or properties/attributes that have been introduced in CAMEL 3.0.

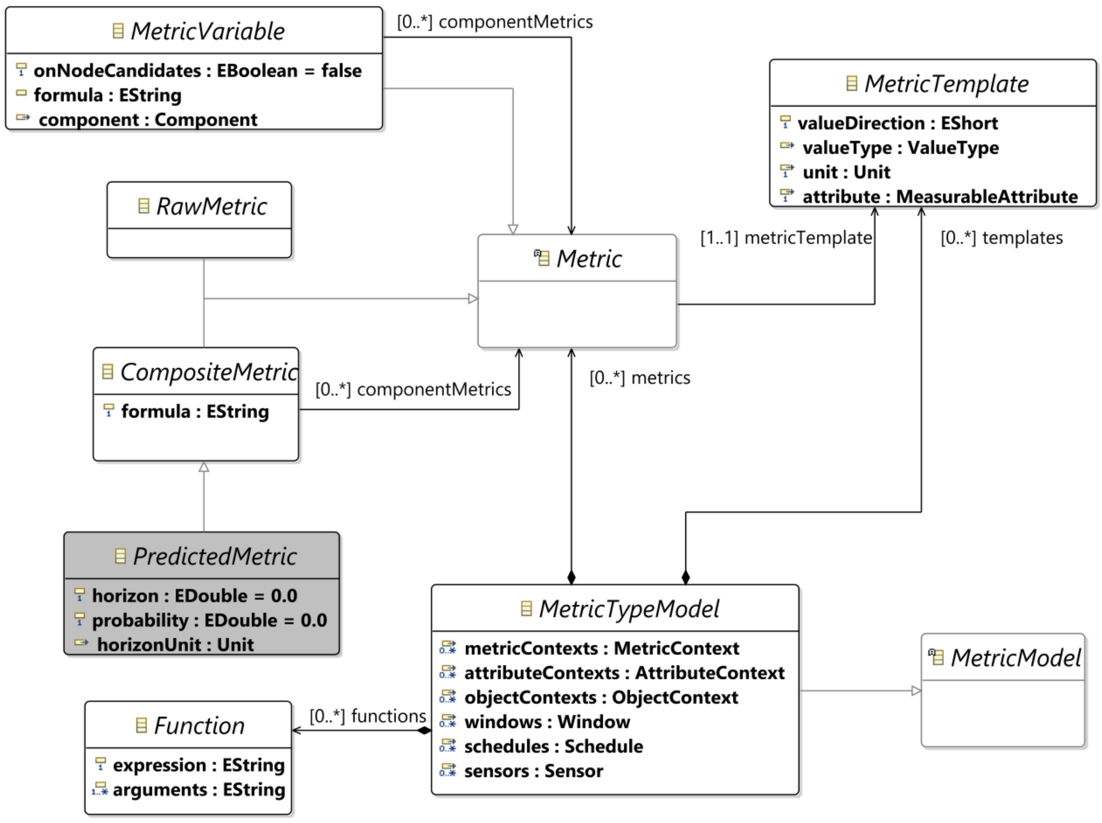


Fig. 8 - The different kinds of metrics in the metric meta-model

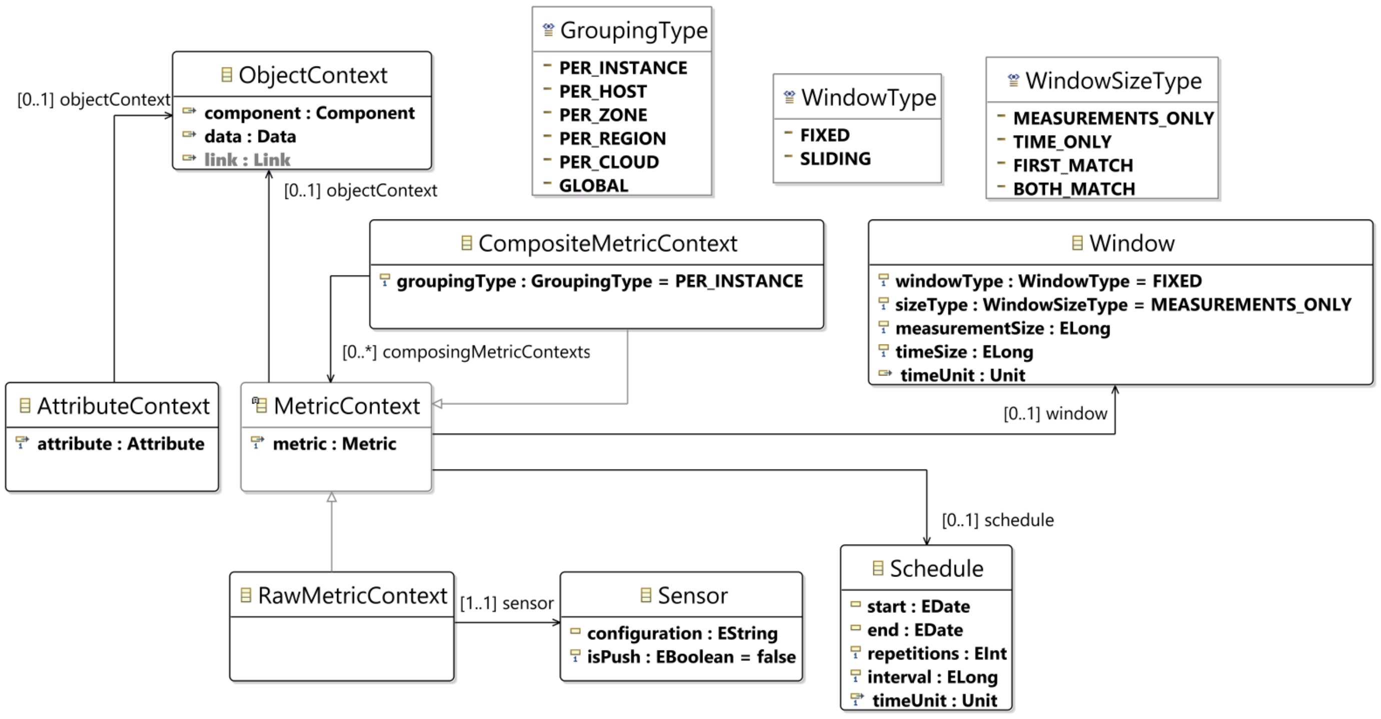


Fig. 9 - The context part of the metric meta-model

3.1.3.7 Data Domain

The data domain in CAMEL follows again the models@runtime approach as it is considered a crucial domain and we need to constantly keep an eye on its state as it can continuously change and can have an impact on the deployment/reconfiguration of an application. This domain has been slightly modified in CAMEL 3.0, mainly in order to keep the right references to the right concepts from the application model (and not the deployment type model as it was the case in CAMEL 2.0).
In this domain, at the type level, two main concepts are introduced: *Data* and *DataSource*. The first represents a data (item) which is maintained by a data source and might include other data items inside it. Such a data item might be consumed or produced by application components, as it was indicated in section 3.1.3.3. A *DataSource* is a source of data (items), i.e., it can be considered the host and management medium for such data. A data source can be external, so it can be outside of the management scope of the platform but can influence the placement of components and other data sources/items, or internal, in which case it maps to an application component that manages it. At the same level, we have also the *DataTypeModel*, a container of both data and their sources.
At the instance level, similarly named concepts have been introduced with similar semantics but they now represent concrete data (items) and sources. Further, such concepts directly refer to their type. In the case of *DataSourceInstance*s, it should be also explicated that if they map to an internal data source, they have to be associated with the respective instance of a node that manipulates them. Thus, while at the type level, data sources are associated with application components, at the instance level, data sources are associated with node instances, i.e., instances of application components.
The following figure depicts the conceptual model of this domain where with grey colour the main changes introduced in CAMEL 3.0 are highlighted.

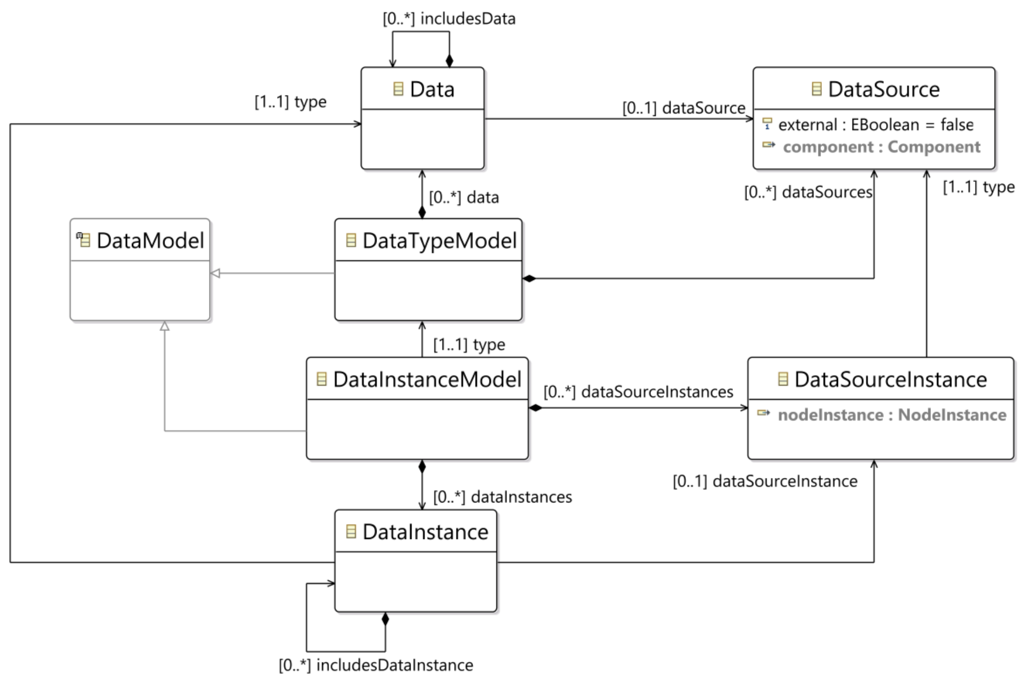


Fig. 10 - The data meta-model in CAMEL

3.1.3.8 Execution Domain

This domain has been designed in order to maintain the execution history of an application. Each deployment episode, which includes the deployment, reconfiguration and undeployment of an application, maps to a certain container of all relevant elements that is named as *ExecutionModel*. Such a model specifies various details about the respective deployment episode, such as what was its start and end point in time, what was its overall (deployment) cost, and what were the requirements and data type models that led its evolution over time, i.e., the evolution of the application's deployment episode. In addition, this model includes various elements that were created/occurred during the application's deployment episode, such as measurements, SLO violations, (scalability) rule triggers (deprecated for the Melodic platform) and history records.

Measurements can be of different kinds but all share some common information, like what is the measurement value and at what point in time this measurement was produced as well as what was the instance of the metric used to produce it. A *NodeMeasurement* (newly introduced in CAMEL 3.0) represents the measurement of any kind of node like a component, a container or a VM. This concept replaces a set of similar concepts that are part of CAMEL 2.0, i.e., the *VMMeasurement*, *ContainerMeasurement*, and *SoftwareComponentMeasurement* ones. A node measurement can be associated to either a node or a node instance. This depends on the nature of the metric (instance) involved. In particular, if the metric is computed over the measurements of all instances of a node, it maps to the node itself. On the other hand, if it is computed from measurements of only one node instance, it maps directly to that node instance only.
A *LinkMeasurement* is another newly introduced concept which enables to measure the quality of a communication link. Similarly to the case of *NodeMeasurement*, a link measurement can be associated with either a link or a link instance. A *CommunicationMeasurement* (a concept from CAMEL 2.0) has a similar scope but attempts to measure the quality of a set of communication links between two components. It is also associated with either a communication or a communication instance.
DataMeasurement is the last kind of measurement that represents measurements related to data metrics (instances). Similarly to the case of the other measurement kinds, this measurement can be associated with either a data or a data instance.
SLO violations, as their name witnesses, indicate the occurrence of an event of an SLO violation. In this respect, they refer to the measurement that led to this violation, the SLO that has been violated and the time point in which the SLO assessment took place. Rule triggers indicate the occurrence of scalability rule triggering, which can happen when the event mapping to the respective scalability rule has occurred. Apart from indicating the scalability rule that has been triggered, rule triggers also specify the point in time that this triggering has occurred.
Finally, a *HistoryRecord* can be considered as a recording of the transition of the application from one deployment or data instance model to another one either in the context of application reconfiguration or due to the production of specific data instances. In either case, this recording also captures other information like what was the cause of the transition, what was its start and end time and what is its type. Further, it also includes *HistoryInfo* elements which capture all the actions that have been performed by the platform in order to realise the transition (along with their timing and affected objects).
The graphical representation of the abstract syntax of this domain is depicted in the following figure where again in grey colour we can see the changes conduced in the context of CAMEL 3.0.

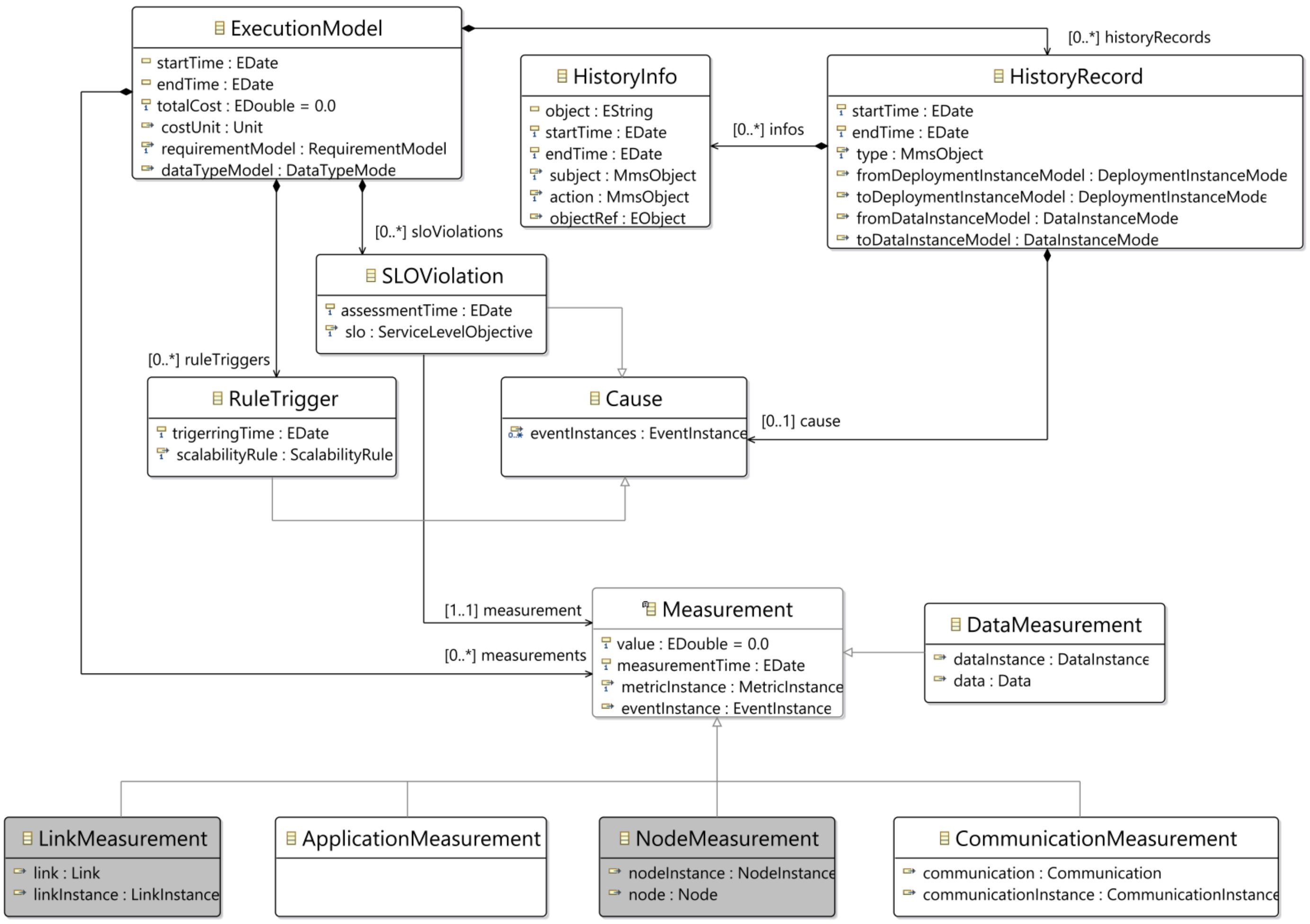


Fig. 11 - The execution meta-model of CAMEL

3.1.3.9 Metadata Domain

This domain has been developed in CAMEL in order to support the capturing of the MDS and its use in annotating CAMEL elements as well as arbitrary feature models that extend, at the model level, these elements. In this domain, the respective container/model kind is named as *MetaDataModel* and contains any kind of MDS element. *MmsObject* is a concept that represents any kind of MDS element. Thus, it conveys all the common information across all MDS element kinds, like the element id, name, URI and (textual) description. CAMEL 3.0 introduces another information piece named as *implemented* that clarifies whether the respective MDS element is supported by the platform (i.e., the platform takes into account its use in annotations and can exploit such an annotation for one or more reasons, especially if it concerns the specification of attributes - e.g., node candidate filtering or deployment reasoning).
The different kinds of MDS elements include concepts, properties, concept instances and property instances. A MDS concept is represented by *MmsConcept*, which includes information like the parent MDS concept of this MDS concept as well as its properties and instances. A MDS property is represented by *MmsProperty*. Such a property can be an object or data property. In the first case, it relates two MDS concepts, so both of them need to be referenced. In the second case, it relates on MDS concept with a specific data type (from which the property can take its values) mapping to a specific URI. A MDS concept instance is represented by *MmsConceptInstance*, which is associated with all its MDS property instances. The latter are finally represented by *MmsPropertyInstance*. An MDS property instance relates to its type and might have either a single value or a concept instance (value) depending on the kind of property its type is (i.e., where its type is a data or object MDS property).
The following figure depicts the conceptual model of this domain where with the grey colour we can see the single update that has been conducted to CAMEL 2.0 in the context of CAMEL 3.0.

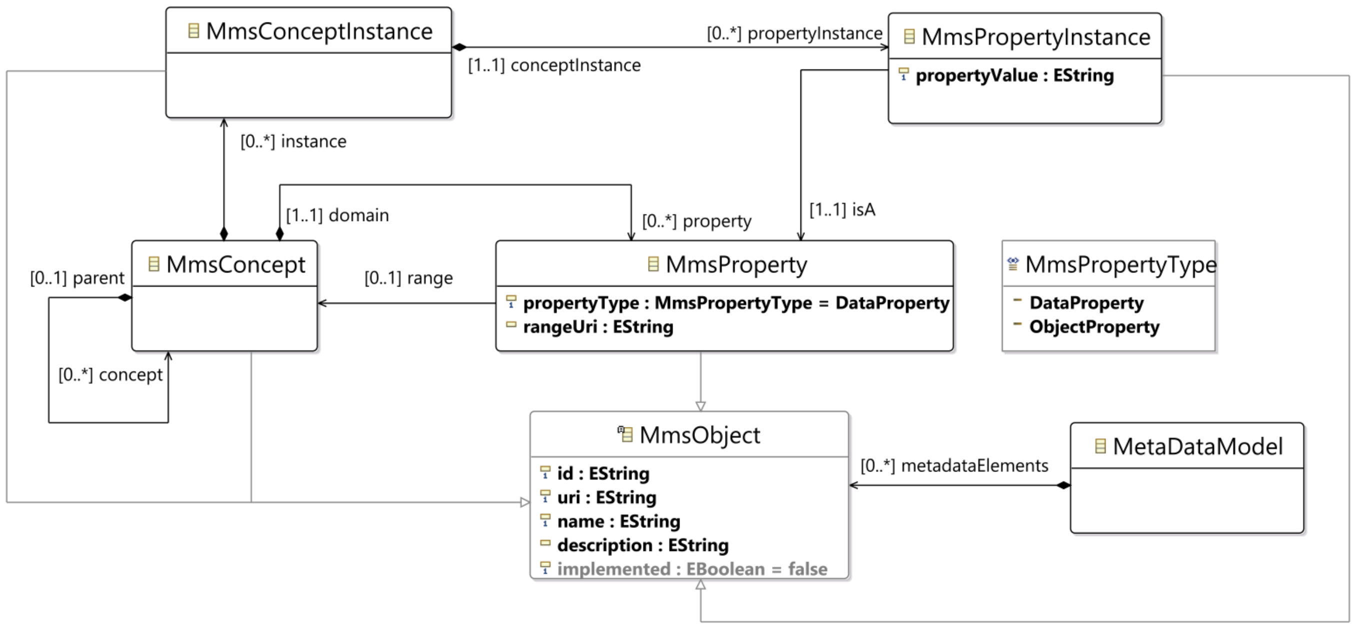


Fig. 12 - The conceptual model of the metadata domain in CAMEL

3.1.3.10 CAMEL 3.0 Benefits

We close the analysis of CAMEL 3.0 by supplying the benefits of this new version of CAMEL which mostly correspond to the initially analysed requirements in section 3.1.1:

- CAMEL 3.0 enables to re-use not only individual components but whole applications. This enhances application development (as the user could discover and re-use existing components from other applications) as well as speeds up the modelling of multi-cloud, polymorphic applications.
- CAMEL 3.0 supports polymorphic application modelling as it allows the specification of multiple forms per application component as well as the requirements mapping to each form. The Reasoning process benefits from this specification as it can select the most suitable form as well as the most suitable resources to support the execution of each application component by considering both requirements and optimisation objectives at both the component/local and application/global level.
- In CAMEL 3.0, the user does not have to specify any kind of deployment model and any kind of hosting topology within it. Everything is automatically derived by the corresponding (multi-cloud) application management platform that exploits CAMEL models.
- CAMEL 3.0 supports communication/network-awareness as it enables to specify communication constraints and requirements as well as metrics focusing on measuring the communication between application components. This enhances the portfolio of use-cases that CAMEL can support.
- CAMEL 3.0 resembles TOSCA at the deployment domain and can enable the specification of complex hosting topologies between various kinds of components (e.g., application software, VMs, containers, operating systems, etc.).

3.2 Language Implementation

CAMEL 3.0 was implemented by extending both the abstract and textual syntax of CAMEL 2.0. The abstract syntax was enhanced by modifying CAMEL's meta-model in ECORE[3]. Such a modification included the incorporation of new classes, attributes and properties, the migration of existing ones in different places as well as the deletion of the unnecessary ones. 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

[3] https://www.eclipse.org/modeling/emf/

OCL[4] Editor[5] of the Eclipse Environment[6]. 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[7] model of CAMEL to comply to the new CAMEL version (in terms of the abstract syntax elements) via the use of the Xtext Editor of the Eclipse Environment[8]. 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 CAMEL 3.0:

- Source-code:
https://bitbucket.7bulls.eu/projects/MEL/repos/camel/browse/camel?at=refs%2Fheads%2Fcamel_3.0
- Meta-model:
  https://bitbucket.7bulls.eu/projects/MEL/repos/camel/browse/camel/camel/model/camel.ecore?at=refs%2Fheads%2Fcamel_3.0
- 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

This section introduces an extensive update of the vocabulary entitled *Metadata Schema* (MDS) which was introduced as part of the Melodic project [18]. This schema aggregates a number of classes and properties that correspond to concepts used for describing requirements, constraints and offerings' characteristics in multi-cloud placement decisions. The structured description of these characteristics 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. 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 work conducted in Morphemic's WP1, MDS was enhanced to cover the desired abstraction constructs, which will be used in CAMEL for modelling polymorphic applications, regarding the areas of data management, polymorphic application design as well as the heterogeneous resources and platforms to be used. Besides the data aspects, the extensions involve concepts and properties related to the modelling of various types of resources and platforms, including HPC resources, accelerators, as well as serverless capabilities along with network related aspects that should drive or affect the cloud application deployment. Currently, 395 updates and additions have been introduced, taking also into account the MORPHEMIC use case requirements analysis [4]. These involve 109 updates or new additions regarding classes and subclasses and 286 additions of object and data properties.

