Resource Requirements

Software application components (see *SoftwareComponent* class) in CAMEL have their deployment regulated through their association to a set of requirements (see *RequirementSet* class). Such a set includes requirements of different types, like location, provider, OS and resource requirements. The latter form of requirements (see *Resource Requirement* class) is essential for both the filtering of the provider/offering space as well as the actual deployment reasoning. It maps to the supply of requirements over infrastructural resources like VMs.

In the previous version of CAMEL, resource requirements were supplied in the form of constraints over fixed parameters mapping to the respective VM characteristics. In particular, only the VM's CPU frequency and number of cores, main memory size and storage size were considered and the constraints posed on them could include both upper and lower bounds on their values. For instance, in order to pose a constraint over the minimum and maximum number of cores, the following requirement specification could be specified in the textual syntax of the previous version of CAMEL:

```
quantitative hardware CoreIntensive {

    core: 8..32

}
```

However, the previous form of requirements posing was too restrictive and did not enable to specify constraints over additional characteristics of a VM or its components. It could also not enable to specify constraints over other kinds of infrastructural components, like Containers. In this respect, in order to extend the current way CAMEL enables the specification of resource requirements, the mechanisms employed for extending CAMEL in order to properly capture the data aspect were exploited. This means that instead of respective values that can be supplied for fixed VM characteristics, now the modeller can specify attributes which refer to the meta-data schema and which are mapped to various kinds of infrastructural components. In addition, in order to distinguish between those attributes that map to the same entity (e.g., VM), these attributes can be grouped into the notion of a feature. In other words, we can group attributes via features and such attributes specify constraints over the respective entities represented by that features like infrastructural entities. Based on the previous example, it is better now to showcase how the respective resource requirement could be specified via the textual syntax of the new version of CAMEL:

```
resource requirement ResourceReq{

feature cpu{

    [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU]

    attribute mincores [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU.
hasMinNumberofCores] : double 8.0 UnitTemplateCamelModel.UnitTemplateModel.Cores

    attribute maxcores [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU.
hasMaxNumberofCores] : double 32.0 UnitTemplateCamelModel.UnitTemplateModel.Cores

  }
}
```

In this resource requirement specification, we have the feature of CPU, i.e., of a certain sub-component of an IaaS service/offering. This can be understood from the annotation of that feature which points first to the IaaS concept in the meta-data schema and finally to the CPU concept. From the annotation, we can actually see the whole branch of the meta-data schema that points to the right notion/concept that should be used for the annotation. In this feature, we have specified two attributes which have the same unit mapping to the number of cores which has been drawn from the UnitTemplate CAMEL model (see examples directory in CAMEL's git repository), which is a CAMEL model in which we have included some unit templates that can be reused for the specification of various CAMEL elements, including attributes. These two attributes actually represent the minimum and maximum bounds over the number of cores characteristic of the CPU infrastructural component. In this respect, by supplying values to these attributes, it is like providing constraints over the range of values that the respective CPU characteristic can take.

Based on the above example, it is apparent that every characteristic of an infrastructural component should be specified in the form of two attributes which represent the two bounds that this characteristic can take. As such, it is now imperative to explicate which are those infrastructural entities and respective characteristics of them which can be exploited by the modellers in their specification of the resource requirements of their software components. To this end, we need to have a walkthrough over the content of the related Meta-data schema part (*Application Placement Model* and especially *IaaS* concept's sub-hierarchy). In such a walkthrough, apart from marking down both the infrastructural entity and its characteristics, we also explicate the possibility that such elements is meaningful to be exploited for the filtering of the provider space as well as the application deployment reasoning. In this respect, we attempt to make a best effort in enumerating all these elements as well as denoting their actual appropriateness in the Melodic platform based on its current development status, its main goals as well as the current setting in the cloud world (e.g., what can be actually represented/advertised by the cloud providers in terms of their infrastructural offerings). Please note that characteristics/attributes mapping to metrics have been intentionally left out from the analysis as they would require to be supplied in SLOs instead.