[4] https://www.omg.org/spec/OCL/
[5] https://projects.eclipse.org/projects/modeling.mdt.ocl
[6] www.eclipse.org
[7] https://www.eclipse.org/Xtext/
[8] https://www.eclipse.org/Xtext/documentation/308_emf_integration.html

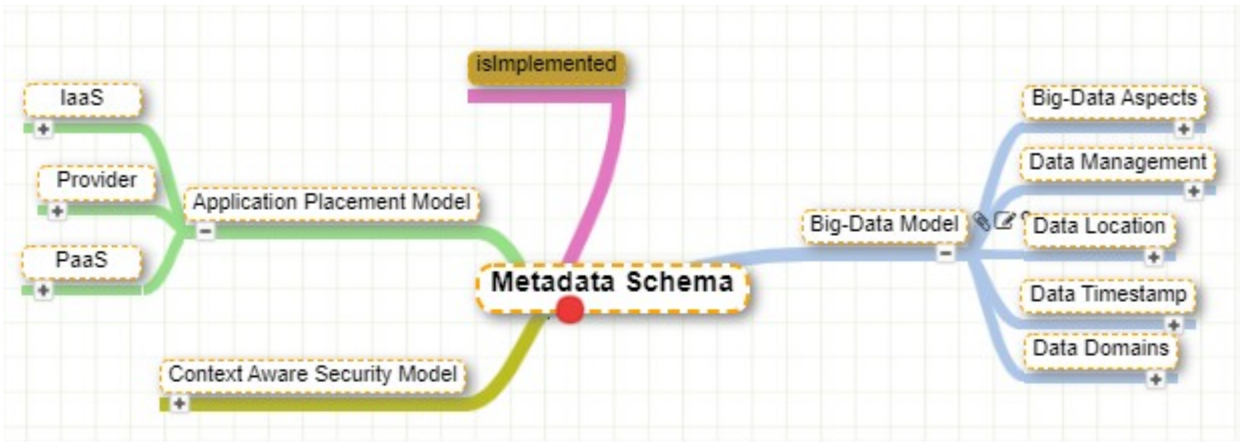


Fig. 13 - Metadata Schema overview

In Fig. 13 a high-level overview of the Metadata Schema (MDS) is provided in order to depict the top-level classes of this schema. One of the first additions is a new top-level property: *isImplemented* that is inherited to all the subclasses of our model. This is a data property with *xsd:Boolean* values that will be used in order to define which classes and properties modelled in the Metadata Schema can be used for a certain application modelling that will drive its management by the MORPHEMIC platform. The *true* value for this property implies that the relevant information can be retrieved when fetching node candidates or can be measured through dedicated sensors once used. Therefore, the MDS entities that can be used in CAMEL when modelling a certain application should have a true value for their *isImplemented* data property.
It could be argued that this data property is not needed as all elements of MDS should be implemented in the Morphemic platform. However, this is not the case for the following reasons: (a) this metadata schema supplies a taxonomy of relevant concepts, thus higher-level concepts are not meaningful to be included in CAMEL model specifications and be also implemented in the platform; (b) some new MDS elements might not be implemented in the platform due to resource restrictions and the prioritisation of the relevant features to be realised; (c) the value of MDS goes beyond the Morphemic platform as MDS can be re-used in the context of multiple platforms, either in the form of research prototypes or stable, commercial implementations.
All the extensions have been described in a tabular format, which follows the description of the initial version of MDS [18] and provides the hierarchy of classes and properties along with their short description. Due to the size of the tables, these are provided in the Appendix I of this deliverable. For each of the main extensions of the model, we now provide in respective sub-sections a short description of the extension along with an abstract figure (i.e., mind map notation) that gives a bird's eye view of the new classes and properties introduced as well as a fine-grained depiction of the extension through UML class diagrams, where the reader may find out the value types of the object and data properties. A more complete view in form of a high-resolution image of the complete MDS taxonomy can be found here[9].
Please note that we have utilised a specific colouring scheme in the abstract figures where the white colour denotes classes, the yellow colour properties of classes and the red colour instances of classes.

4.1.1 Processing

In Fig. 14, we provide the abstract view of the MDS updates regarding the *Processing* class (which is subclass of *Application Placement Model/IaaS*). This class involves any infrastructural feature bound to the processing capability of virtualised resources. The classes *Processing, CPU* and *Memory* have been enriched with additional properties, while new important classes have been added. In Fig. 15, the updated abstract view of the *Memory* class is given.

[9] https://melodic.cloud/UuTf-KRW.png

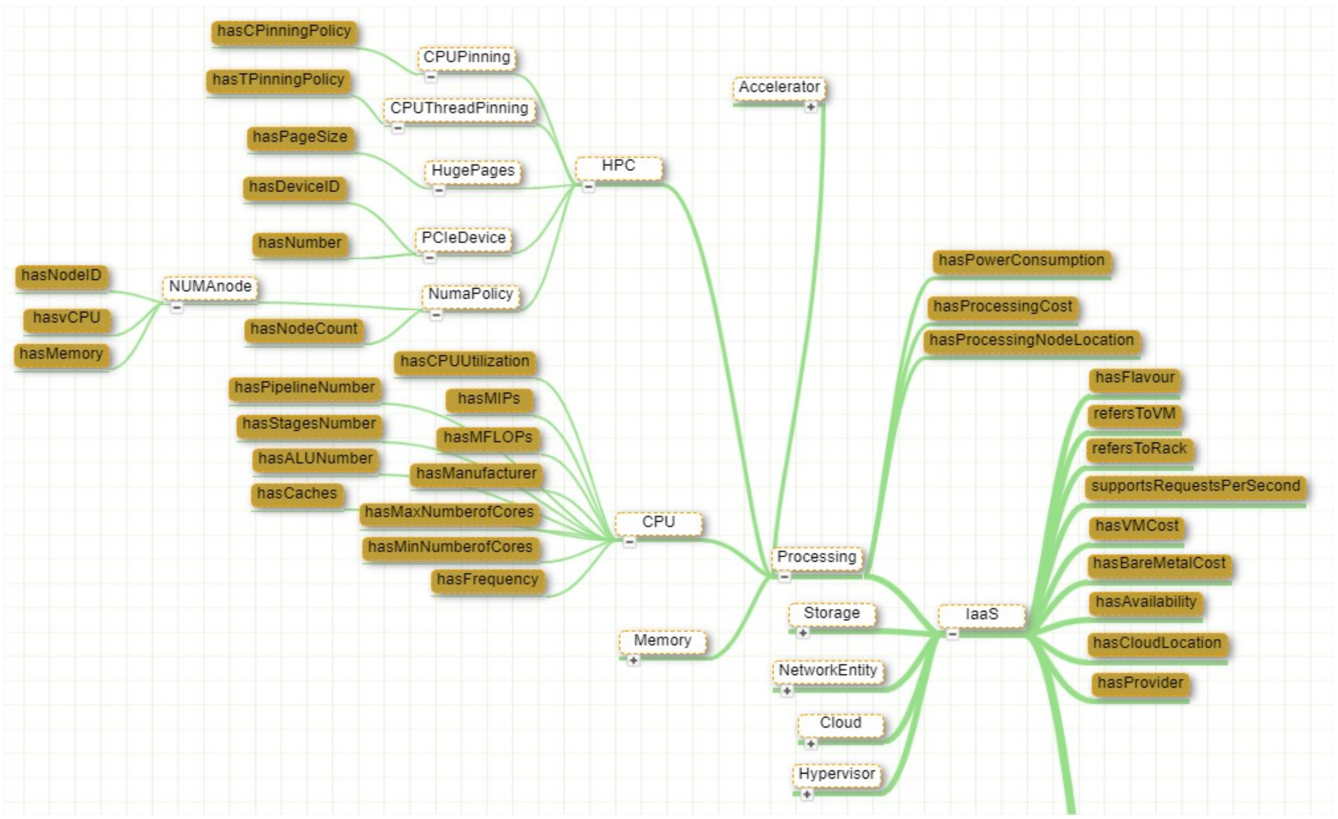


Fig. 14 - IaaS high-level as well as CPU and HPC classes

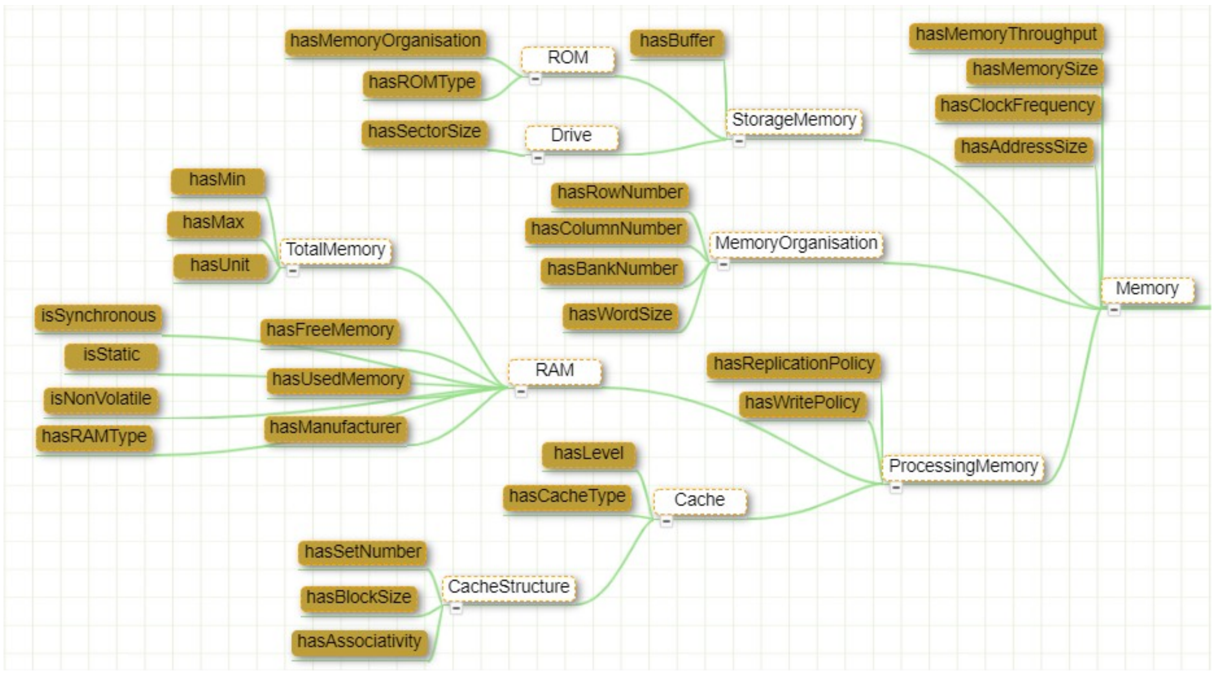


Fig. 15 - The memory class hierarchy

In addition, *HPC* (see the respective hierarchy in Fig. 14) and *Accelerator* (see the respective hierarchy in Fig. 16) classes were introduced. *HPC* refers to the high performance features provided by the computing platform which can boost the overall performance of an application. The *Accelerator* class refers to application-specific hardware designed or programmed to compute operations faster than a general-purpose computer processor. It involves the following main subclasses:

- *GPU* (see Fig. 16) which 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.

- *ASIC* (see Fig. 16) which refers to application-specific integrated circuits (ASICs), a category of hardware accelerators that employ strategies, such as optimised memory use and the use of lower precision arithmetic, to accelerate calculation and increase the throughput of computation.
- *VPU* (see Fig. 16) which refers to the vision processing unit (VPU), a category of microprocessors intended to accelerate machine vision algorithms and tasks.
- *FPGA* (see its hierarchy in Fig. 17) which refers to IaaS resources that use field programmable gate arrays (FPGAs), as integrated circuits made to be configured by the user after manufacturing.

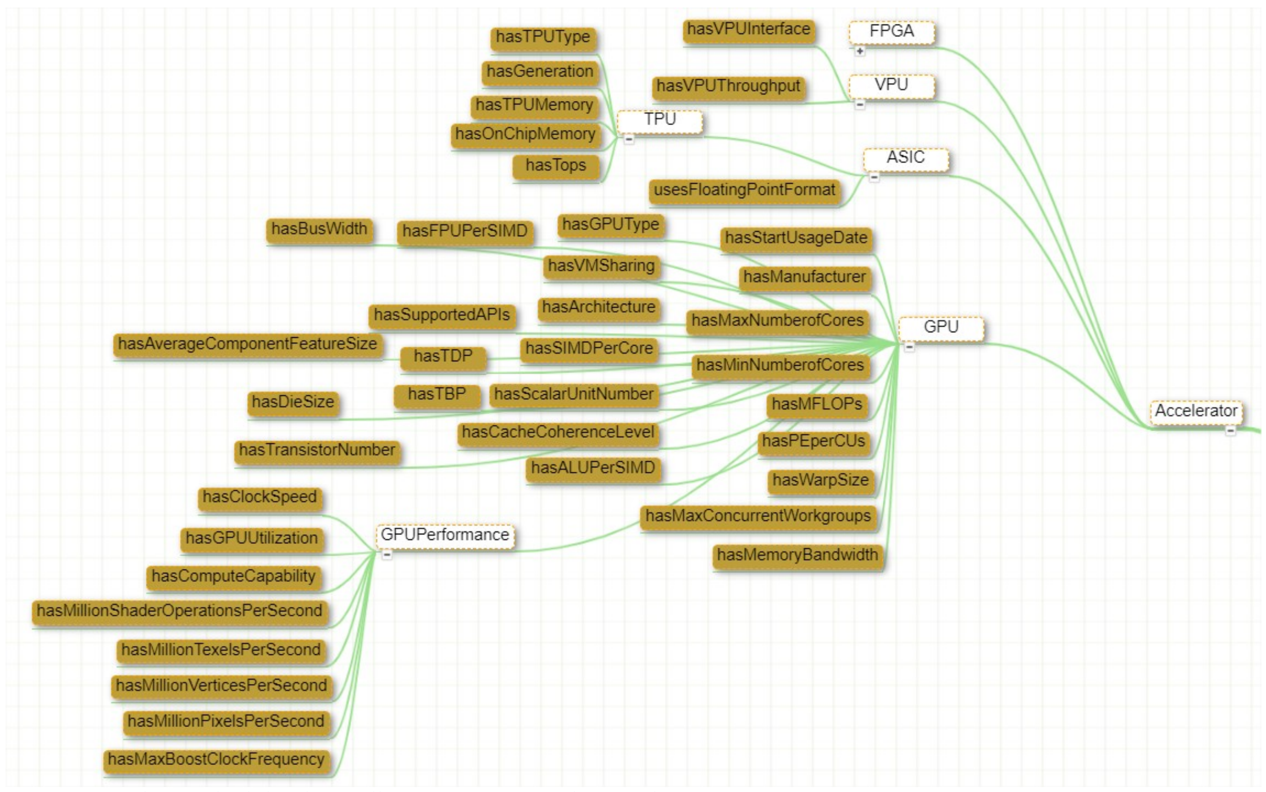


Fig. 16 - Accelerator high-level as well as GPU, ASIC, and VPU Classes

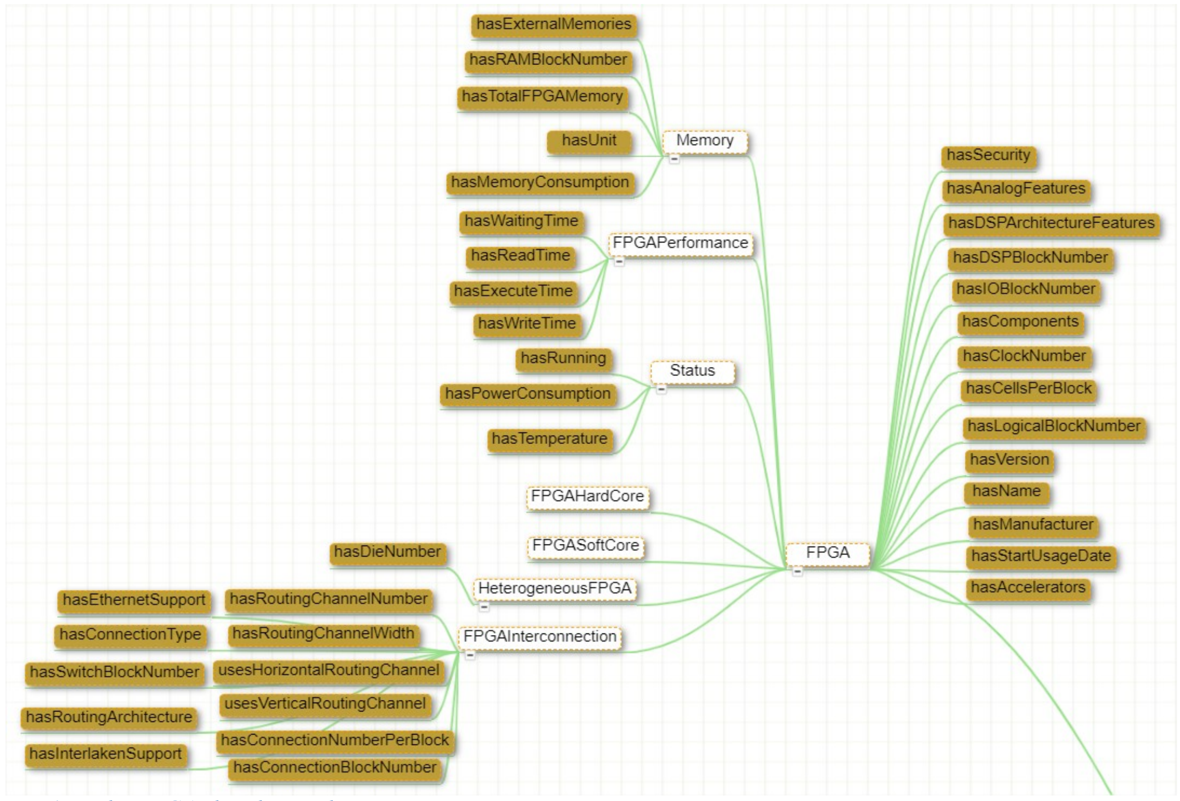


Fig. 17 - The FPGA class hierarchy

The complete class diagram for the Processing domain can be seen in Fig. 18.

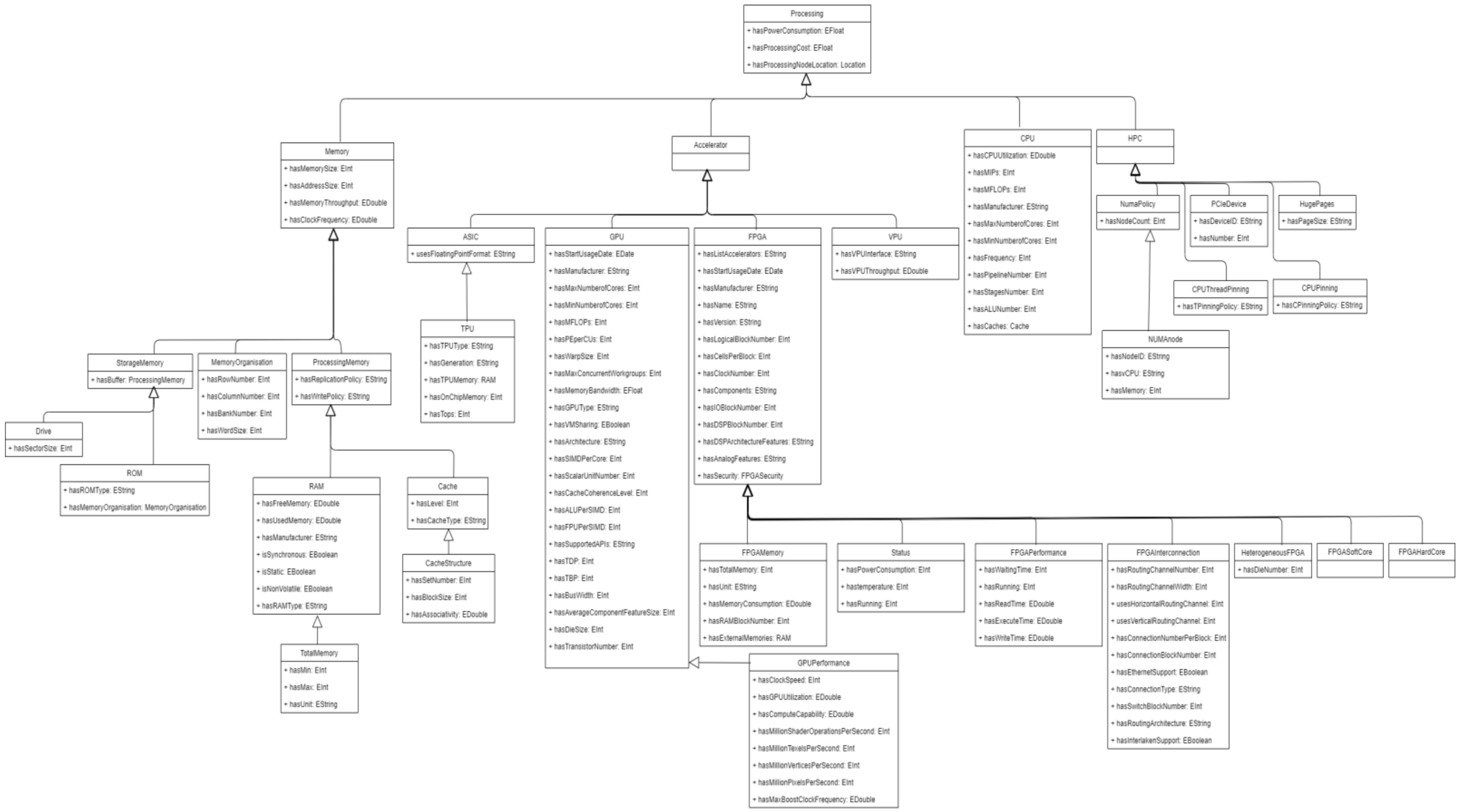

Processing
+ hasPowerConsumption: EFloat
+ hasProcessingCost: EFloat
+ hasProcessingNodeLocation: Location
Memory
+ hasMemorySize: EInt
+ hasAddressSize: EInt
+ hasMemoryThroughput: EDouble
+ hasClockFrequency: EDouble
Accelerator
CPU
+ hasCPUUtilization: EDouble
+ hasMIPs: EInt
+ hasMFLOPs: EInt
+ hasManufacturer: EString
+ hasMaxNumberofCores: EInt
+ hasMinNumberofCores: EInt
+ hasFrequency: EInt
+ hasPipelineNumber: EInt
+ hasStagesNumber: EInt
+ hasALUNumber: EInt
+ hasCaches: Cache
HPC
NumaPolicy
+ hasNodeCount: EInt
PCIeDevice
+ hasDeviceID: EString
+ hasNumber: EInt
HugePages
+ hasPageSize: EString
CPUThreadPinning
+ hasTPinningPolicy: EString
CPUPinning
+ hasCPinningPolicy: EString
NUMAnode
+ hasNodeID: EString
+ hasvCPU: EString
+ hasMemory: EInt
ASIC
+ usesFloatingPointFormat: EString
TPU
+ hasTPUType: EString
+ hasGeneration: EString
+ hasTPUMemory: RAM
+ hasOnChipMemory: EInt
+ hasTops: EInt
GPU
+ hasStartUsageDate: EDate
+ hasManufacturer: EString
+ hasMaxNumberofCores: EInt
+ hasMinNumberofCores: EInt
+ hasMFLOPs: EInt
+ hasPEperCUs: EInt
+ hasWarpSize: EInt
+ hasMaxConcurrentWorkgroups: EInt
+ hasMemoryBandwidth: EFloat
+ hasGPUType: EString
+ hasVMSharing: EBoolean
+ hasArchitecture: EString
+ hasSIMDPerCore: EInt
+ hasScalarUnitNumber: EInt
+ hasCacheCoherenceLevel: EInt
+ hasALUPerSIMD: EInt
+ hasFPUPerSIMD: EInt
+ hasSupportedAPIs: EString
+ hasTDP: EInt
+ hasTBP: EInt
+ hasBusWidth: EInt
+ hasAverageComponentFeatureSize: EInt
+ hasDieSize: EInt
+ hasTransistorNumber: EInt
FPGA
+ hasListAccelerators: EString
+ hasStartUsageDate: EDate
+ hasManufacturer: EString
+ hasName: EString
+ hasVersion: EString
+ hasLogicalBlockNumber: EInt
+ hasCellsPerBlock: EInt
+ hasClockNumber: EInt
+ hasComponents: EString
+ hasIOBlockNumber: EInt
+ hasDSPBlockNumber: EInt
+ hasDSPArchitectureFeatures: EString
+ hasAnalogFeatures: EString
+ hasSecurity: FPGASecurity
VPU
+ hasVPUInterface: EString
+ hasVPUThroughput: EDouble
StorageMemory
+ hasBuffer: ProcessingMemory
MemoryOrganisation
+ hasRowNumber: EInt
+ hasColumnNumber: EInt
+ hasBankNumber: EInt
+ hasWordSize: EInt
ProcessingMemory
+ hasReplicationPolicy: EString
+ hasWritePolicy: EString
Drive
+ hasSectorSize: EInt
ROM
+ hasROMType: EString
+ hasMemoryOrganisation: MemoryOrganisation
RAM
+ hasFreeMemory: EDouble
+ hasUsedMemory: EDouble
+ hasManufacturer: EString
+ isSynchronous: EBoolean
+ isStatic: EBoolean
+ isNonVolatile: EBoolean
+ hasRAMType: EString
Cache
+ hasLevel: EInt
+ hasCacheType: EString
CacheStructure
+ hasSetNumber: EInt
+ hasBlockSize: EInt
+ hasAssociativity: EDouble
TotalMemory
+ hasMin: EInt
+ hasMax: EInt
+ hasUnit: EString
FPGAMemory
+ hasTotalMemory: EInt
+ hasUnit: EString
+ hasMemoryConsumption: EDouble
+ hasRAMBlockNumber: EInt
+ hasExternalMemories: RAM
Status
+ hasPowerConsumption: EInt
+ hastemperature: EInt
+ hasRunning: EInt
FPGAPerformance
+ hasWaitingTime: EInt
+ hasRunning: EInt
+ hasReadTime: EDouble
+ hasExecuteTime: EDouble
+ hasWriteTime: EDouble
FPGAInterconnection
+ hasRoutingChannelNumber: EInt
+ hasRoutingChannelWidth: EInt
+ usesHorizontalRoutingChannel: EInt
+ usesVerticalRoutingChannel: EInt
+ hasConnectionNumberPerBlock: EInt
+ hasConnectionBlockNumber: EInt
+ hasEthernetSupport: EBoolean
+ hasConnectionType: EString
+ hasSwitchBlockNumber: EInt
+ hasRoutingArchitecture: EString
+ hasInterlakenSupport: EBoolean
HeterogeneousFPGA
+ hasDieNumber: EInt
FPGASoftCore
FPGAHardCore
GPUPerformance
+ hasClockSpeed: EInt
+ hasGPUUtilization: EDouble
+ hasComputeCapability: EDouble
+ hasMillionShaderOperationsPerSecond: EInt
+ hasMillionTexelsPerSecond: EInt
+ hasMillionVerticesPerSecond: EInt
+ hasMillionPixelsPerSecond: EInt
+ hasMaxBoostClockFrequency: EDouble

Fig. 18 - The UML class diagram for the Processing domain

4.1.2 Network

Another important subclass on the *IaaS* class that has been introduced in the MDS is the *NetworkEntity* to represent any kind of network entity that could be included in a specific cloud. Several properties and subclasses have been introduced to cover all the network-related elements that can be considered for managing applications hosted over multi-clouds and fog computing environments. The main subclasses of the *NetworkEntity* class are the following:

- *Network* (see Fig. 19) which refers to the network related aspects that bind the operation and communication capabilities of an offered or a requested (Fog) IaaS resource.
- *NetworkQoS* (see Fig. 19) that represents the main aspects of the network quality of service.
- *SoftwareNetworkEntity* (see Fig. 19) that represents a software-based network entity.
- *HardwareNetworkEntity* (see Fig. 20) that refers to all the aspects of hardware-based network entities that may constitute network nodes serving the cloud application components.

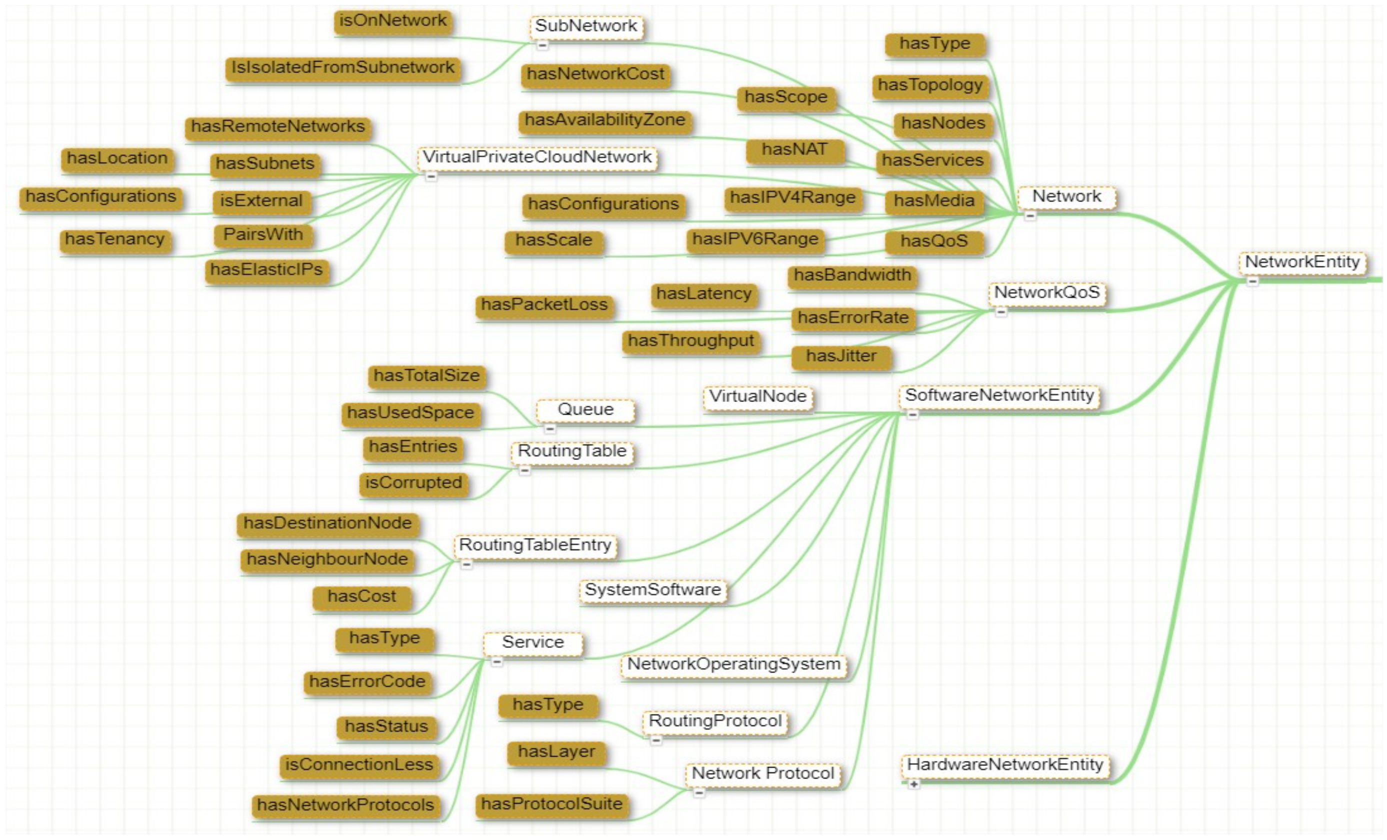


Fig. 19 - Network Entity high-level as well as Network, NetworkQoS and SoftwareNetworkEntity classes

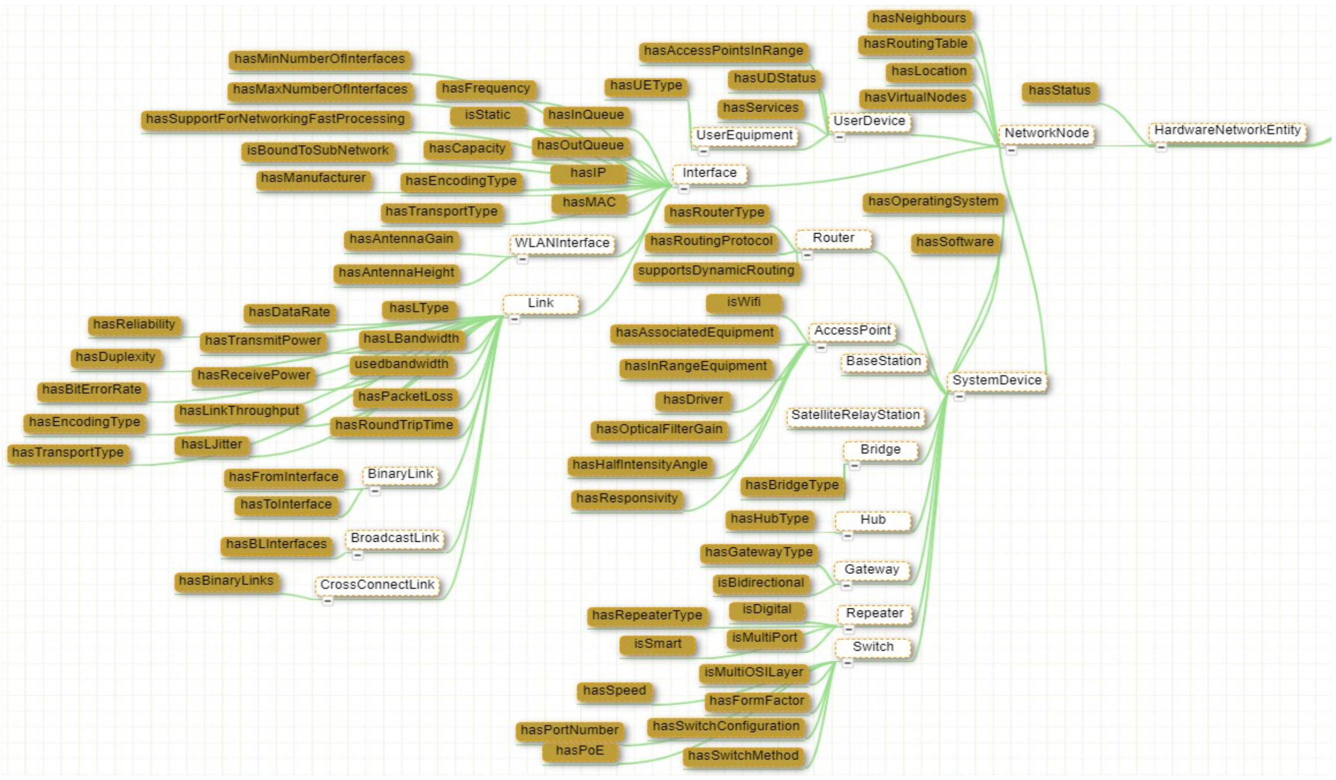


Fig. 20 - The hierarchy of the HardwareNetworkEntity class

In Fig. 21, we provide all the details regarding the extensions of the *NetworkEntity* class in a UML class diagram.

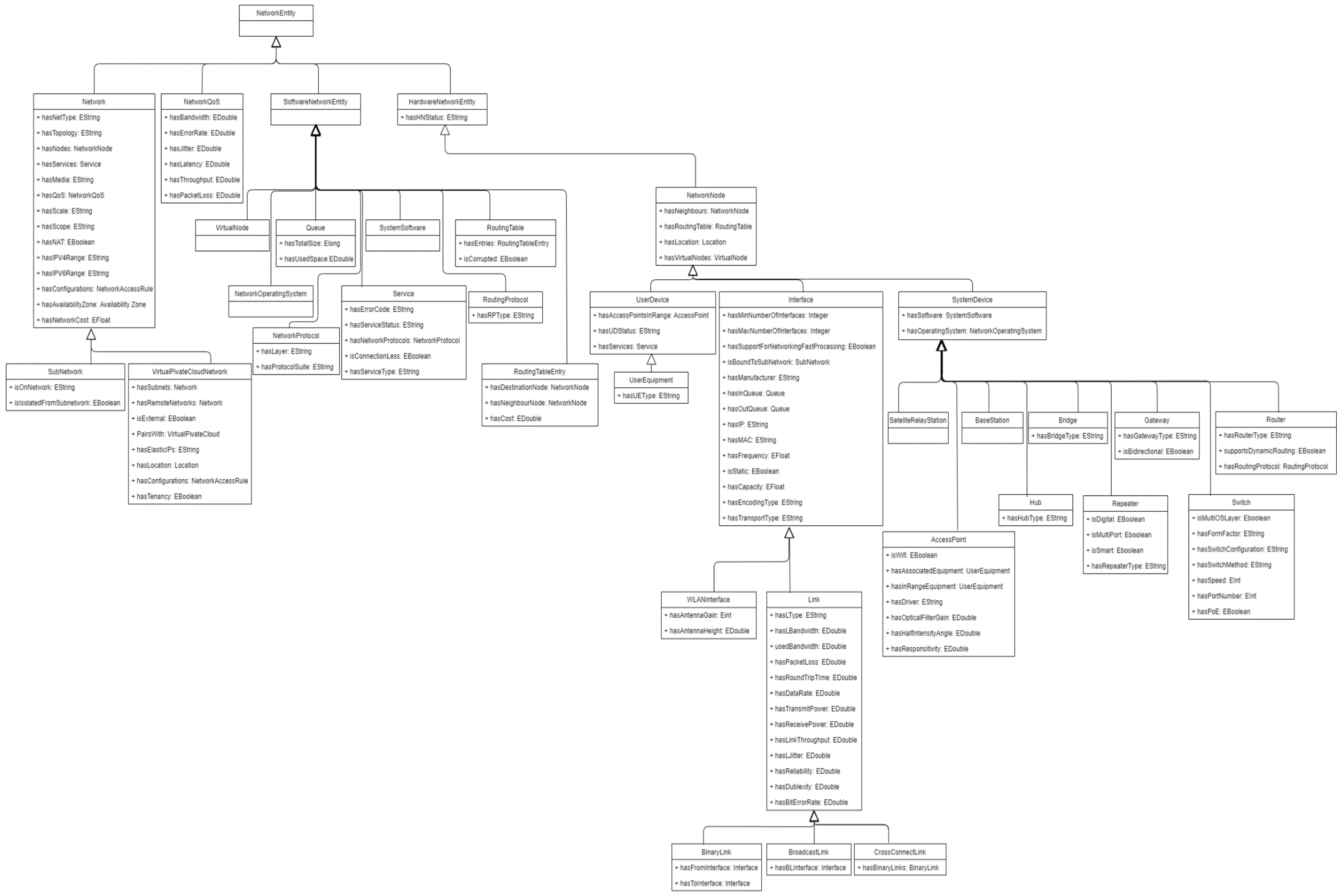


Fig. 21 - The UML class diagram for the network domain

4.1.3 PaaS

Another important class of the *Application Placement Model* is the *PaaS* class which encapsulates all the attributes related to platform level cloud resources that are required and offered for deploying Morphemic-enabled applications. The main addition refers to the new *Serverless* class that represents all the aspects of serverless platform-as-a-service, i.e., a PaaS that enables the deployment of functions/serverless components in the cloud. The main subclasses of the *Serverless* class are the following:

- *ServerlessPlatform* (see Fig. 22) which represents a specific serverless platform. Underneath this class, all major serverless platforms can be found as instances of the *ServerlessPlatform* class.
- *ServerlessFramework* (see Fig. 22) which represents a framework that enables the construction and deployment of serverless functions
- *EventType* (see Fig. 22) which represents the types of events that can trigger the execution of functions.
- *Composition* (see Fig. 22) which represents the technology of a serverless platform to support the composition of functions.
- *Cost* (see Fig. 22) which represents the cost model of a serverless PaaS.
- *FreeQuota* (see Fig. 22) which refers to the free quota per month that is associated with an account in the serverless PaaS.
- *Limits* (see Fig. 22) which represents the set of limitations that are associated with a serverless PaaS.

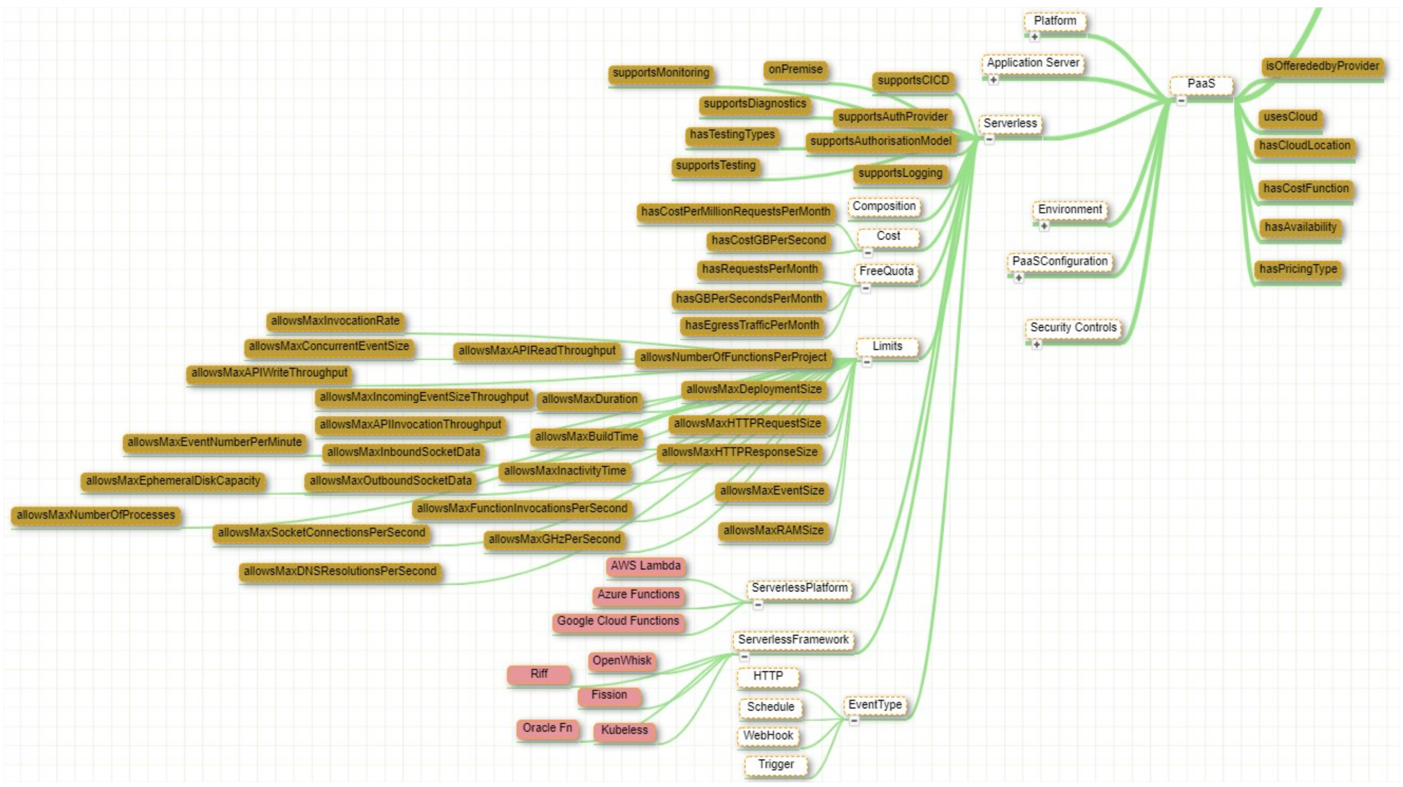


Fig. 22 - PaaS high-level and Serverless classes

In Fig. 23, we provide all the details and extensions of the *Serverless* class in a UML class diagram.

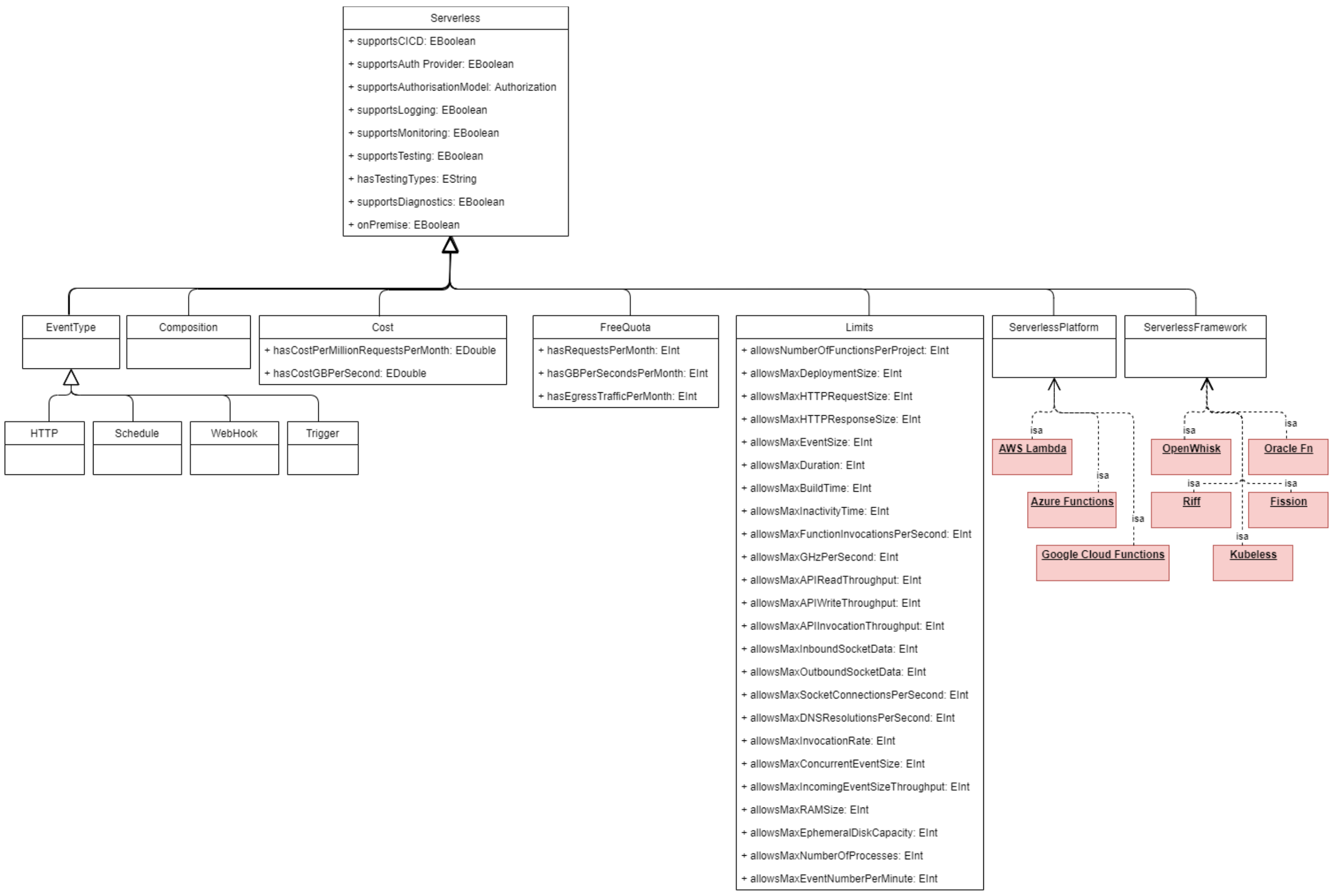

Serverless
+ supportsCICD: EBoolean
+ supportsAuth Provider: EBoolean
+ supportsAuthorisationModel: Authorization
+ supportsLogging: EBoolean
+ supportsMonitoring: EBoolean
+ supportsTesting: EBoolean
+ hasTestingTypes: EString
+ supportsDiagnostics: EBoolean
+ onPremise: EBoolean
EventType
Composition
Cost
+ hasCostPerMillionRequestsPerMonth: EDouble
+ hasCostGBPerSecond: EDouble
FreeQuota
+ hasRequestsPerMonth: EInt
+ hasGBPerSecondsPerMonth: EInt
+ hasEgressTrafficPerMonth: EInt
Limits
+ allowsNumberOfFunctionsPerProject: EInt
+ allowsMaxDeploymentSize: EInt
+ allowsMaxHTTPRequestSize: EInt
+ allowsMaxHTTPResponseSize: EInt
+ allowsMaxEventSize: EInt
+ allowsMaxDuration: EInt
+ allowsMaxBuildTime: EInt
+ allowsMaxInactivityTime: EInt
+ allowsMaxFunctionInvocationsPerSecond: EInt
+ allowsMaxGHzPerSecond: EInt
+ allowsMaxAPIReadThroughput: EInt
+ allowsMaxAPIWriteThroughput: EInt
+ allowsMaxAPIInvocationThroughput: EInt
+ allowsMaxInboundSocketData: EInt
+ allowsMaxOutboundSocketData: EInt
+ allowsMaxSocketConnectionsPerSecond: EInt
+ allowsMaxDNSResolutionsPerSecond: EInt
+ allowsMaxInvocationRate: EInt
+ allowsMaxConcurrentEventSize: EInt
+ allowsMaxIncomingEventSizeThroughput: EInt
+ allowsMaxRAMSize: EInt
+ allowsMaxEphemeralDiskCapacity: EInt
+ allowsMaxNumberOfProcesses: EInt
+ allowsMaxEventNumberPerMinute: EInt
ServerlessPlatform
ServerlessFramework
HTTP
Schedule
WebHook
Trigger
isa
AWS Lambda
Azure Functions
Google Cloud Functions
OpenWhisk
Oracle Fn
Riff
Fission
Kubeless

Fig. 23 - UML class diagram showing the serverless class hierarchy

The extensions in the *PaaS* class also involved significant extensions in the *Security Controls* class. Specifically, the subclasses *SecurityConfiguration* and HardwareBasedSecurity were introduced (see Fig. 24). The first refers to all the aspects of configurations over a network entity that is specified through a set of network access rules (defined as a separate subclass called *NetworkAccessRule*). The latter aggregates all the security capabilities that can be offered as a service through dedicated hardware components. The *HardwareBasedSecurity* involves the following subclasses:

- *FPGASecurity* (see Fig. 24) which refers to the security capabilities implanted in a FPGA hardware.
- *SecureEnclave* (see Fig. 24) which refers to the capability of a trusted execution environment based on dedicated microkernels that support isolation and encryption.

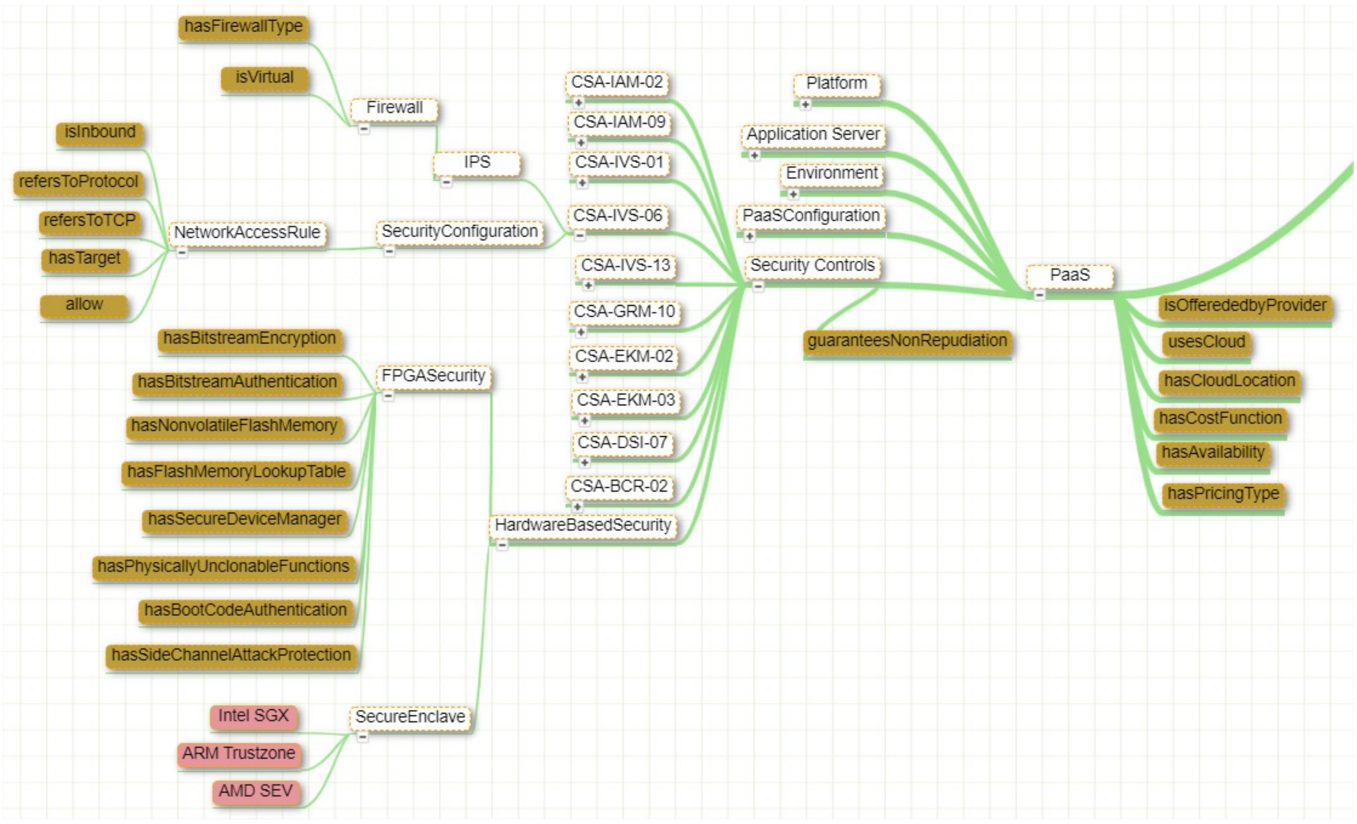


Fig. 24 - PaaS high-level as well as SecurityConfiguration and HardwareBasedSecurity classes

In Fig. 25, we provide the extensions regarding the *SecurityControls* class in a UML class diagram.

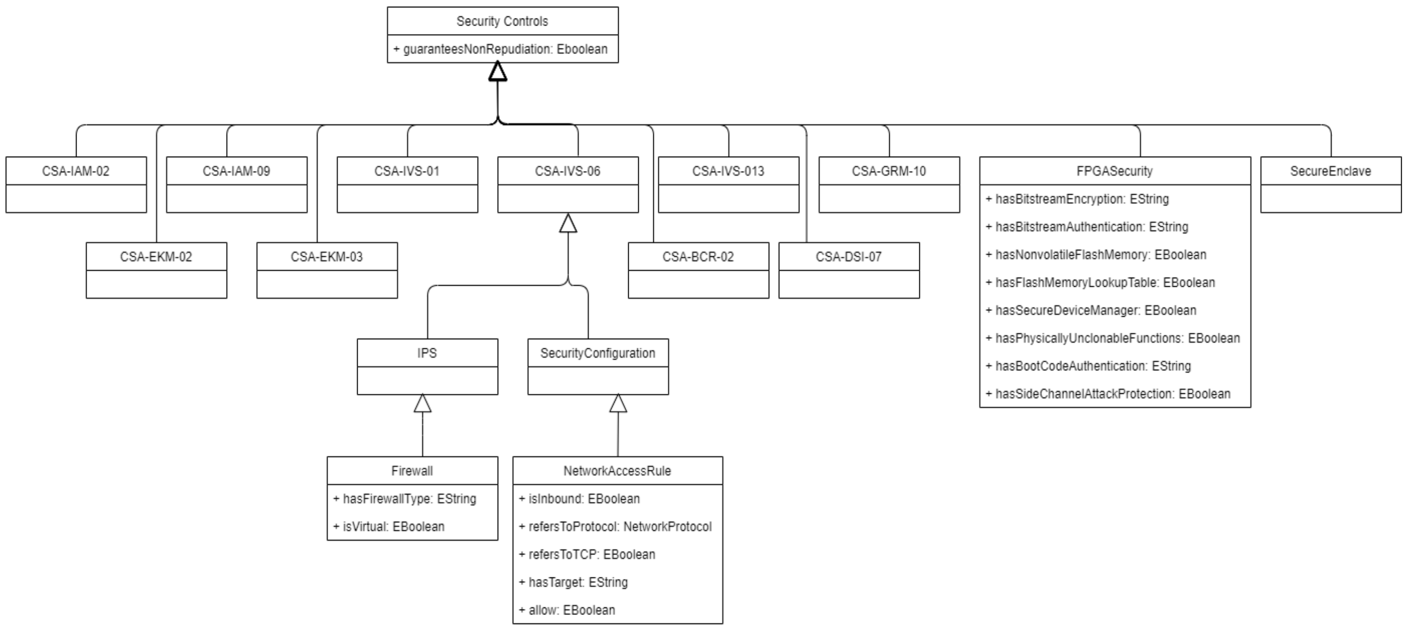

Security Controls
+ guaranteesNonRepudiation: Eboolean
CSA-IAM-02
CSA-IAM-09
CSA-IVS-01
CSA-IVS-06
CSA-IVS-013
CSA-GRM-10
CSA-EKM-02
CSA-EKM-03
CSA-BCR-02
CSA-DSI-07
FPGASecurity
+ hasBitstreamEncryption: EString
+ hasBitstreamAuthentication: EString
+ hasNonvolatileFlashMemory: EBoolean
+ hasFlashMemoryLookupTable: EBoolean
+ hasSecureDeviceManager: EBoolean
+ hasPhysicallyUnclonableFunctions: EBoolean
+ hasBootCodeAuthentication: EString
+ hasSideChannelAttackProtection: EBoolean
SecureEnclave
IPS
SecurityConfiguration
Firewall
+ hasFirewallType: EString
+ isVirtual: EBoolean
NetworkAccessRule
+ isInbound: EBoolean
+ refersToProtocol: NetworkProtocol
+ refersToTCP: EBoolean
+ hasTarget: EString
+ allow: EBoolean

Fig. 25 - UML class diagram showing the Security Control class hierarchy

4.1.4 Big Data

Also, a number of important extensions were introduced in the Big-Data Model part of the MDS. Specifically, in the *Big Data Aspects* a number of instances were appended under the *Format* and *Type* subclasses of the *Data Variety* class. *Data Variety* refers to the different types of data that should be processed by a Morphemic-enabled cloud application, stating an increased diversity of data that should be stored, processed or combined. The abstract view of this part of MDS is presented in Fig. 26 while the fine-grained details are presented in the UML class diagram of Fig. 27.

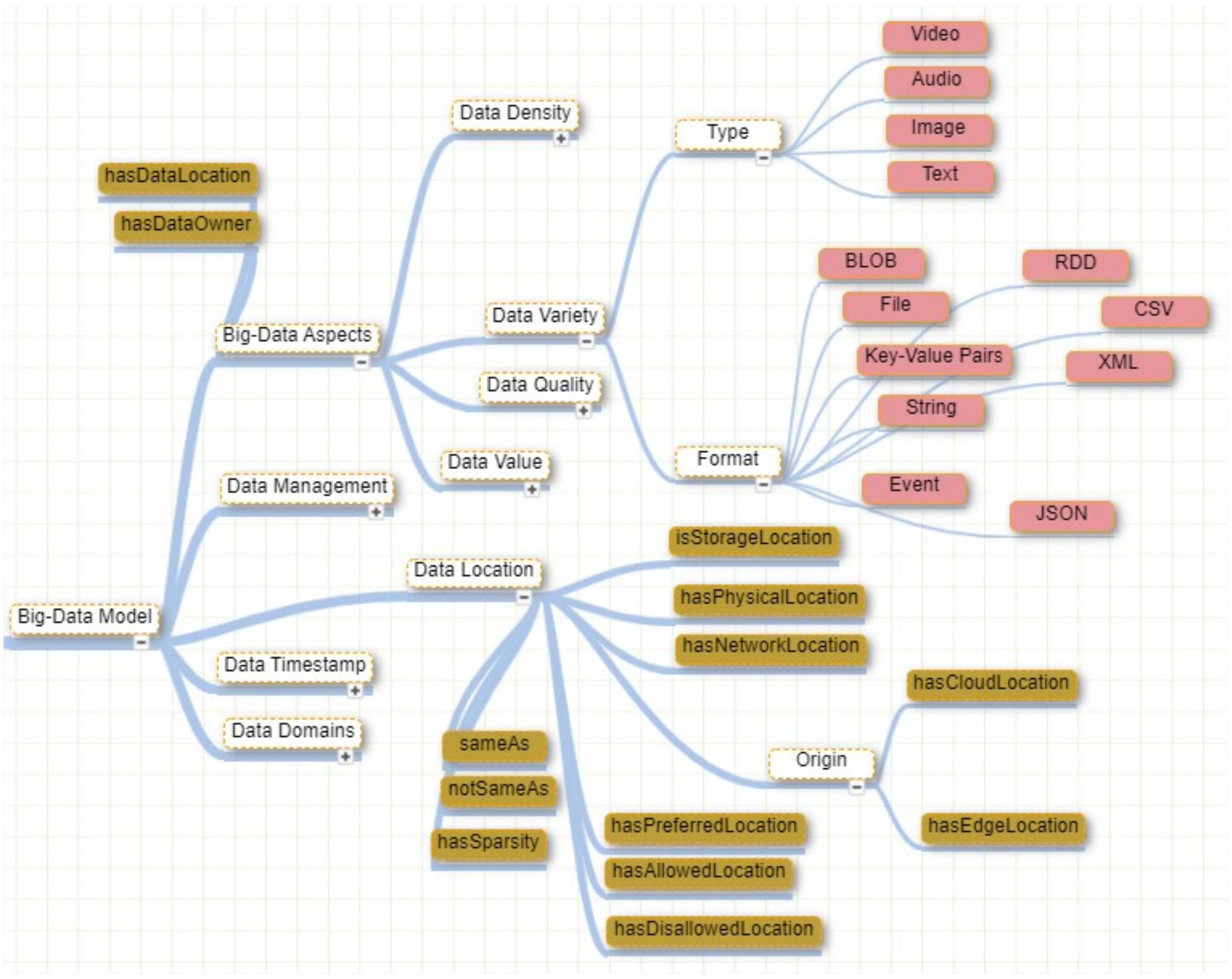


Fig. 26 - Big-Data Model high-level as well as DataVariety and Data Location classes

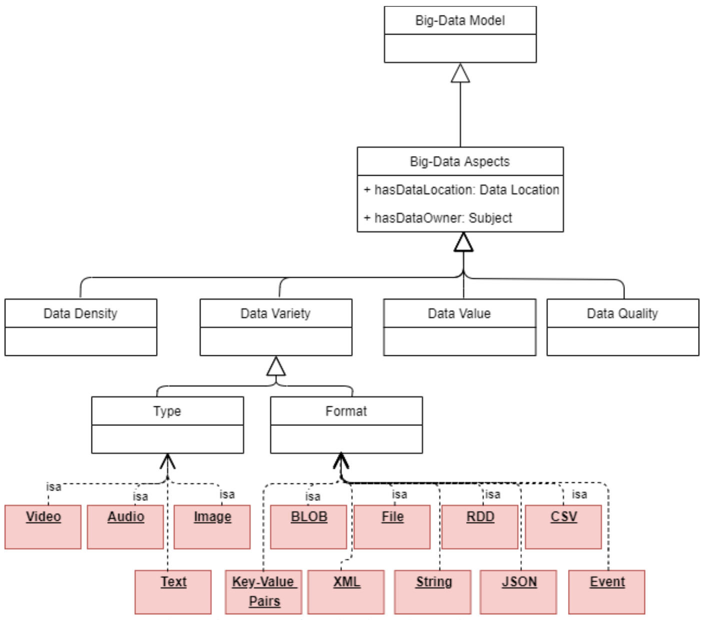


Fig. 27 - UML class diagram for the big data domain

Besides, smaller additions in the form of new properties were performed in the *Data Location* class. The details of these extensions are available in the UML diagram of Fig. 28.

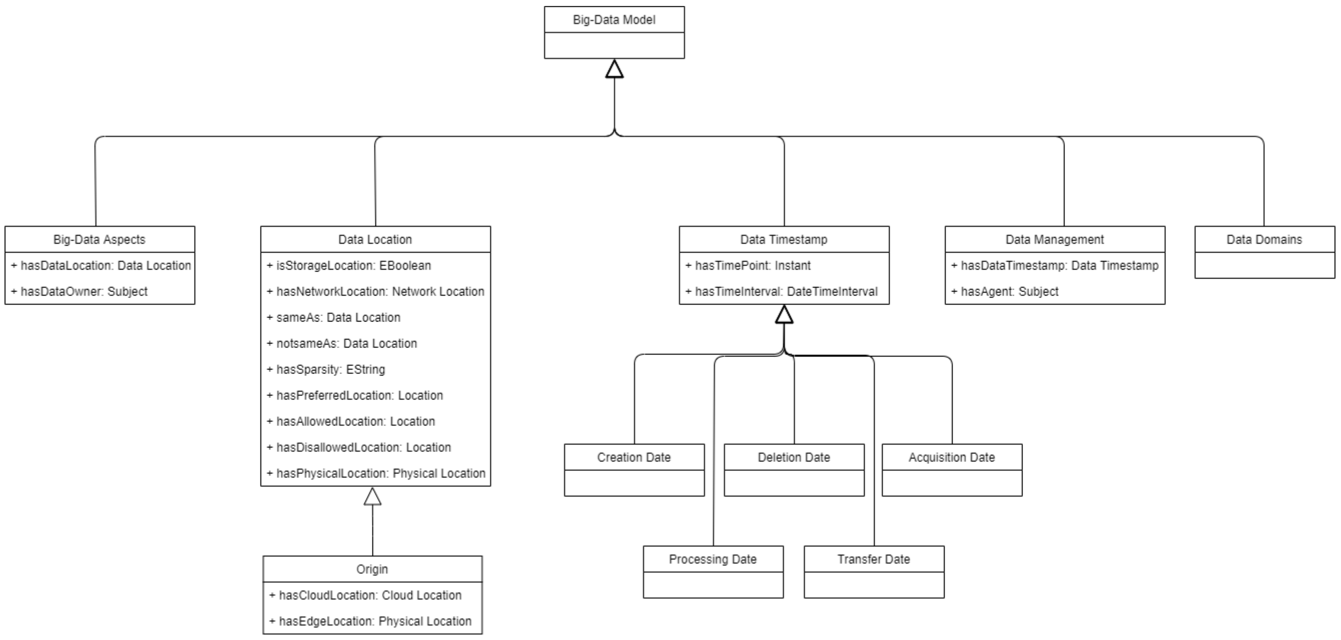

Big-Data Model
Big-Data Aspects
+ hasDataLocation: Data Location
+ hasDataOwner: Subject
Data Location
+ isStorageLocation: EBoolean
+ hasNetworkLocation: Network Location
+ sameAs: Data Location
+ notsameAs: Data Location
+ hasSparsity: EString
+ hasPreferredLocation: Location
+ hasAllowedLocation: Location
+ hasDisallowedLocation: Location
+ hasPhysicalLocation: Physical Location
Origin
+ hasCloudLocation: Cloud Location
+ hasEdgeLocation: Physical Location
Data Timestamp
+ hasTimePoint: Instant
+ hasTimeInterval: DateTimeInterval
Data Management
+ hasDataTimestamp: Data Timestamp
+ hasAgent: Subject
Data Domains
Creation Date
Deletion Date
Acquisition Date
Processing Date
Transfer Date

Fig. 28 - UML class diagram covering the location class

Furthermore, extensions were provided in the *Data Management* class of the *Big-Data Model,* which encapsulates all the relevant concepts that can be used in order to describe major technological choices with respect to how big data is acquired, stored, processed, transferred or replicated for redundancy reasons. The extensions involve both new subclasses and properties. In Fig. 29, the new high-level view of the *Data Management* class is provided while Fig. 30 presents the *Data Storage* class and its new view.

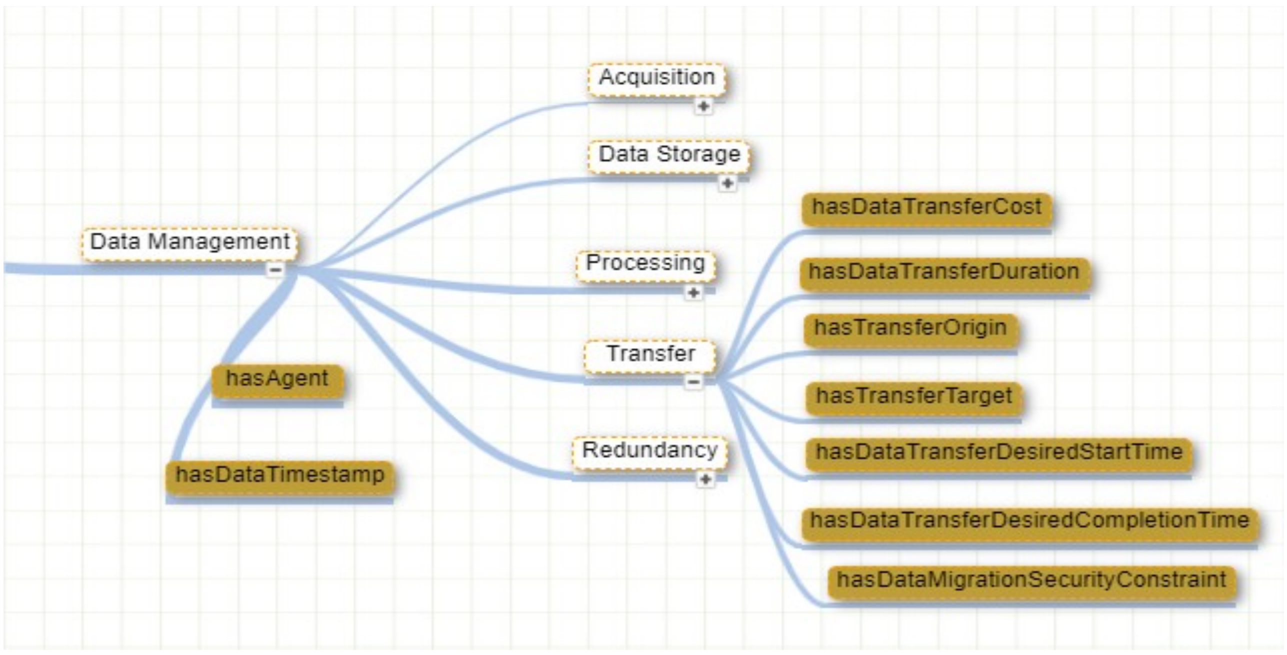


Fig. 29 - Data Management high-level and Transfer classes

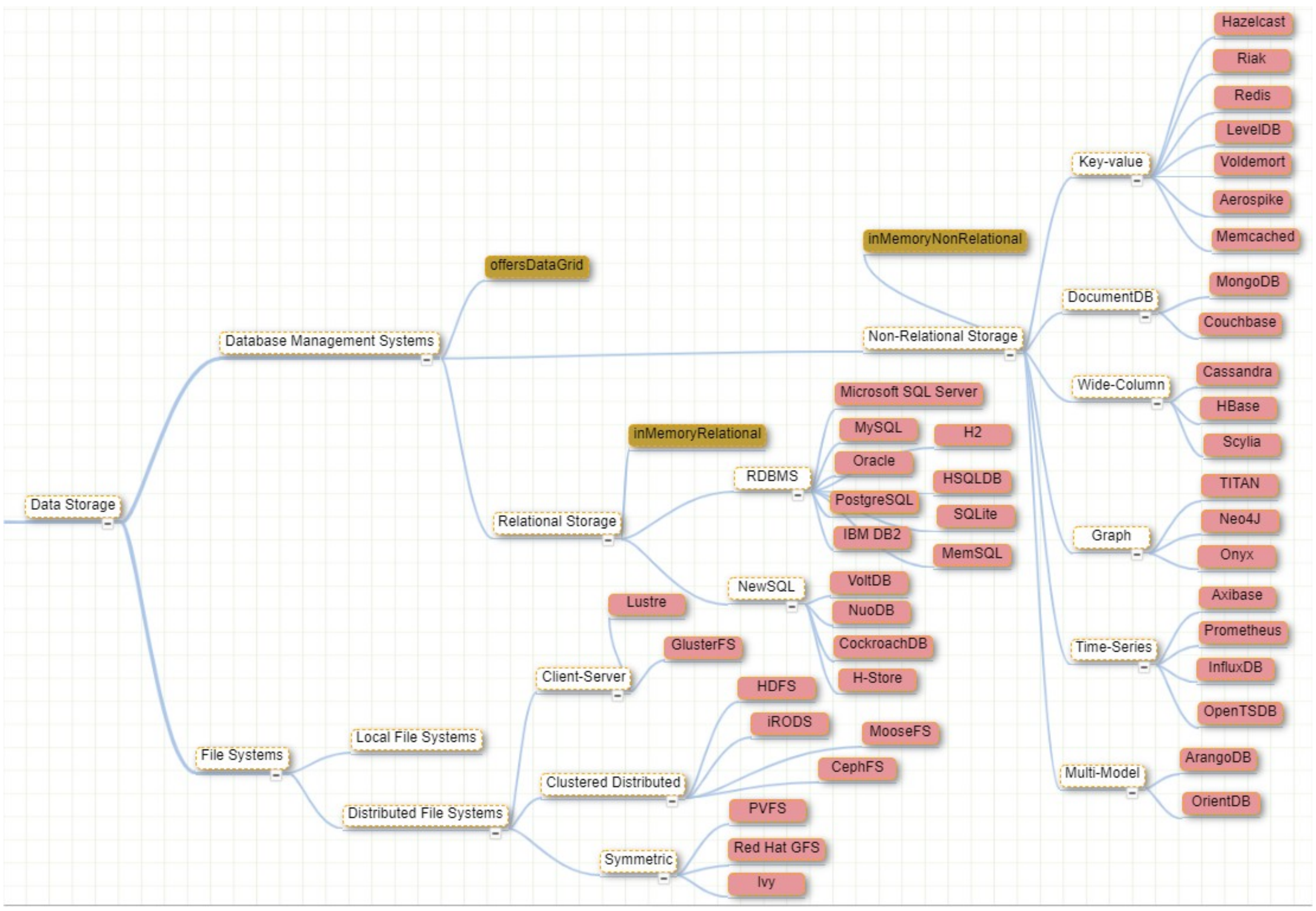


Fig. 30 - The Data Storage class hierarchy

In Fig. 31 and Fig. 32 the fine-grained details of the *Data Management* and *Data Storage* classes are presented, respectively.

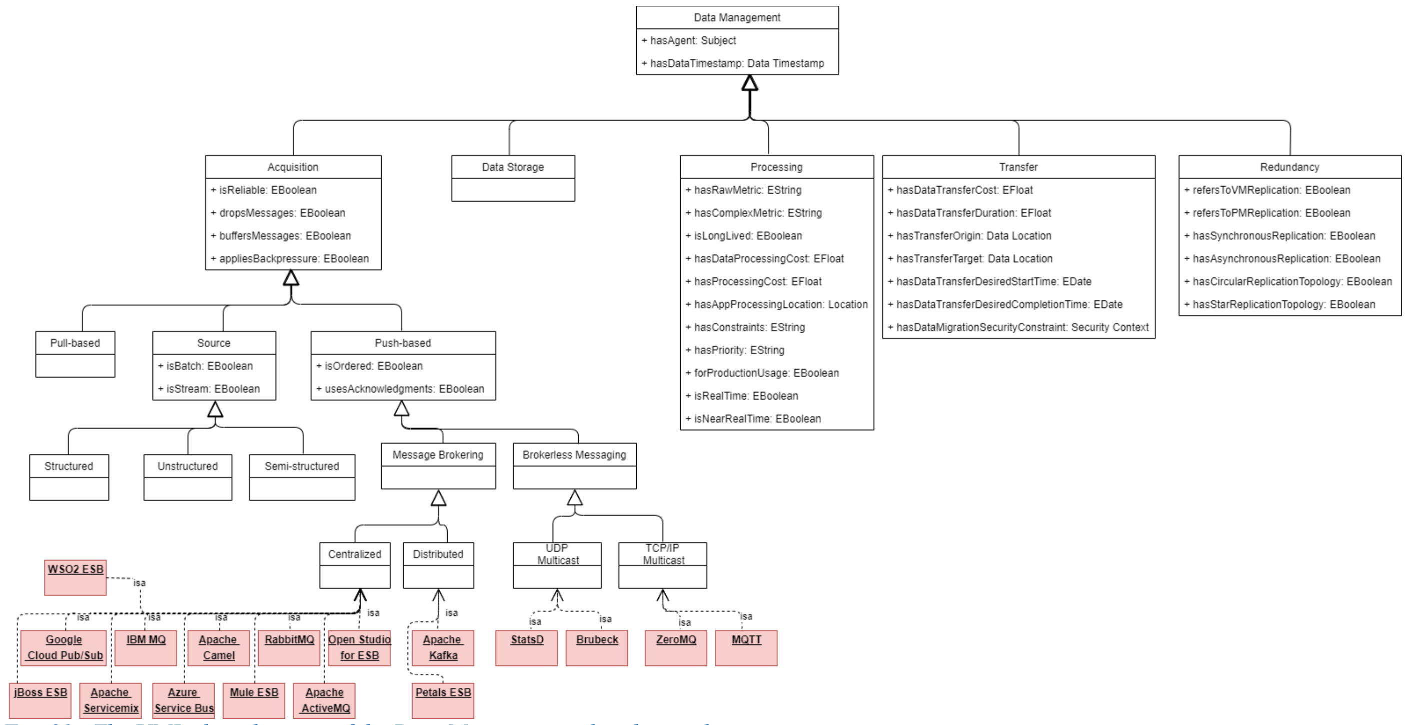

Data Management
+ hasAgent: Subject
+ hasDataTimestamp: Data Timestamp
Acquisition
+ isReliable: EBoolean
+ dropsMessages: EBoolean
+ buffersMessages: EBoolean
+ appliesBackpressure: EBoolean
Data Storage
Processing
+ hasRawMetric: EString
+ hasComplexMetric: EString
+ isLongLived: EBoolean
+ hasDataProcessingCost: EFloat
+ hasProcessingCost: EFloat
+ hasAppProcessingLocation: Location
+ hasConstraints: EString
+ hasPriority: EString
+ forProductionUsage: EBoolean
+ isRealTime: EBoolean
+ isNearRealTime: EBoolean
Transfer
+ hasDataTransferCost: EFloat
+ hasDataTransferDuration: EFloat
+ hasTransferOrigin: Data Location
+ hasTransferTarget: Data Location
+ hasDataTransferDesiredStartTime: EDate
+ hasDataTransferDesiredCompletionTime: EDate
+ hasDataMigrationSecurityConstraint: Security Context
Redundancy
+ refersToVMReplication: EBoolean
+ refersToPMReplication: EBoolean
+ hasSynchronousReplication: EBoolean
+ hasAsynchronousReplication: EBoolean
+ hasCircularReplicationTopology: EBoolean
+ hasStarReplicationTopology: EBoolean
Pull-based
Source
+ isBatch: EBoolean
+ isStream: EBoolean
Push-based
+ isOrdered: EBoolean
+ usesAcknowledgments: EBoolean
Structured
Unstructured
Semi-structured
Message Brokering
Brokerless Messaging
Centralized
Distributed
UDP Multicast
TCP/IP Multicast
WSO2 ESB
isa
Google Cloud Pub/Sub
IBM MQ
Apache Camel
RabbitMQ
Open Studio for ESB
Apache Kafka
StatsD
Brubeck
ZeroMQ
MQTT
JBoss ESB
Apache Servicemix
Azure Service Bus
Mule ESB
Apache ActiveMQ
Petals ESB

Fig. 31 - The UML class diagram of the Data Management class hierarchy

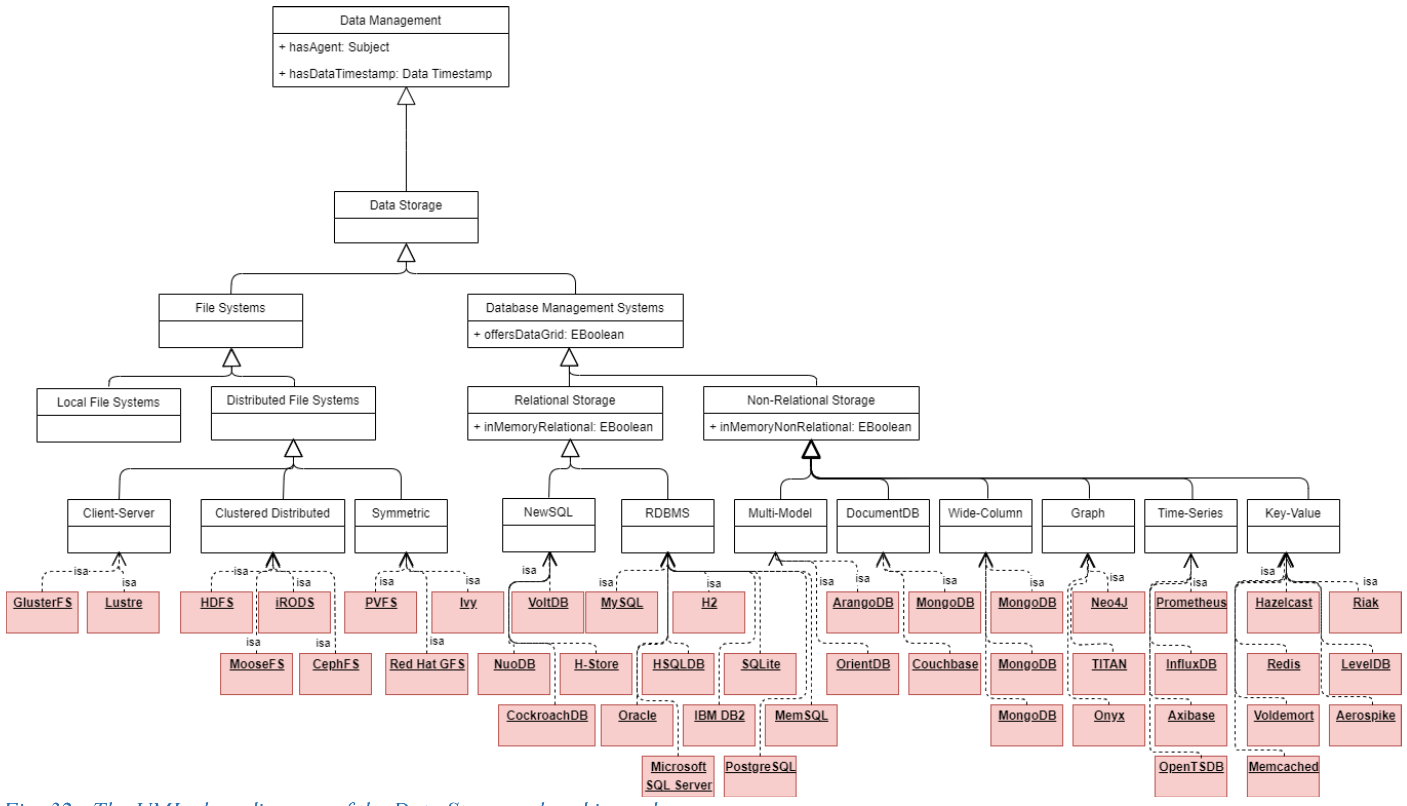


Fig. 32 - The UML class diagram of the Data Storage class hierarchy

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 [18] 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[10] 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[11]. MDS was also serialized in XMI[12]. 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 CAMEL models. We note that the reader may find the serialization of the complete model here[13]. 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 here[14].

5 Use-Case Modelling

The Morphemic project involves three use-case partners who plan to exploit the Morphemic platform in order to support the polymorphic modelling and adaptive provisioning of their applications. One of these partners is IS-Wireless, which has the ambition to deploy and adaptively provision its use-case application, exploiting 5G Software-defined Radio Access Networks (RAN), in cloud and hybrid (cloud & edge) environments. This use-case application has created particular requirements on both CAMEL and MDS in terms of relevant extensions so it is a suitable candidate for demonstration purposes. Concerning CAMEL and especially CAMEL 3.0 it has led to creating the notion of communication requirement, while it has guided the extension of MDS concerning the network domain and the introduction of relevant network QoS attributes.
This use-case application is still in development and will incorporate relevant scenarios that showcase the new features to be exhibited by the Morphemic platform. As such, to demonstrate CAMEL 3.0 in this deliverable, we have taken the basic scenario of cloud deployment with no optimisation objectives. This scenario already involves all the relevant application components as well as most of the requirements concerning resources, location, communication QoS and service level objectives (SLOs). In this respective, it is a very good candidate in order to showcase some CAMEL extensions like the one related to the modelling of communication requirements.
A RAN system comprises various protocols like the MAC (Medium Access Control) one. Many of these protocols can be actually virtualised and situated in different locations. A Cloud RAN can comprise three main units on which the different protocols are distributed: the Radio Unit (RU) comprising low-level protocols, the Distributed Unit (DU) comprising intermediate-level protocols and the Central Unit (CU) comprising high-level protocols. The first unit, the RU one is tightly connected to a physical location (actually a cell site), while the actual location of the other units can vary as they can be virtualized in the form of virtual network functions (VNF). As such, the other units are amenable for management by the Morphemic platform. The CU unit can be also separated into the control (CP) and user plane (UP) such that the respective parts, i.e., CU-CP and CU-UP can be independently managed and deployed.
For the static deployment scenario, various kinds of requirements have been considered, some of which can be considered as generic while others communication-specific. The generic requirements are the following:

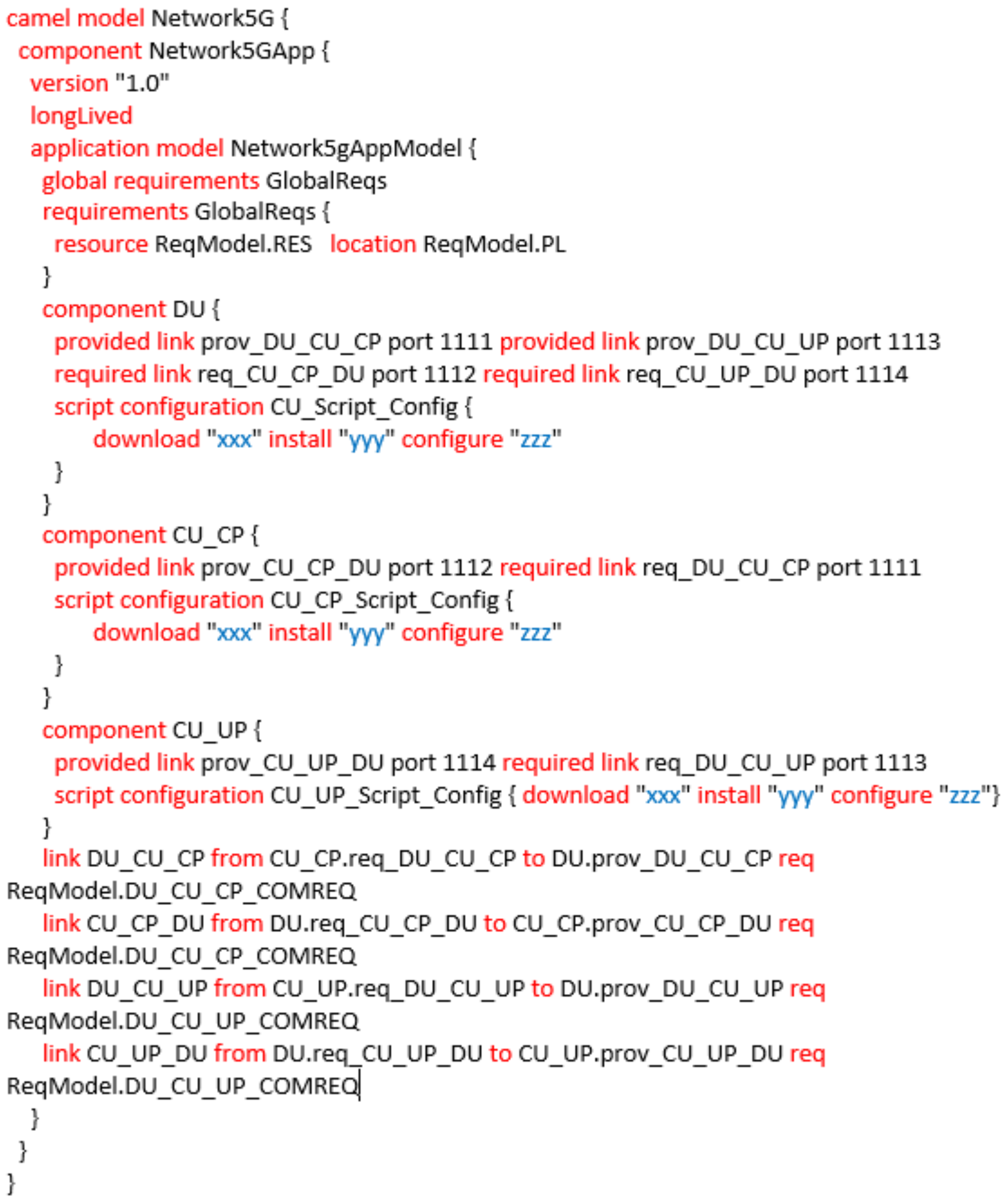
- the location of all components should be determined on a regional/country level granularity (e.g. deployment should be done in Poland)
- each component has the requirements of 5 as minimum number of cores and 2 as the number of GBs for the RAM
- the application average availability should be at least 99.999%

The communication-specific requirements concern the quality of the communication between pairs of components. These are the following:

- The latency between DU and CU-CP should be at most 5 (milliseconds) while the throughput at least 0.1 Gbps. This holds for both directions of communication
- The latency between DU and CU-UP should be at most 1 (millisecond) while the throughput at least 4 Gbps. Again, this holds for both communication directions

[10] https://app.mindmapmaker.org/
[11] https://melodic.cloud/UuTf-KRW.png
[12] http://www.omg.org/spec/XMI/
[13] https://gitlab.ow2.org/melodic/camel/-/tree/rc3.1/metadata-schema/current
[14] https://bitbucket.7bulls.eu/projects/MEL/repos/metadata-schema/browse/muse

Based on the above requirements and the main structure of the application that will be visible to a Morphemic platform instance, the respective CAMEL 3.0 model of the RAN application has been developed and will be presented in a step-wise manner by explicating the main content per each domain of CAMEL covered. The content will be demonstrated by considering the textual syntax of CAMEL.
The first snippet of this model is shown in the following figure where the focus in on the application domain. As it can be seen, the CAMEL model includes a single component that represents the whole application. This component has a specific version and is long-lived. It is realised via an application model that involves three main components and their communication between them. Each of the components is mainly deployed through a script-based configuration, the details of which are not shown as it is considered as private information about the use-case that cannot be published. In addition, each application component has one required hosting port as well as 2 provided and 2 required link/communication ports. In the same model, there is a definition of a global set of requirements (i.e., this set holds for all application components) which includes references to the generic requirements about the component location and the demanded amount of resources. The actual requirements are specified in the requirement model of this application. Finally, a set of four communications with their quality requirements are described that connect the required and provided communication ports of the application components so as to cover the communication between DU and CU-CP as well as DU and CU-UP in both directions.



```
camel model Network5G {
 component Network5GApp {
  version "1.0"
  longLived
  application model Network5gAppModel {
   global requirements GlobalReqs
   requirements GlobalReqs {
    resource ReqModel.RES  location ReqModel.PL
   }
   component DU {
    provided link prov_DU_CU_CP port 1111 provided link prov_DU_CU_UP port 1113
    required link req_CU_CP_DU port 1112 required link req_CU_UP_DU port 1114
    script configuration CU_Script_Config {
        download "xxx" install "yyy" configure "zzz"
    }
   }
   component CU_CP {
    provided link prov_CU_CP_DU port 1112 required link req_DU_CU_CP port 1111
    script configuration CU_CP_Script_Config {
        download "xxx" install "yyy" configure "zzz"
    }
   }
   component CU_UP {
    provided link prov_CU_UP_DU port 1114 required link req_DU_CU_UP port 1113
    script configuration CU_UP_Script_Config { download "xxx" install "yyy" configure "zzz"}
   }
   link DU_CU_CP from CU_CP.req_DU_CU_CP to DU.prov_DU_CU_CP req
ReqModel.DU_CU_CP_COMREQ
   link CU_CP_DU from DU.req_CU_CP_DU to CU_CP.prov_CU_CP_DU req
ReqModel.DU_CU_CP_COMREQ
   link DU_CU_UP from CU_UP.req_DU_CU_UP to DU.prov_DU_CU_UP req
ReqModel.DU_CU_UP_COMREQ
   link CU_UP_DU from DU.req_CU_UP_DU to CU_UP.prov_CU_UP_DU req
ReqModel.DU_CU_UP_COMREQ
  }
 }
}
```

Fig. 33 - Snippet of the application's CAMEL model focusing on the deployment domain

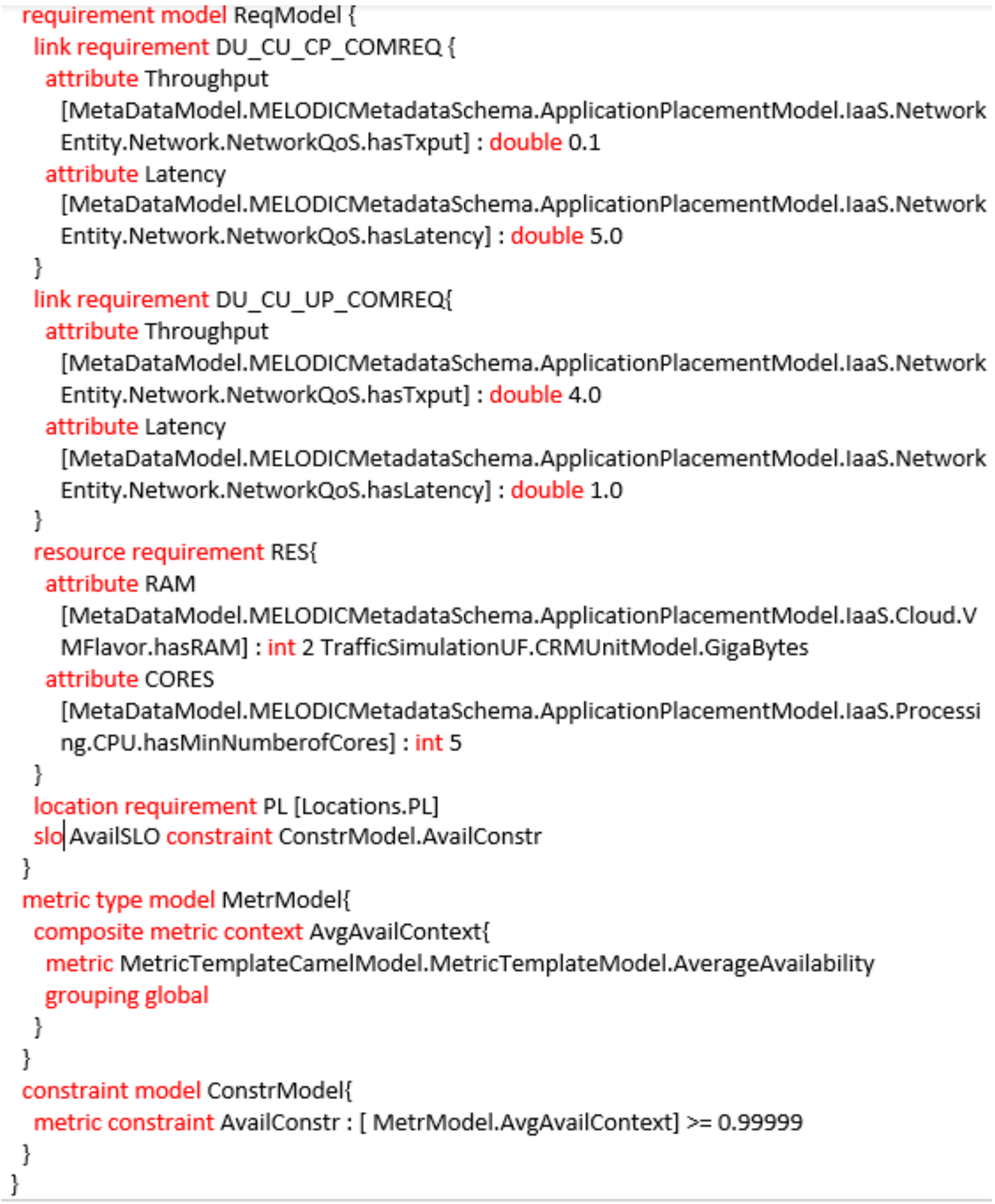
The next snippet shows the requirement, metric type and constraint models of the RAN application. The requirement model includes two link requirements covering the quality of communication between DU and CU-CP as well as between DU and CU-UP. Each communication requirement involves the specification of two attributes that define the respective constraints on communication latency and throughput. As it can be easily seen, both attributes are annotated via the use of the MDS to point to the right network QoS data properties.

The requirement model also involves one resource requirement that is to be applied to all application components. This resource requirement covers the specification of constraints on two attributes, annotated with MDS relevant data properties, which are the RAM and minimum number of cores ones.

The requirement model ends with the specification of a location and SLO requirement. The location requirement explicates the actual location of the application components, which is Poland, while the SLO requirement is associated with a specific constraint on average application availability. Please note that the location of Poland is drawn from a location model which involves a physical location hierarchy of three levels: continent, sub-continent and countries. This location model can be put in the respective Eclipse environment of the CAMEL textual editor in order to be re-used in the context of application modelling in CAMEL.

Similarly, the metric type model re-uses already specified CAMEL elements, in particular availability-based metrics, from a metric (template) model. As such, it only involves the specification of a composite metric context which explicates the actual metric to be computed (average application availability) as well as the way the measurements of the downstream metric (i.e., raw application availability) can be aggregated to produce the measurements of this composite availability metric. Other contextual details are not specified as we cover the whole application and not its components (i.e., object context is not required) and as these can be inferred by the platform itself (e.g., composite metric computation frequency).

The application's CAMEL model ends with the specification of a small constraint model that poses the constraint on average application availability to be at least 99.999%. This is the constraint that is used for the specification of the sole SLO in the application's requirement model.



```
 requirement model ReqModel {
  link requirement DU_CU_CP_COMREQ {
   attribute Throughput
    [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Network
    Entity.Network.NetworkQoS.hasTxput] : double 0.1
   attribute Latency
    [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Network
    Entity.Network.NetworkQoS.hasLatency] : double 5.0
  }
  link requirement DU_CU_UP_COMREQ{
   attribute Throughput
    [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Network
    Entity.Network.NetworkQoS.hasTxput] : double 4.0
   attribute Latency
    [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Network
    Entity.Network.NetworkQoS.hasLatency] : double 1.0
  }
  resource requirement RES{
   attribute RAM
    [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Cloud.V
    MFlavor.hasRAM] : int 2 TrafficSimulationUF.CRMUnitModel.GigaBytes
   attribute CORES
    [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processi
    ng.CPU.hasMinNumberofCores] : int 5
  }
  location requirement PL [Locations.PL]
  slo AvailSLO constraint ConstrModel.AvailConstr
 }
 metric type model MetrModel{
  composite metric context AvgAvailContext{
   metric MetricTemplateCamelModel.MetricTemplateModel.AverageAvailability
   grouping global
  }
 }
 constraint model ConstrModel{
  metric constraint AvailConstr : [ MetrModel.AvgAvailContext] >= 0.99999
 }
}
```

Fig. 34 - Snippet of application's CAMEL model covering the requirement, metric and constraint domains

Thus, as it can be easily inferred, CAMEL 3.0 and enhanced MDS were able to cover the complete modelling of the static scenario of the RAN use-case of IS-Wireless thanks to the slight extensions conducted on CAMEL 2.0 (with respect to the communication requirement specification) as well as on the MDS (with respect to the network domain). The dynamic scenario of this use-case is still under development and will exploit the Morphemic Preprocessor features corresponding to the polymorphic proactive and reactive adaptation. Potentially, its complete modelling in CAMEL 3.0 could be shown in the next edition of this deliverable, which will cover the improvements on CAMEL 3.0 and MDS based on the partner feedback as well as full demonstrations of CAMEL & MDS to explicate their enhanced expressiveness power.

6 Conclusions & Future Work

This deliverable has attempted to explain all the modelling enhancements that have been achieved in the Morphemic project with the main aim to support the complete modelling of polymorphic applications, including the coverage of proactive adaptation aspects. Such enhancements came with the extension of two related modelling frameworks, the CAMEL language and the Metadata Schema (MDS) ones.

In case of CAMEL, a new version called CAMEL 3.0 has been produced that enables to completely model multi-cloud polymorphic applications by satisfying all relevant requirements as were introduced in section 3.1.1. This is a preliminary version that might be modified in the near future so as to accommodate relevant feedback from both the technical and use-case partners of the Morphemic project. Thus, the potential evolution of this language along with a complete example of its modelling in terms of an advanced, polymorphic-enabled scenario of a use-case will be covered in the next version of this deliverable, which is named as D1.3 "Final Data, Cloud Application & Resource Modelling". That deliverable will also cover complete pointers to all relevant CAMEL v3.0 sources (documentation, examples, presentations, relevant platform implementation details, etc.) in order to enable also an external audience to fully comprehend CAMEL and become capable of modelling polymorphic multi-cloud applications through this language.

The MDS schema was already quite rich when its current version has been produced in the context of the Melodic project, properly covering basic cloud resource and service types as well as multiple data management aspects. However, in the context of polymorphic application modelling as well as supporting all use-case applications of the project with some domain-specific peculiarities, MDS has been extended in order to cover a richer set of resources, including now HPC, edge and hardware-accelerated ones, as well as the network domain, quite relevant for the Morphemic use-cases, enabling to include those concepts that are necessary to support network-aware application deployment, such as those related to communication quality. MDS will be evaluated in the context of the project use-cases and might be enhanced to cover additional concepts and properties, if this becomes necessary. All additional MDS enhancements as well as corresponding changes will be incarnated in the next version of this deliverable, i.e., the D1.3 deliverable.

To this end, the forthcoming deliverable will cover the final versions of both modelling frameworks which will surely enable the complete modelling and consequent adaptation of polymorphic, multi-cloud applications that might also include data-intensive tasks. Such frameworks already advance the competition and pave the way for additional advancement in the context of cloud application modelling.

7 References

[1] K. Kritikos, P. Skrzypek, and F. Zahid, "Are Cloud Platforms Ready for Multi-cloud?," in *Service-Oriented and Cloud Computing*, vol. 12054, A. Brogi, W. Zimmermann, and K. Kritikos, Eds. Cham: Springer International Publishing, 2020, pp. 56–73.

[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] Y. Verginadis, I. Patiniotakis, and G. Mentzas, "Metadata Schema for Data-Aware Multi-Cloud Computing," in *2018 Innovations in Intelligent Systems and Applications (INISTA)*, Thessaloniki, Jul. 2018, pp. 1–9, doi: 10.1109/INISTA.2018.8466270.

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

[5] Brice Morin, Olivier Barais, Jean-Marc Jézéquel, Franck Fleurey, and Arnor Solberg, "Models@run.time to Support Dynamic Adaptation," *Computer*, vol. 42, no. 10, pp. 44–51, 2009, doi: 10.1109/MC.2009.327.

[6] C. Chapman, W. Emmerich, F. G. Márquez, S. Clayman, and A. Galis, "Software architecture definition for on-demand cloud provisioning," *Cluster Comput*, vol. 15, no. 2, pp. 79–100, Jun. 2012, doi: 10.1007/s10586-011-0152-0.

[7] 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.

[8] 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.

[9] 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.

[10] D. Palma and T. Spatzier, "Topology and Orchestration Specification for Cloud Applications (TOSCA)," Organization for the Advancement of Structured Information Standards (OASIS), Jun. 2013. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/cos01/TOSCA-v1.0-cos01.pdf.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

[19] T. Fitz, M. Theiler, and K. Smarsly, "A metamodel for cyber-physical systems," *Advanced Engineering Informatics*, vol. 41, p. 100930, Aug. 2019, doi: 10.1016/j.aei.2019.100930.

[20] Alessandra De Paola, Luca Gatani, Giuseppe Lo Re, Alessia Pizzitola, and Alfonso Urso, "A Network Ontology for Computer Network Management," Consiglio Nazionale delle Ricerche, Palermo, Italy, Technical Report RT-ICAR-PA-03-22, 2003.

[21] K. G. Kyriakopoulos, D. J. Parish, and J. N. Whitley, "FlowStats: An ontology based network management tool," in *2015 Second International Conference on Computing Technology and Information Management (ICCTIM)*, Johor, Malaysia, Apr. 2015, pp. 13–18, doi: 10.1109/ICCTIM.2015.7224586.

[22] Andriy Luntovskyy, Taissia Trofimova, Natalia Trofimova, Dietbert Gütter, and Alexander Schill, "To a Proposal towards Standardization of Network Design Markup Language," Dresden, Germany, 2007.

[23] Q. Zhou, A. J. G. Gray, and S. McLaughlin, "ToCo: An Ontology for Representing Hybrid Telecommunication Networks," in *The Semantic Web*, vol. 11503, P. Hitzler, M. Fernández, K. Janowicz, A. Zaveri, A. J. G. Gray, V. Lopez, A. Haller, and K. Hammar, Eds. Cham: Springer International Publishing, 2019, pp. 507–522.
[24] M. A. Rahman, A. Pakstas, and F. Z. Wang, "Towards Communications Network Modelling Ontology for Designers and Researchers," in *2006 International Conference on Intelligent Engineering Systems*, London, UK, 2006, pp. 258–263, doi: 10.1109/INES.2006.1689380.
[25] A. Uzun and A. Küpper, "OpenMobileNetwork: extending the web of data by a dataset for mobile networks and devices," in *Proceedings of the 8th International Conference on Semantic Systems - I-SEMANTICS '12*, Graz, Austria, 2012, p. 17, doi: 10.1145/2362499.2362503.
[26] J. J. van der Ham, "A semantic model for complex computer networks: the network description language," [s.n.], S.l, 2010.
[27] X. Qiao, X. Li, and J. Che, "Telecommunications Service Domain Ontology: Semantic Interoperation Foundation of Intelligent Integrated Services," in *Telecommunications Networks - Current Status and Future Trends*, J. Ortiz, Ed. InTech, 2012.
[28] SAKTHI MURUGAN, P. SHANTHI BALA, and G. AGHILA, "An Ontology for Exploring Knowledge in Computer Networks," *International Journal on Computational Sciences & Applications (IJCSA)*, vol. 3, no. 4, 2013.
[29] G. G. Castañé, H. Xiong, D. Dong, and J. P. Morrison, "An ontology for heterogeneous resources management interoperability and HPC in the cloud," *Future Generation Computer Systems*, vol. 88, pp. 373–384, Nov. 2018, doi: 10.1016/j.future.2018.05.086.
[30] A. Zhou, K. Ren, X. Li, W. Zhang, and X. Ren, "Building Quick Resource Index List Using WordNet and High-Performance Computing Resource Ontology towards Efficient Resource Discovery," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Zhangjiajie, China, Aug. 2019, pp. 885–892, doi: 10.1109/HPCC/SmartCity/DSS.2019.00129.
[31] C. Kessler, L. Li, A. Atalar, and A. Dobre, "XPDL: Extensible Platform Description Language to Support Energy Modeling and Optimization," in *2015 44th International Conference on Parallel Processing Workshops*, Beijing, China, Sep. 2015, pp. 51–60, doi: 10.1109/ICPPW.2015.17.
[32] Object Management Group, "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems," Object Management Group (OMG), OMG Document formal/2009-11-02, 2009. [Online]. Available: https://www.omg.org/spec/MARTE/1.0/PDF.
[33] Imran Rafiq Quadri, Samy Meftali, and Jean-Luc Dekeyser, "A Model based design flow for Dynamic Reconfigurable FPGAs," *International Journal of Reconfigurable Computing*, 2019.
[34] 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.
[35] J. Domaschka, F. Griesinger, D. Baur, and A. Rossini, "Beyond Mere Application Structure Thoughts on the Future of Cloud Orchestration Tools," *Procedia Computer Science*, vol. 68, pp. 151–162, 2015, doi: 10.1016/j.procs.2015.09.231.
[36] Yiannis Verginadis *et al.*, "D2.2 Architecture and Initial Feature Definitions," Melodic Project Deliverable, Feb. 2018.

Appendix I

In this appendix, we provide all the details of the main MDS classes, subclasses and properties that were considered for extensions as part of the advanced requirements and capabilities of the MORPHEMIC platform. We have used different font colour to highlight the updates introduced in MDS as part of this work.

Table 4: Extensions of the *Iaas/Processing* Class with respect to *CPU* and *HPC* classes

Class Taxonomy Levels					**Properties**	**Description**
IaaS						This class encapsulates all the attributes related to cloud infrastructural resources that are required and offered for deploying Morphemic-enabled applications. It reuses and extends the requirement model of CAMEL (Rossini et al., 2015).
	Processing					This class involves any infrastructural feature bound to the processing capability of virtualised resources.
					hasPower Consumption	This data property refers to the estimated consumption of electricity required for processing.
					hasProcessingCost	This data property refers to the estimated cost for processing
					hasProcessingNodeLocation	This object property has range the Location class (of the Security Context Element) to define the physical or network of the processing node.
		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.
					hasCPU Utilization	This property associates the CPU class with a double that represents the current percentage of use for a certain processing unit.
					hasMIPs	This property associates the CPU class with an integer that expresses million instructions per second as a measure of processing speed supported by a certain IaaS resource.
					hasMFLOPs	This property associates the CPU class with an integer that represents the capability for mega floating-point operations per second.
					has Manufacturer	This property expresses as a string the manufacturer of the certain processing unit.
					hasMin Numberof Cores	This property denotes an integer that captures the minimum number of CPU cores available or requested.
					hasMax Numberof Cores	This property denotes an integer that captures the maximum number of CPU cores available or requested.
					hasFrequency	This property captures the CPU performance as it specifies the operating frequency of the CPU cores, expressed in cycles per second. It associates the CPU class with an integer value.
					hasPipeline Number	The number of pipelines per core included in this processor
					hasStages Number	The number of (processing) stages per pipeline involved in this processor
					hasALU Number	The number of ALUs (Arithmetic Logic Units) within this processor

					hasCaches	The range of this object property is the Cache class in order to define the set of caches involved in the processor
		HPC				This class refers to the high performance features provided by the computing platform which can boost the overall performance of the application
			CPU pinning			This class denotes capability of the platform to associate guest virtual CPUs with hosts physical CPUs
					hasCPinning Policy	This property is an enumeration of possible associations between vCPUs in guest and the pCPUs in the host. The acceptable values are: • DEDICATED: Virtual CPUs are pinned to physical CPUs • SHARED: Multiple VMs may share the same physical CPUs • ANY: (Default) Any policy is acceptable
			CPU thread pinning			This class enables utilization of thread pinning functionality
					hasTPinning Policy	This property describes how to place the guest CPUs when the host supports hyper threads. The acceptable values are: • AVOID: Avoids placing a guest on a host with threads. • SEPARATE: Places vCPUs on separate cores and avoids placing two vCPUs on two threads of same core. • ISOLATE: Places each vCPU on a different core and places no vCPUs from different guests on the same core. • PREFER: Attempts to place vCPUs on threads of the same core.
			Huge pages			This class describes the capability to allocate different size of memory pages
					hasPageSize	This property is an enumeration of possible sizes of allocated memory pages: • LARGE: Require hugepages (either 2MB or 1GB) • SMALL: Doesn't require hugepages • SIZE_2MB: Requires 2MB hugepages • SIZE_1GB: Requires 1GB hugepages • PREFER_LARGE: Application prefers hugepages
			PCIe device			This class refers to the direct and exclusive access of the VM to the PCI device(s) via PCI passthrough capability
					hasDeviceID	This property describes the targeted device by its ID
					hasNumber	This property denotes the number of devices of the same ID to be attached to the VM
			NUMA policy			This class defines a non-uniform memory access (NUMA) topology of the guest. Specifically identifies if the guest should be run on a host with one NUMA node or multiple NUMA nodes.

					hasNodeCount	This property expresses the number of NUMA nodes to be exposed to the VM.
				NUMA node		This class specifies the characteristics of a Numa node
					hasNodeID	This property denotes the NUMA node ID
					hasvCPU	This property denotes the list of VPCUs to allocate on this NUMA node.
					hasMemory	This property denotes the memory size expressed in MB for this NUMA node.

Table 5: Extensions of the *Iaas/Processing* Class with respect to *Memory* classes

	Memory					Represents any kind of memory that can be used for storage or processing purposes
					hasMemory Size	The actual size of the memory
					has Address Size	The size of the address (space) in the memory
					hasMemory Throughput	The throughput supported by the memory
					hasClock Frequency	The clock frequency of the memory
		ProcessingMemory				A memory that can be used for processing purposes
					hasReplicationPolicy	The replication policy (e.g., LRU, NFU) adopted by this processing memory
					hasWritePolicy	The write policy (e.g., write back or write through) adopted by this processing memory
			Cache			A kind of extremely fast processing memory that acts as a buffer between the RAM and the CPU. It actually holds frequently requested data and instructions so that they can be immediately available to the CPU when requested
					hasLevel	The actual level of the cache (memory)
					hasCacheType	The type of the cache (e.g., data, instruction, unified, etc.)
				Cache Structure		This subclass refers to the structure of a cache memory
					hasSetNumber	The number of sets (i.e., groups of blocks)
					hasBlockSize	The size of the cache memory blocks
					hasAssociativity	The associativity of the cache
			RAM			This class corresponds to A kind of processing memory that can be read and written in any order. It is typically exploited for the storage of working data and machine code.
					hasFreeMemory	This property associates the RAM class with a value expressed in double-precision floating-point format (double) that denotes the amount of unused memory currently available by the virtualised resource.
					hasUsedMemory	This property associates the RAM

						class with a value expressed in double format that denotes the amount of used memory in the virtualised resource.
					hasManufacturer	This property associates the RAM class with a string that denotes the producer of the hardware.
					isSynchronous	Indicates whether the RAM is synchronous (i.e., clocked) or not
					isStatic	Indicates whether the RAM is static or dynamic
					isNonVolatile	Indicates whether the RAM is non-volatile
					hasRAMType	The type of the RAM (e.g., SRAM, DRAM, etc.)
				Total Memory		This subclass captures the desired or offered value of the virtualised storage dedicated for frequent program instructions.
					hasMin	This property associates the Total Memory class with an integer that represents the least amount of memory capacity required or offered.
					hasMax	This property associates the Total Memory class with an integer that represents the largest amount of memory capacity required or offered.
					hasUnit	This property associates the Total Memory class with a string that represents the measurement module of the memory capacity.
		MemoryOrganisation				The organisation of a memory. A memory is usually organised under banks of rows and columns of words.
					hasRowNumber	The number of rows
					hasColumn Number	The number of columns
					hasBank Number	The number of banks
					hasWordSize	The size of words
		Storage Memory				A memory utilised for (permanent) storage purposes.
			ROM			A kind of storage, read-only memory which is non-volatile. The data stored in a ROM cannot be electronically modified after the production of the memory device
					hasMemory Organisation	This object property associates with the Memory Organisation class to define the organisation aspects of this ROM memory
					hasROMType	The type of the ROM (e.g., masked, EPROM, etc.)
			Drive			A kind of mass storage memory that can take various forms like hard disk drives and optical disk drives
					hasSectorSize	The size of sectors in the drive

Table 6: Extensions of the *Iaas/Processing* Class with respect to *Accelerator* class

		Accelerator				This class refers to application-specific hardware designed or programmed to compute operations faster than a general-purpose computer processor.
			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.
					hasStart UsageDate	This property denotes the date when a certain GPU began to operate. It can be used as an attribute that reveals how new the processing units used by a certain IaaS resource are.
					has Manufacturer	This property expresses as a string the manufacturer of the processing unit.
					hasMFLOPs	This property associates the GPU class with an integer that represents the capability for mega floating-point operations per second, which is a common measure of processing speed.
					hasPEperCUs	This property expresses with an integer the number of processing elements per compute units that a certain GPU offers.
					hasWarpSize	This property expresses as an integer the number of threads supported to coalesce memory access and instruction dispatch.
					hasMax Concurrent Workgroups	This property denotes an integer that represents the maximum work-groups that may be simultaneously executed on compute units supported by a certain GPU.
					hasMin Numberof Cores	This property denotes an integer that captures the minimum number of GPU cores available or requested.
					hasMax Numberof Cores	This property denotes an integer that captures the maximum number of GPU cores available or requested.
					hasGPUtype	The type of the GPU (e.g., integrated, dedicated, hybrid, etc.)
					hasVMSharing	Indicates whether the GPU can be shared among multiple VMs
					has Architecture	The kind of architecture involved in the GPU (e.g., AMD_SOUTHERN_ISLANDS)
					hasSIMD	The number of SIMDs (Single

					PerCore	Instruction, Multiple Data) per core in the GPU
					hasScalarUnitNumber	The number of scalar units in the GPU
					hasCache Coherence Level	The cache coherence level in the GPU
					hasALUPer SIMD	The number of ALUs per SIMD in the GPU
					hasFPUPer SIMD	The number of FPUs per SIMD in the GPU
					hasSupported APIs	The APIs supported by the GPU (e.g., CUDA, OpenGL, etc.)
					hasTDP	Stands for Thermal Design Power and maps to the maximum amount of heat the core of the GPU generates under intense workload
					hasTBP	Stands for Total Board Power and maps to the maximum amount of total heat that the whole GPU board (including auxiliary systems) generates under intense workload
					hasBusWidth	The width of the bus in the GPU
					hasAverage Component FeatureSize	The average feature size across the components of the GPU
					hasDieSize	The size of the die in the GPU
					hasTransistorNumber	The number of transistors in the GPU
				GPUPerformance		This class encapsulates all the aspects that characterize the running status of a GPU
					hasGPU Utilization	This property associates the GPU class with a double that represents the current percentage of use for a certain processing unit.
					hasClock Speed	This property captures the GPU operating speed expressed in cycles per second. It associates the GPU class with an integer value.
					hasCompute Capability	The compute capability of the GPU in terms of the feature supported by the CUDA hardware. There are actually respective levels that can be mapped to double values starting from 1.0.
					hasMillion Shader OperationsPerSecond	Amount of million shader operations per second that can be reached through this GPU
					hasMillionTexels PerSecond	Amount of million texels (i.e., pixels in 3 dimensions) per second that can be rendered by the GPU
					hasMillion VerticesPer Second	Amount of million vertices per second that can be processed by the GPU

					hasMillion PixelsPer Second	Amount of million pixels per second that can be processed by the GPU
					hasMaxBoost Clock Frequency	The maximum clock frequency on which the GPU can operate that can lead to maximizing its performance
			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.
					hasStart UsageDate	This property denotes the date when a certain FPGA began to operate. It can be used as an attribute that reveals how new the processing units used by a certain IaaS resource are.
					has Manufacturer	This property expresses as a string the manufacturer of the FPGA.
					hasName	This property expresses as a string the name of the FPGA board.
					hasVersion	This property expresses as a string the version of the FPGA shell.
					hasList Accelerators	This property associates the FPGA class with a list of strings, being the names of the accelerators in the current bitstream loaded
					hasLogical BlockNumber	The number of logical blocks in the FPGA
					hasCellsPer Block	The number of cells per (logical) block in the FPGA
					hasClock Number	The number of clocks in the FPGA
					has Components	A set of components that comprise or connect with this FPGA. Such components could take the form of multi-gigabit tranceivers, ethernet medium ACUs, interconnect bridges, etc.
					hasIOBlock Number	The number of IO blocks in the FPGA
					hasDSPBlock Number	The number of DSP (Digital Signal Processor) blocks in the FPGA
					hasDSP Architecture Features	The features exhibited by the adopted DSP architecture in the FPGA. Such features can be fixed-point arithmetic, multiple data memories, MAC support, etc.
					hasIOBlock Number	The number of IO blocks in the FPGA
					hasAnalog Features	The analog features in the FPGA, such as quartz crystal oscillators and slew rate output pins.

					hasSecurity	This object property associates with the FPGASecurity class (subclass of HardwareBasedSecurity) in order to define the security characteristics of the FPGA.
				FPGAMemory		This subclass is used to describe the memory capacity of FPGA accelerators involved.
					hasTotalFPGAMemory	This property associates the Memory class with an integer that represents the total amount of memory capacity required or offered.
					hasUnit	This property associates the Memory class with a string that represents the measurement module of the memory capacity.
					hasMemory Consumption	This property associates FPGA class with a double that represents the current percentage of memory used for a certain FPGA board.
					hasRAMBlockNumber	This property refers to the number of RAM blocks in the FPGA.
					hasExternal Memories	This object property associates the FPGAMemory with the RAM class in order to express the set of external memories utilised by the FPGA**.**
				FPGA Performance		This subclass encapsulates the measured capabilities of a certain FPGA board based on its current usage.
					hasWaiting Time	This property associates FPGA class with and integer that represents the requests waiting in a specific accelerator's queue to be executed
					hasReadTime	This property associates FPGA class with a double that represents the time taken to copy data from host memory to the FPGA double data rate (DDR) for a specific accelerator.
					hasExecute Time	This property associates a FPGA class with a double that represents the time taken to execute a request on a specific FPGA accelerator.
					hasWriteTime	This property associates a FPGA class with a double that represents the time taken to copy data from FPGA DDR to host memory for a specific accelerator.
				Status		This subclass encapsulates aspects about the current status of a certain FPGA board.
					hasRunning	This property associates FPGA class with an integer (1-3) that

						represents which stages (read – write - execute) of a specific accelerator's software pipelinening are active
					hasPower Consumption	This property associates the FPGA class with an integer that represents the current power consumption in watts (W).
					has Temperature	This property associates the FPGA class with an integer that represents the current temperature of the FPGA die in degrees Celsius (C).
				FPGAHardCore		A kind of FPGA whose logic includes hard-macro processors
				FPGASoftCore		A kind of FPGA whose logic involves the use of soft processor IP cores
				Heterogeneous FPGA		An architecture that involves multiple FPGA dies stacked side by side on a silicon interposer. This architecture enables to different parts of the FPGA to be developed according to different process technologies.
					hasDieNumber	The number of dies stacked side by side on the silicon interposer of the FPGA
				FPGA Interconnection		The interconnection in the FPGA modelled as a concept with multiple attributes.
					hasRouting Channel Number	The number of routing channels in the FPGA interconnect
					hasRouting ChannelWidth	The width of each routing channel in the FPGA interconnect
					horizontal Routing Channel Number	Number of horizontal routing channels in the FPGA interconnect
					Vertical Routing Channel Number	Number of vertical routing channels in the FPGA interconnect
					hasConnectionNumberPer Block	The number of connections per block in the FPGA interconnect
					hasConnectionBlockNumber	The number of connection blocks in the FPGA interconnect
					hasConnectionType	The types of connections supported in the FPGA interconnect
					hasSwitch BlockNumber	The number of switch blocks in the FPGA interconnect
					hasRouting Architecture	The routing architecture adopted by the FPGA interconnect. Architectures such as hierarchical, island style, etc could be followed.
					hasInterlaken Support	Support for the interlaken chip-to-chip interconnect protocol in the FPGA interconnect
					hasEthernet Support	Support for the Ethernet protocol in the FPGA interconnect
			VPU			Vision Processing Unit (VPU)

						class refers to a category of microprocessors intended to accelerate machine vision algorithms and tasks.
					hasVPU Interface	This property refers to the capability of a VPU to digest data directly from cameras bypassing any off-chip buffers.
					hasVPU Throughput	This property refers to the possible on-chip data flow supported between the parallel execution units.
			ASIC			Application-Specific Integrated Circuit (ASIC) class refers to a category of hardware accelerators that employ strategies such as optimised memory use and the use of lower precision arithmetic to accelerate calculation and increase the throughput of computation.
					usesFloating PointFormat	This property clarifies the ow-precision floating-point formats that can be used in this ASIC for supporting the acceleration. The possible values are: • Half-precision binary floating-point computer number format that occupies 16 bits in memory and can express values in the range ±65,504, with precision up to 0.0000000596046. • bfloat16: it represents a wide dynamic range of numeric values by using a floating radix point and occupying 16 bits in computer memory.
				TPU		Tensor Processing Unit (TPU) refers to a class of specialised custom-made circuits that implement all the necessary control and arithmetic logic (e.g. matrix multiplications) in a way that accelerates neural network training.
					hastTPUType	This property refers to the kind of TPU that can be employed. The possible values are: • Cloud: TPU pod accessible from public data centres. • Edge: custom-built TPUs for applications that reside at the edge.
					hasGeneration	The generation to which this TPU belongs (e.g., first, second, etc.)

					hasTPU Memory	The RAM memory that can be included in this TPU
					hasOnChip Memory	The amount of on-chip memory available in this TPU
					hasTops	Stands for tera operations per second that can be supported by the TPU
					hasGeneration	The generation to which this TPU belongs (e.g., first, second, etc.)

Table 7: Extensions of the *Iaas/NetworkEntity* Class

Class Taxonomy Levels					**Properties**	**Description**
IaaS						This class encapsulates all the attributes related to cloud infrastructural resources that are required and offered for deploying Morphemic-enabled applications. It reuses and extends the requirement model of CAMEL (Rossini et al., 2015).
	NetworkEntity					This class represents any kind of network entity that could be included in a specific cloud
		Network				This class refers to the network related aspects that bind the operation and communication capabilities of an offered or a requested Fog IaaS resource.
					hasNetType	The type of the network (e.g., mobile, fixed, wired, wireless)
					hasTopology	The topology of the network which can take various forms (mesh, point2point, start, etc.)
					hasNodes	This object property has range the NetworkNode class to define the nodes comprising this network.
					hasServices	This object property has range the Service class (subclass of SoftwareNetwokEntity) to define the services (e.g., voice, video, etc.) offered by this network
					hasMedia	The transmission media (e.g., coaxial cable, radio waves, etc.) used for the communication within the network
					hasQoS	This object property associates the Network class with the NetworkQoS class that comprises a set of well-known QoS parameters/attributes
					hasScale	The scale of the network (e.g., MAN, WAN, etc.)
					hasScope	This property denotes if a network resource can initiate and receive communication over the internet or the intranet.

					hasNAT	Indication of whether NAT is supported
					hasIPV4Range	The range of IPs (version 4) that can be used in the network
					hasIPV6Range	The range of IPs (version 6) that can be used in the network
					has Configurations	A set of (security) configurations of the network that map to network access rules
					has Availability Zone	This object property associates the Network class to the Availability Zone class of the Big Data Model for denoting isolated locations used to make network resources available.
					hasNetwork Cost	This property denotes a float value that expresses the cost for exploiting the network resources.
			Sub Network			This class represents a subnetwork configured on Cloud or Fog network resources
					IsOnNetwork	This property denotes a network object to be operated by a given network resource.
					IsIsolatedFrom Subnetwork	This boolean property associates with another subnetwork and assesses whether or not these two are isolated (e.g. VLAN).
			Virtual Private Cloud Network			A virtual private cloud (VPC) is a kind of a network entity that refers to the network access of an on-demand configurable pool of computing resources allocated within a public cloud. It supplies a certain level of isolation between the clients of the respective cloud. Such an isolation (per client) is achieved through a private IP subnet and a virtual communication construct.
					hasSubnets	This property corresponds to one or more string values that represent the subnet(s) of an IaaS resource that comprise a VPC.
					isExternal	Indicates whether the VPC is external with respect to a specific cloud. In some cases, external VPCs can be paired with one or more internal VPCs of a cloud.
					PairsWith	This object property denotes the VPC with which it is paired the current one
					hasElasticIPs	The set of elastic IPs offered in the VPC

					hasLocation	This object property associates a VPC with the class Location of the Context Aware Security Model
					hasRemote Networks	The set of remote networks that the VPC can or connects with
					has Configurations	Set of (security) configurations of the VPC
					hasTenancy	This Boolean data property indicated whether multi-tenancy is supported or not within this VPC.
		Network QoS				A class that represents the main aspects that express the quality of service of a network
					hasBandwidth	This property associates the Network class with a float value that states the maximum throughput of a logical or physical communication path. It corresponds to the net bit rate, channel capacity or the maximum throughput that can be conveyed per unit of time.
					hasErrorRate	Percentage of bit errors within a time frame. It could be also mapped to error packet ratio, i.e., the number of incorrectly received packets with respect to the total received ones (within a time frame).
					hasJitter	Difference in the end-to-end delay variation between selected packets in a flow with any lost packets being ignored
					hasLatency	This property associates a float value with the maximum latency of a logical communication path.
					hasThroughput	Rate of successful message delivery over a communication channel
					hasPacketLoss	Percentage of packet loss with respect to the number of packets sent (within a particular time frame/space)
		Software Network Entity				This subclass represents a software-based network entity
			Queue			Represents a network queue associated to an interface in which packets can be hosted while waiting for processing or transmission
					hasTotalSize	The total size of the queue
					hasUsedSpace	The percentage of the total queue size allocated
			Routing Table			A table that controls the routing in the context of a specific network node
					hasEntries	This property refers to the set of entries/rows of the table

					isCorrupted	Indicates whether one or more entries of the table have been corrupted
			Routing TableEntry			Represents an entry of a routing table
					hasDestinationNode	This object property has as range the NetworkNode class to define the destination node for the routing
					hasNeighbour Node	A neighbouring node via which the traffic could be directed towards the destination node
					hasCost	The cost of using the neighbouring node to reach the destination node, i.e., of the routing path followed
			Service			Represents a service that can be offered by a network
					hasServiceType	The type of the offered service (e.g., basic, value-added, voice data, etc.)
					hasErrorCode	The code of error that can occur during the use of that service
					hasService Status	The current status of the service (e.g., running, unavailable, etc.)
					isConnection Less	Indicates whether this is a connectionless or a connection-based service
					hasNetwork Protocols	The set of network protocols on which this service relies
			System Software			Represents the software that runs in a network node
			Virtual Node			A virtual node of the network that can run on top of a hardware/physical node
			Routing Protocol			A protocol that can be used in routing
					hasRPType	The type of the routing table (link state, distance vector, etc.)
			Network Protocol			System of rules that allow two or more entities of a communication system to transmit information via any kind of variation of a physical quantity. Such a protocol defines the rules, syntax, semantics and synchronisation of the communication as well as possible error recovery methods.
					hasLayer	The relevant layer(s) on the OSI model where this protocol applies
					hasProtocol Suite	The protocol suite (e.g., AppleTalk or Internet Protocol) to which the protocol may belong
			Network Operating System			A specialised operating system to be operated on a network entity like a router, switch or

						firewall
		HardwareNetwork Entity				This class refers to all the aspects of hardware-based network entities that may constitute network nodes serving the cloud application components.
					hasHNStatus	The status of the hardware network entity which can be on, off or abnormal
			Network Node			A physical node of the network
					hasNeighbours	A set of neighbouring network nodes to this node. It is a recursive object property
					hasRouting Table	The routing table exploited for routing purposes for this node
					hasLocation	This object property has as range the Location class (of the Security Context Element) in order to define the (exact) location of this network node
					hasVirtual Nodes	The set of virtual (network) nodes executed on top of this (physical) network node
				System Device		A system device is a kind of network node that has a specific location within a network and plays a certain role within it (e.g., bridge, base station, access point, etc.)
					hasSoftware	This object property has range the SystemSoftware class to define that a system device might run specialised system software
					hasOperating System	This object property has range the NetworkOperatingSystem class to define that a system device might run a (specialised) (network) operating system
				Access Point (one level down the System Device)		Represents a device that allows other devices to connect to a wired network through the use of wireless LAN technology such as Wi-Fi. Usually connects to a router as a standalone device or can be an integral component of the router itself.
					isWifi	Indicates whether the Wi-Fi wireless LAN technology is utilised by this access point
					hasAssociated Equipment	This object property has range the UserEquipment class to define any associated equipment to this access point
					hasInRange Equipment	This object property has range the UserEquipment class to define any in-range equipment to this access point
					hasDriver	The driver utilised by this

						access point
					hasOpticalFilterGain	The gain involved in the optical filtering
					hasHalf IntensityAngle	The half-intensity angle of a luminous source
					hasResponsivity	The input-output gain of a detector system. There can be electrical and optimal responsivity. The electrical responsivity can be defined as the detector voltage response per incident power. The optical responsivity instead refers to the radiative coupling and includes antenna losses, losses of the feeding network and impedance mismatch between the antenna and the transistor.
				Base Station (one level down the System Device)		A radio receiver/transmitter that serves as a hub of a local wireless network while it might also be a gateway between a wired and a wireless network. Typically consists of a low-power transmitter and a wireless router.
				SatelliteRelay Station (one level down the SystemDevice)		A broadcast transmitter that repeats the signal of a radio or television station to an area not covered by the originating station.
				Bridge (one level down the System Device)		A computer networking / system device that constructs a single, aggregated network from multiple communication networks or network segments
					hasBridgeType	The type of a network bridge (e.g., multi-port, transparent, etc.)
				Hub (one level down the System Device)		A system device that connects multiple Ethernet devices and making them act as a single network segment
					hasHubType	The type of a hub, such as active or passive
				Gateway (one level down the System Device)		A system device that allows data to flow from one discrete network to the other. The communication can happen using more than one (network) protocol while the operation can occur at any level of the OSI model. It is also advertised to provide interoperability between networks that utilise different protocols
					hasGateway Type	The type of a gateway (e.g., synchronization, media, cloud storage, etc.)
					isBidirectional	Whether a gateway is bidirectional or unidirectional

						(when the false value is given for this attribute). A bidirectional gateway also allows the flow of responses (from destination to origin networks)
				Repeater (one level down the System Device)		A system device that receives a signal and retransmits it. This enables signals to cover longer distances or to reach the other side of an obstruction.
					isDigital	Indicates whether it is a digital (or an analog) repeater
					isMultiPort	Indicates whether it is a multi- or single-port repeater
					isSmart	Indicates whether this repeater is smart
					hasRepeater Type	Indicates the type of a repeater (e.g., optical, WiFi, etc.)
				Switch (one level down the System Device one)		Represents a system device that connects (network) devices through a network by using packet switching to receive and forward data towards the destination device
					isMultiOSILayer	Indicates whether the switch operation covers multiple and not just one OSI layer
					hasFormFactor	Indicates the form factor of a switch (e.g., standalone, rack mounted, stackable, etc.)
					hasSwitch Configuration	Indicates the kind of configuration of the switch (e.g., unmanaged, managed, enterprise managed, etc.)
					hasSwitch Method	Indicates the method utilised for the network switching (e.g., cut through, fragment free, etc.)
					hasSpeed	The transmission speed of the switch
					hasPortNumber	The number of ports in the switch
					hasPoE	Indicates whether the switch has a Power over Ethernet injection built-in
				Router (one level down the System Device)		This class refers to a networking device that forwards data packets between networks applying directing functions.
					hasRouterType	This data property refers to which type characterize the router (e.g. wireless, broadband, core, edge).
					supports Dynamic Routing	This Boolean data property refers whether or not adaptive routing is supported.
					hasRouting Protocol	This object property associates Router class with the RoutingProtocol class (of the SoftwareNetoworkEntity) to define the protocol used.

				User Device		A kind of hardware (network) node which can vary in terms of location within a network while it can offer different types of services
					hasAccess PointsInRange	This object property has range the AccessPoint class to define the access points in range of this device
					hasUDStatus	The status (attached, detached, unreachable) of this device
					hasServices	This object property has range the Service class to define the services offered by this device
				User Equipment (one level down the User Device)		A kind of a user device like a phone, laptop, etc.
					hasUEType	The type of the equipment (LiFi, WiFi, Cellular)
				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.
					hasMinNumberOf Interfaces	This property expresses the minimum amount of network interfaces an IaaS offering is requested to have or already has.
					hasMaxNumberOf Interfaces	This property expresses the maximum amount of network interfaces an IaaS offering is requested to have or already has.
					hasSupportFor NetworkingFastPr ocessing	This property denotes a boolean value to assess the capability of network adapter to support fast processing.
					isBoundTo SubNetwork	This object property refers to the subnet associated to the network adapter.
					hasManufacturer	This property expresses as a string the manufacturer of the virtual network adapter.
					hasInQueue	The inbound Queue of the interface
					hasOutQueue	The outbound Queue of the interface
					hasIP	The IP mapping to this interface
					hasMAC	The MAC address mapping to this interface
					hasFrequency	Data transfer frequency
					isStatic	This Boolean property indicates whether the interface is static or configurable
					hasCapacity	This property refers to the maximum throughput, data/rate

						that can be supported by the interface
					hasEncoding Type	The type of encoding performed via the interface (e.g., packet, ethernet, PDH, etc.)
					hasTransport Type	The type of transport supported via this interface, i.e., of the technology used to map the data on to the encoding. Each type maps to a specific value as defined by the IETF. Multiple values for transport type might map to the same encoding type
				WLAN Interface (one level down the Interface)		An interface for WLAN networks
					has Antenna Gain	The gain of the antenna in the (WLAN) interface
					has Antenna Height	The height of the antenna in the (WLAN) interface
				Link (one level down the Interface)		A link enables to establish a connection between the interfaces of two (or more) devices
					hasLType	The type of the link (e.g., Wired, Wifi, Lifi)
					hasLBandwidth	The bandwidth of the link
					usedbandwidth	The used bandwidth of the link
					hasPacketLoss	The percentage of lost packets in the link
					hasRoundTrip Time	The amount of time required for a packet to reach a destination and arrive back through the current link
					hasDataRate	Average number of bits, characters, symbols or data blocks passing through a link per unit time
					hasTransmit Power	The amount of energy needed to transmit a packet via the link
					hasReceive Power	The amount of energy needed to receive a packet via the link
					hasLink Throughput	The throughput of the link
					hasLJitter	The jitter related to this link
					hasReliability	The reliability of this link to carry out its intended functionality, i.e., establish connections and successfully pass data through.
					hasDuplexity	Whether the link is half or full duplex or simplex
					hasBitErrorRate	The rate of bit errors related to this link
				Binary Link (one level down the Link one)		A binary link that connects two interfaces together
					hasFrom Interface	The origin interface of the link

					hasToInterface	The target interface of the link
				Broadcast Link (one level down the Link one)		A link that connects multiple interfaces where information/data is broadcasted over them
					hasBLInterfaces	The set of interfaces associated with this broadcast link
				Cross Connect Link (one level down the Link one)		Represents an internal data transport within a network device, i.e., between the network elements inside such a device
					hasBinaryLinks	The set of binary links associated with this cross-connect link

Table 8: Extensions regarding the *Paas/Serverless* Class

Class Taxonomy Levels					**Properties**	**Description**
PaaS						This class encapsulates all the attributes related to platform level cloud resources that are required and offered for deploying Morphemic-enabled applications
					isOferedby Provider	This object property associates the PaaS class with the Provider Class (of the Application Placement Model) for expressing characteristics and identity of the resource provider.
					usesCloud	This object property associates the PaaS class with the Cloud Class (of the Application Placement Model) for denoting with one reference the characteristics of the underlying IaaS level resources used for offering PaaS services.
					hasCloud Location	This object property associates the PaaS class with the Cloud Location Class (of the Big Data Model) for expressing the network or physical location of the virtualised resource.
					hasCost Function	This property associates the PaaS class with a string that refers to a function that provides an accurate calculation of the expected cost for using platform level services.
					hasAvailability	This property may associate a PaaS resource with a float that expresses the expected uptime of the platform level resource.
					hasPricingType	This Property refers to the different pricing per use schemes that each provider may offer regarding platform level cloud resources.
	Serverless					This class represents a Serverless PaaS, i.e., a PaaS that enables the deployment of functions/serverless components in the cloud

				supportsCICD	Whether the serverless platform supports the continuous integration and deployment of function-based/serverless components
				supports AuthProvider	Indicates whether authentication provider(s) are supported by the serverless platform
				supports AuthorisationModel	This object property has as a range the Authorization class (of the Security Controls) to indicate the authorisation model(s) supported by the serverless platform
				supports Logging	This data property indicates whether the serverless platform supports the logging of serverless function when they are executed
				supports Monitoring	This data property indicates whether the serverless platform supports the monitoring of serverless functions
				supports Testing	This data property indicates whether the serverless platform supports the testing of serverless functions
				hasTesting Types	This data property indicates the types of testing (e.g., unit testing) supported over serverless functions by the platform
				supports Diagnostics	This data property indicates whether the platform provides some diagnostics over the problematic execution/deployment of serverless functions
				onPremise	This data property indicates whether the serverless platform can be installed and operated on premise
		Composition			This subclass represents the technology of a serverless platform to support the composition of functions
		Cost			This subclass represents the cost model of a serverless PaaS
				hasCostPer Million RequestsPer Month	This data property refers to the factor that calculates the cost induced per million of requests issued on deployed functions related to a single account in the serverless PaaS
				hasCostGBPer Second	This data property refers to the memory/duration-based cost factor for a specific user account. It is calculated as the total cost of all function calls for a certain account, considering the memory used/allocated per time used (in seconds) to complete the function call.
		FreeQuota			This subclass refers to a free quota per month that is associated with an account in the serverless PaaS
				hasRequests PerMonth	This data property refers to the number of requests that are offered for free for this account
				hasGBPer SecondsPer Month	This data property refers to the total number of GB per seconds that are allowed for free to be consumed by the user account's functions.
				hasEgress TrafficPer Month	This data property refers to the egress traffic per month that is allowed in total for all functions related to a specific user

					account
		Limits			This subclass represents the set of limitations that are associated with a serverless PaaS.
				allowsNumberOf Functions PerProject	This data property refers to the number of functions per project that are allowed for a user account
				allowsMax Deployment Size	This data property refers to the maximum deployment size of any function associated with a user account
				allowsMax HTTPRequest Size	This data property refers to the maximum size that HTTP requests can have over any function of a user account
				allowsMax HTTPResponseSize	This data property refers to the maximum size that HTTP responses from any function of a user account can have
				allowsMax EventSize	This data property refers to the maximum size that any event can have in the context of a user account
				allowsMax Duration	This data property refers to the maximum duration that any function can have of a user account
				allowsMax BuildTime	This data property refers to the maximum build time that any function can have
				allowsMax InactivityTime	This data property refers to the maximum inactivity time that any function (instance) can have before its container is dropped from the host.
				allowsMax Function InvocationsPer Second	This data property refers to the maximum number of invocations that can be performed over any function
				allowsMaxGHzPerSecond	This data property refers to the maximum processing frequency that can be associated with any function per second
				allowsMaxAPI Read Throughput	This data property refers to the maximum read throughput of a serverless API (Gateway - collection of functions)
				allowsMaxAPIWrite Throughput	This data property refers to the maximum write throughput of a serverless API (Gateway - collection of functions)
				allowsMaxAPI Invocation Throughput	This data property refers to the maximum API invocation throughput of a serverless API (Gateway - collection of functions)
				allowsMax InboundSocketData	This data property refers to the maximum amount of inbound socket data per function
				allowsMax Outbound SocketData	This data property refers to the maximum amount of outbound socket data per function
				allowsMax Socket Connections PerSecond	This data property refers to the maximum number of socket connections per second
				allowsMaxDNSResolutionsPerSec	This data property refers to the maximum amount of DNS resolutions

				ond	per second
				allowsMax InvocationRate	This data property refers to the maximum invocation rate of any function
				allowsMax Concurrent EventSize	This data property refers to the maximum size of concurrent events that can be consumed by a function
				allowsMax IncomingEventSize Throughput	The maximum incoming event size throughput that can be supported
				allowsMaxRAMSize	This data property refers to the maximum amount of RAM that can be allocated for a function
				allowsMax EphemeralDiskCapacity	This data property refers to the maximum ephemeral disk capacity that can be supported per function
				allowsMax NumberOf Processes	This data property refers to the maximum number of processes that can be supported per function
				allowsMax EventNumber PerMinute	This data property refers to the maximum number of events that can be supported/consumed per minute
		EventType			This subclass represents the type of events that can be consumed to trigger a function
			HTTP		This subclass represents an HTTP (request) event/trigger of a function
			Schedule		This subclass represents a schedule-based event that can trigger a function
			WebHook		This subclass represents a web hook-based invocation event of a function
			Trigger		This subclass represents an event trigger in general that may refer to log, storage, poll or push-based triggers.
		Serverless Platform			This subclass represents the serverless platform that offers a serverless PaaS service. <u>*Instances:*</u> *AWS Lambda, Azure Functions, Google Cloud Functions.*
		Serverless Framework			This subclass is a serverless framework that can enable the development and deployment of functions. <u>*Instances:*</u> *OpenWhisk*[15], *Fission*[16], *Kubeless*[17], *Oracle Fn*[18], *Riff*[19]

Table 9: Extensions to the *Paas/Security Controls* classes

Class Taxonomy Levels				**Properties**	**Description and Related Ontology (if any)**
Security Controls					This is a subclass of the PaaS class and refers to all the possible security enforcement mechanisms that may be offered or required as a service for protecting the operation of hosted cloud

[15] https://openwhisk.apache.org/
[16] https://fission.io/
[17] https://kubeless.io/
[18] https://fnproject.io/
[19] https://projectriff.io/

					applications. All its subclasses refer to specific security controls that have been classified based on the latest version of the Cloud Controls Matrix (CSA, 2016) introduced by the Cloud Security Alliance.
				guarantees Non Repudiation	This property refers to a Boolean value that states whether or not the offered PaaS services can provide proof of the integrity and origin of data with high assurance.
	CSA-IAM-02				This class refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Identity & Access Management - Credential Lifecycle / Provision Management. Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	CSA-IAM-09				This class refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Identity & Access Management - User Access Authorization. Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	CSA-IVS-01				This class refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Infrastructure & Virtualization Security - Audit Logging / Intrusion Detection. Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	CSA-IVS-06				This class refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Infrastructure & Virtualization Security - Network Security.
		IPS			This subclass is used to provide information about the characteristics of intrusion prevention systems (IPS) for examining network traffic flows and patterns in order to detect and prevent vulnerability exploits.
			Firewall		Represents a firewall that can be run within a network (in a particular node)
				hasFirewall Type	The type of the firewall (e.g., packet filtering, circuit level gateway, etc.)
				isVirtual	Indicates whether the firewall is virtual or not
		Security Configu-ration			A security configuration over a network entity (such as a network or VPC) that is specified through a set of network access

					rules
			NetworkAccess Rule		A rule that controls the access of a network entity (such as a network or subnet)
				isInbound	Indicates whether the rule concerns inbound or outbound (in case it takes a false value) traffic
				refersTo Protocol	This object property has range the NetworkProtocol class to define the (network) protocol concerned in a certain access rule.
				refersToTCP	Whether it maps to TCP or UDP packets
				hasTarget	The entity (IP or range of IPs) for which the flow is controlled by this rule. It is the source of inbound traffic or the destination of outbound traffic.
				allow	Indicates whether the traffic should be allowed or forbidden (in case a false value is given)
	CSA-IVS-13				This class refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Infrastructure & Virtualization Security - Network Architecture. Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	CSA-GRM-10				This class refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Governance and Risk Management - Risk Assessments. Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	CSA-EKM-02				This class refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Encryption & Key Management - Key Generation Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	CSA-EKM-03				This class refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Encryption & Key Management - Sensitive Data Protection Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	CSA-DSI-07				This class refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain

					entitled as: Data Security & Information Lifecycle Management - Secure Disposal Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	CSA-BCR-02				This class refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Business Continuity Management & Operational Resilience - Business Continuity Testing Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	Hardware Based Security				This class aggregates all the security capabilities that can be offered as a service through dedicated hardware components.
		FPGA Security			This class refers to the security capabilities implanted in a FPGA hardware.
				hasBitstream Encryption	The encryption algorithm utilised for bitstream encryption
				hasBitstream Authentication	The method used for bitstream authentication
				hasNonvolatileFlashMemory	Indicates whether a non-volatile flash memory is utilised in the FPGA
				hasFlash MemoryLookupTable	Indicates whether a flash memory lookup table exists in the FPGA
				hasSecure DeviceManager	Examines whether a secure device manager is exploited in the FPGA
				hasPhysically Unclonable Functions	Examines whether the supported functions in the FPGA are physically uncloneable
				hasBootCode Authentication	Examines the support for boot code authentication
				hasSide ChannelAttack Protection	Indicates whether the FPGA can be protected from side channel attacks
		Secure Enclave			This class refers to the capability of a Trusted Execution environment based on dedicated microkernels that support isolation and encryption. Instances: Intel SGX (Software Guard Extensions), ARM Trustzone, AMD SEV (Secure Encrypted Virtualization)

Table 10: Extensions with respect to the *Big-Data Model/Big Data Aspects classes*

Class Taxonomy Levels				***Properties***	**Description and Related Ontology (if any)**
Big Data Aspects					This class encapsulates all the attributes that can be used in order to describe the main characteristics of big data to be processed by Morphemic-enabled cloud applications hosted on multi-clouds. Based on such attributes, preferences on quantitative and qualitative dimensions of virtualized

					resources can be expressed.
				hasData Owner	This property associates the Big Data Aspects class with the Subject class of the Context Aware Security model in order to express the owner of the data to be handled by a Morphemic-enabled application.
				hasData Location	This property associates the Big Data Aspects class with the Data Location class of the Big Data Model in order to denote where certain data artefacts may be found.
	Data Density				This subclass reveals details on big data observed or expected velocity and volume. Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	Data Variety				This class refers to the different types of data that should be processed by a Morphemic-enabled cloud application, stating an increased diversity of data that should be stored, processed or combined.
		Format			This subclass refers to the structural variety that big data may involve which is expressed using certain schemes and models. **Instances:** BLOB (binary large object), JSON, XML, File, Key-Value Pairs, String, Event, CSV, RDD (Resilient Distributed Datasets)
		Type			This subclass refers to the media variety that big data may involve with respect to the medium in which data get delivered. **Instances:** Audio, Image, Video, Text
	Data Value				This class refers to big data aspects that reveal the business importance of data which is bound to the potential of improving a business entity's decision making capabilities. Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	Data Quality				This class encapsulates another group of important big data concepts that reveal aspects about how accessible, secure, compact, volatile or uncertain the data is. Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).

Table 11: Extensions with respect to the *Big-Data Model/Data Location classes*

Class Taxonomy Levels				**Properties**	**Description and Related Ontology (if any)**
Data Location					This class encapsulates all the concepts that can be used for describing the origin of data or the current or required physical/ network location where the data can be stored or processed by a Morphemic-enabled application.
				isStorage Location	This data property associates the Data Location class with a Boolean value that specifies whether or not the data location mentioned is where the data will be stored or processed.
				sameAs	This object property associates the Data location class to another Data location recursively in order to facilitate the expression of requirements that dictate the use of the same location(s) as the ones previously selected for other data artefacts.
				notSameAs	This object property associates the Data location class to another Data location recursively in order to facilitate the expression of requirements that forbid the use of the same location(s) as the ones previously selected for other data artefacts.
				hasSparsity	This data property associates the Data location class to a string

					that denotes how distributed (e.g. Low, Medium, High) are the data sources or data locations exploited for producing a dataset to be processed by a Morphemic-enabled application.
				has Preferred Location	This object property associates the Data location class to the Location class of the Context Aware Security Model in order to facilitate the expression of preferences for using certain network, physical and/or cloud location(s) in order to store or process data artefacts.
				has Allowed Location	This object property associates the Data location class to the Location class of the Context Aware Security Model in order to facilitate the expression of permitted network, physical and/or cloud location(s) for storing or processing data artefacts.
				has Disallowed Location	This object property associates the Data location class to the Location class of the Context Aware Security Model in order to facilitate the expression of forbidden network, physical and/or cloud location(s) for storing or processing data artefacts.
				hasPhysical Location	This object property associates the Data Location class to the Physical Location from the Context Aware Security model in order to define the concrete physical region where data may be stored or processed.
				hasNetwork Location	This object property associates the Data Location class to the Network Location from the Context Aware Security model in order to define the network region where data may be stored or processed.
	Origin				This subclass involves all the relevant concepts for defining the source location of the data artefacts to be processed by a Morphemic-enabled application.
				hasCloud Location	This object property associates the Origin class to the Cloud Location class from the Context Aware Security model in order to define the positioning of Data to certain Cloud host infrastructures.
				hasEdge Location	This object property associates the Origin class to the Physical Location class from the Context Aware Security model in order to define the positioning of Data to certain physical areas or points that denote hosting on Edge nodes.

Table 12: Big-Data Model/Data Timestamp and *Data Domains* classes descriptions provided for overview reasons

Data Timestamp					This class includes all the necessary concepts for describing the temporal characteristics of data artefacts to be processed by a Morphemic-enabled application. Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
Data Domains					This class encapsulates all the relevant concepts that characterize data based on the industries that produce it or need to extract information from it (Murthy et al., 2014). Specifically, we reuse and extend the big data taxonomy introduced by the Cloud Security Alliance (Murthy et al., 2014). Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).

Table 13: Extensions with respect to the *Big-Data Model/ Data Management classes*

Class Taxonomy Levels				Properties	Description and Related Ontology (if any)
Data Mana-gement					This class encapsulates all the relevant concepts that can be used in order to describe major technological choices with respect to how big data is acquired, stored, processed, transferred or replicated for redundancy reasons.
				hasData Timestamp	This object property associates the Data Management class with the Data Timestamp class of the Big Data model in order to express the time when certain data artefacts where acquired, processed or transferred.
				hasAgent	This object property associates the Data Management class with the Subject class of the Context Aware Security model in order to express the responsible entity for performing data acquisition, processing transferring and storage.
	Acquisi-tion				This subclass is used in order to describe the required or offered types of big data acquisition in the frame of a Morphemic-enabled cloud application devised to process it.
				isReliable	This data property associates the Acquisition class with a boolean value that captures whether or not the means of data acquisition required or offered guarantee the accuracy of the data received.
				buffersMessages	This data property associates the Acquisition class with a boolean value that expresses the capability of a Morphemic-enabled application to resolve any bottlenecks by buffering the surplus data, in cases that the data acquisition rates are larger than the processing rates.
				applies Backpressure	This property associates the Acquisition class with a boolean value that denotes the capability of interrupting the data source transmission in cases that the receiver and its buffers are not able to receive additional data for a short period of time.
				dropsMessages	This property associates the Acquisition class with a boolean

					value that refers to bottleneck situations being resolved by dropping any surplus data.
		Source			This subclass refers to aspects of the data source that characterize the data model and the nature of data to be digested by a Morphemic-enabled application.
				isBatch	This Boolean property is used to define the nature of data artefacts to be stored and processed in batch mode by a Morphemic-enabled application.
				isStream	This Boolean property is used to define the nature of data artefacts to be stream processed by a Morphemic-enabled application.
			Structured		This subclass refers to a data source that produces data artefacts comprised of clearly defined data types whose pattern makes them easily searchable
			Unstructured		This subclass refers to a data source that produces data artefacts who do not follow a structured data model and is usually difficult to search (e.g. audio source, video source etc.)
			Semi-structured		This subclass refers to a data source that produces data artefacts that although they do not follow a strict data model, they contain semantic tags (e.g. XML source).
		Pull-based			This subclass aims to capture concepts related to the pull-based paradigm for acquiring data, where there is a request for triggering the transmission of data which is initiated by the Morphemic-enabled application, i.e. the receiver entity (Yang et al. 2017) and the receipt takes place in a synchronous manner.
		Push-based			This subclass aims to capture concepts related to the push-based paradigm for acquiring data asynchronously, where the request for a given transaction is initiated by the publisher (Yang et al. 2017).
				isOrdered	This property associates the Push-based class with a boolean value that states whether or not the certain push-based technique used for relaying big data, guarantees the time ordering of the received data before their processing takes place.
				uses Acknowledg-ments	This property associates the Push-based class with a boolean value that defines whether or not the data source will repeatedly attempt to re-send the data to the Morphemic-enabled application until a receipt confirmation message is sent.

			Message Brokering		This subclass refers to a certain type of push-based acquisition of data where an intermediary software component undertakes the task of translating and rooting data transparently to any given number of subscribed receivers (Morphemic-enabled applications) that expect the acquisition of certain data in a pre-defined format (Hohpe & Woolf, 2004).
			Centralized		This is a subclass of the Message Brokering class where the push-based paradigm, for communicating data between producers and subscribers, is implemented with a central broker application, usually called enterprise service bus – ESB (Chappell, 2004). **Instances:** WSO2 ESB[20], jBoss ESB[21], Mule ESB[22], Apache Servicemix[23], Apache Camel[24], Open Studio for ESB[25], Google Cloud Pub/Sub[26], IBM MQ[27], Azure Service Bus[28], RabbitMQ[29], Apache ActiveMQ[30].
			Distributed		This is a subclass of the Message Brokering class where the push-based paradigm for communicating data uses several dispersed, but integrated software applications (also called distributed ESB) with message brokering capabilities, instead of just one centralised broker entity in order to avoid any performance bottlenecks. **Instances:** Apache Kafka[31], Petals ESB[32]
			Brokerless Messaging		This subclass refers to a certain type of push-based acquisition of data where there are not intermediaries in the middle for translating and rooting data, instead direct peer-to-peer communication between the data sources and the receivers (i.e. Morphemic-enabled application) is considered for low latency and/or high transaction rate applications (ZeroMQ, 2008).

[20] http://wso2.com/products/enterprise-service-bus/
[21] http://jbossesb.jboss.org/
[22] https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb
[23] http://servicemix.apache.org/
[24] https://camel.apache.org/
[25] https://www.talend.com/products/application-integration/esb-open-studio/
[26] https://cloud.google.com/pubsub/
[27] https://www.ibm.com/products/mq
[28] https://azure.microsoft.com/en-us/services/service-bus/
[29] https://www.rabbitmq.com/
[30] https://activemq.apache.org/
[31] https://kafka.apache.org/
[32] https://petals.linagora.com/

			UDP Multicast		This is a subclass of the Brokerless Messaging class and refers to the simultaneous group communication (multicast) using the User Datagram Protocol –UDP (Kurose & Ross, 2010). **Instances:** StatsD[33], Brubeck[34].
			TCP/IP Multicast		This is a subclass of the Brokerless Messaging class and refers to the technique for one-to-many communication over an IP infrastructure in a network. (Kurose & Ross, 2010). **Instances:** ZeroMQ[35], MQTT[36].
	Data Storage				This subclass encapsulates all the concepts that can be used for characterising the way that input or output data should be stored. The hierarchy involved updates the storage infrastructure taxonomies that (Murthy et al., 2014) and (Mazumdar et al., 2019) presented.
		Database Manage-ment Systems			This class refers to data storage approaches applied through relational or non-relational systems.
				offersDataGrid	This Boolean property denotes the capability of a database system to provide access to extremely large amounts of geographically distributed data.
			Relational Storage		This subclass refers to databases used for persisting data that are structured in a way that capture and present relations between stored data artefacts (Codd, 1970).
				inMemory Relational	This Boolean property characterises a relational database management system that primarily relies on (cloud) hosting resource's main memory for persisting data instead of employing a disk storage mechanism. For example H2[37], HSQLDB[38], MemSQL[39], and SQLite[40] can be used as in memory relational database systems.
			RDBMS		This subclass of Relational class, groups all the traditional SQL-based database management systems. **Instances:** Microsoft SQL Server[41], MySQL[42], Oracle[43], PostgreSQL[44],

[33] https://github.com/etsy/statsd
[34] https://githubengineering.com/brubeck/
[35] http://zeromq.org/
[36] http://mqtt.org/
[37] http://h2database.com/html/main.html
[38] http://hsqldb.org/
[39] https://www.memsql.com/software/
[40] https://sqlite.org/
[41] https://www.microsoft.com/en-us/sql-server/sql-server-downloads
[42] https://www.mysql.com/

					IBM DB2[45], H2, HSQLDB, MemSQL, SQLite.
			NewSQL		This subclass refers to a type of parallel database management systems that provides the same scalable performance of non-relational systems while still maintaining the same level of transactional support (i.e. support the properties of Atomicity, Consistency, Isolation, and Durability – ACID) as the traditional relational databases (Murthy et al., 2014). **Instances:** VoltDB[46], H-Store[47], NuoDB[48], CockroachDB[49].
			Non-Relational Storage		This subclass refers to databases (also called NoSQL) used for persisting data that are not modelled using tabular relations and present certain advantages over the relational databases, especially for big data since they offer design simplicity and more efficient horizontal scaling.
				inMemoryNon Relational	This Boolean property characterises a non-relational database management system that primarily relies on (cloud) hosting resource's main memory for persisting data instead of employing a disk storage mechanism. For example Aerospike[50], Redis[51], ArangoDB[52], Memcached[53] and Hazelcast[54] can be used as in memory non-relational database systems.
			Key-Value		This subclass of Non-Relational Storage refers to a non-relational data storage paradigm designed for storing, retrieving, and managing associative arrays based on keys. These associative arrays contain a collection of objects with many different fields within them (Tweed & James, 2010). **Instances:** Riak[55], Redis, LevelDB[56], Voldemort[57], Memcached, Aerospike, Hazelcast.

43 https://www.oracle.com/database/technologies/appdev/sqldeveloper-landing.html
44 https://www.postgresql.org/
45 https://www.ibm.com/analytics/db2
46 https://www.voltdb.com/
47 http://hstore.cs.brown.edu/
48 https://nuodb.com/
49 https://www.cockroachlabs.com/product/
50 http://www.aerospike.com/
51 https://redis.io/
52 https://www.arangodb.com/
53 https://memcached.org/
54 https://hazelcast.org/
55 https://riak.com/
56 https://github.com/google/leveldb

			Document DB		This subclass refers to a software application designed for performing CRUD operations over semi-structured data called document-oriented information. **Instances:** MongoDB[58], NosDB[59].
			Wide-Column		This subclass involves compressed, high performance databases that follow a column-oriented data model that does not rely on a fixed schema. Specifically, they provide nestable, map-like structures for data items which improve flexibility over fixed schema (Chang et al. 2008). **Instances:** Bigtable[60], HBase[61], Cassandra[62].
			Time-Series		This subclass refers to databases that are optimized for storing and serving series of data points indexed in time order, through associated pairs of time(s) and value(s) (Mueen et al., 2009). **Instances: Prometheus[63], InfluxDB[64], Axibase[65], OpenTSDB[66]**
			GraphDB		This subclass refers to databases that use graph structures (with nodes, edges and properties) for storing and retrieving data. Some are implemented adopting the relational paradigm by storing the graph data in a table while others use a key-value store or document-oriented database for storage (Angles & Gutierrez, 2008). **Instances:** Neo4j[67], OnyxDB[68], Titan[69].
			Multimodel DB		This subclass refers to database management systems that support multiple data models (e.g. document, graph, relational, and key-value models) in one integrated backend (Lu & Holubová, 2017). **Instances:** ArangoDB, OrientDB[70], DynamoDB[71]
		File			This class groups different systems

57 http://www.project-voldemort.com/voldemort/
58 https://www.mongodb.com/
59 https://www.npmjs.com/package/nosdb
60 https://cloud.google.com/bigtable/
61 https://hbase.apache.org/
62 http://cassandra.apache.org/
63 https://prometheus.io/
64 https://www.influxdata.com/
65 https://axibase.com/
66 http://opentsdb.net/
67 https://neo4j.com/
68 https://www.onyxdevtools.com/
69 http://titan.thinkaurelius.com/
70 http://orientdb.com/orientdb/
71 https://aws.amazon.com/dynamodb/

		Systems			used for organising the way data can be placed, retrieved and modified in and from a computer storage medium. They can be local in computer system or distributed.
			Local File System		This subclass involves subsystems available in all operating systems that allow the non-scalable persistence, retrieval and update of information in the hard drives of a computer.
			Distributed File System		This subclass refers to an extended networked file system that allows multiple distributed nodes to internally share data or files (Levy & Silberschatz, 1990). This system provides scalability, fault-tolerance, concurrent file access and metadata support.
				Client-Server	This subclass of the Distributed File System class refers to a client-server architecture based file system in which all communications between servers and clients are conducted via remote procedure calls (Mazumdar et al., 2019). **Instances: GlusterFS[72], Lustre[73]**
				Clustered Distributed	This subclass of the Distributed File System class refers to a fault-tolerant file system that offers multiple nodes to enable concurrent access to the same block device. (Mazumdar et al., 2019) **Instances: HDFS[74], CephFS[75], iRODS[76], MooseFS[77].**
				Symmetric	This subclass of the Distributed File System class refers to systems with masterless architectures, employing a distributed hash table approach for data distribution and replication across systems (Mazumdar et al., 2019). **Instances:** PVFS[78], Ivy (Muthitacharoen et al., 2002), Red Hat GFS[79]
	Processing				This subclass encapsulates all the concepts that can be used for describing and classifying the various types of big data processing that can be conducted by a Morphemic-enabled cloud application. The hierarchy introduced updates both the DICE

[72] https://www.gluster.org/
[73] https://opensfs.org/lustre/
[74] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[75] https://docs.ceph.com/en/latest/cephfs/
[76] https://github.com/irods/irods
[77] https://moosefs.com/
[78] https://web.archive.org/web/20160701052501/http://www.pvfs.org/
[79] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/4/html-single/global_file_system/index

					model for big data intensive application (Gómez et al., 2016) and the computer infrastructure taxonomy presented by CSA Big Data Taxonomy (Murthy et al., 2014). Note: the details of this subclass have not been changed, so we do not mention all its subclasses and properties here. All the details are available in (Verginadis et al., 2017).
	Transfer				This subclass of Data Management class refers to any concept that can be used for describing aspects related to communicating data artefacts between their data sources and the processing or storing locations.
				hasData TransferCost	This property associates the Transfer class with a float that denotes the actual or expected cost for transferring data.
				hasDataTransfer Duration	This property associates the Transfer class with a float that denotes the time needed for transferring data between different locations.
				hasTransfer Origin	This property associates the Transfer class with the Data Location class of the Big Data Model in order to identify the source location of a data-transferring task.
				hasTransfer Target	This property associates the Transfer class with the Data Location class of the Big Data Model in order to identify the sink location of a data-transferring task.
				hasDataTransfer DesiredStart Time	This property associates the Transfer class with a date datatype in order to define the desired start time of a data-transferring task.
				hasDataTransfer Desired CompletionTime	This property associates the Transfer class with a date datatype in order to define the desired end time of a data-transferring task.
				hasData Migration Security Constraint	This property associates the Transfer class with the Security Context Element class in order to list DateTime, Location and Connectivity related constraints during data migration.
	Redun-dancy				This subclass encapsulates any approach used for persisting the same data artefacts in several separate places, either in a single database, or in remote databases for detecting and reconstructing lost or damaged data (Doorn & River, 2002). Note: the details of this subclass have not been changed, so we do not mention all its subclasses and

					properties here. All the details are available in (Verginadis et al., 2017).