Feature	Attribute(s)	Filt.	Reas.	Notes	Upperware	Cloudiator

CPU	hasMinNumberofCores, hasMaxNumberofCores		~	Need to have sth like total number of cores to potentially use in opt. objectives. But logically speaking, should be involved in the CP model.	SUPPORTED (used in resource requirements and then to filter the Node Candidates)	
	hasFrequency		-	Better to have max and min attributes for the filtering		
	hasMFLOPs, hasMIPs	~	~	Need to get this from manufacturer of CPU. This is not provided by the cloud provider normally.		
	hasManufacturer*	-	-	Questionable if every cloud provider literally provides this for every offering, if not at all. Further, not sure whether this will be exploited by users, esp. for public clouds.		
				Not the attribute, but the feature itself (CPU) is used to indicate the type of the variable.	SUPPORTED (type of the Constraint Problem Variable - CORES used in the reasoning process)	
RAM. TotalMemory	totalMemoryHasMin, totalMemoryHasMax		~	Same as in number of cores.	SUPPORTED (used in resource requirements and then to filter the Node Candidates)	
	totaMemoryHasUnit	-	-	Not needed as the unit is provided in the attribute specification in CAMEL		
RAM	RAMhasManufacturer*	-	-	Same as in case of CPU		
				Not the attribute, but the feature itself (RAM) is used to indicate the type of the variable.	SUPPORTED (type of the Constraint Problem Variable - RAM used in the reasoning process)	
GPU	hasMemoryBandwidth	~	-	Questionnable if a provider advertises this. Should get GPU's model name and then search manufacture's site for this.		
	hasWarpSize	-	-	Not clear if this is advertised by manufacturers		
	GPUhasMFLOPs	~	~	Same as in CPU. Need to get this from manufacturer		
	hasMaxConcurrentWorkGroups	-	-	Not clear if this is advertised by manufacturers		
	hasClockSpeed	~	-	Could be used for filtering but not usually supplied by the cloud provider. Further, we need min and max values here.		
	GPUhasManufacturer*	-	-	Same as in case of CPU		
	hasPEperCUs	-	-	Not clear if this is advertised by manufacturers		
	GPUhasMaxNumberofCores, GPUhasMinNumberofCores		~	Same as in case of CPU		
Processing	hasPowerConsumption	~	~	This is valid for all processing elements. However, not clear whether this is advertised by cloud providers. Need to be obtained from manufacturer. More meaningful though for private clouds		
Storage	hasSolidStateDrive		-	Logically speaking, most cloud providers indicate this		
	hasWriteThroughput, hasReadThroughput	~	-	Not clear if this is advertised by cloud provider		
	isPersistent		-	Logically, this is indicated by a cloud provider. Question is if there is a way to retrieve this automatically, possibly from the type of the storage.		
				Not the attribute, but the feature itself (Storage) is used to indicate the type of the variable.	SUPPORTED (type of the Constraint Problem Variable - STORAGE used in the reasoning process)	
Storage. Capacity	hasMin, hasMax		~	Questionnable if we need to have an optimisation objective on this. But it should participate in the CP model.	SUPPORTED (used in resource requirements and then to filter the Node Candidates)	
	hasUnit	-	-	This is specified in the attribute specification in CAMEL		
On-Instance Storage, Off-Instance Storage		~	-	Better to have here a storageType attribute in Storage class which could map to these classes or to their sub-classes		
Network	hasBandwidth		-	Can be included also in the CP model but not clear if it can participate in optimisation objective. Possibly also we need min and max attributes for it		

IPType, Version(*)			-	Could be used for filtering but we need to define respective attributes for them. Potentially, we could think of whether we need to extend the members/sub-concepts of these concepts or we should have a N multiplicity of the respective attributes		

A special remark needs to be supplied here for those components which map/manage data sources. For such components, apart from CPU/RAM requirements, we definitely need to supply also Storage-based requirements. We have designated more or less the attributes relevant for the latter requirements. It is be now appropriate to check which from those attributes can be really supported by the platform and how. The same question, of course, goes to other infrastructural components and their attributes. The consortium's technical experts need to join forces in order to decide about this.

* Attributes marked with the asterisk seem to have N multiplicity. In CAMEL, this could be supported in two possible ways: (a) provide all the values in the form of a single string, e.g., "val1,val2, ..., valN"; (b) provide the attribute as many times as the values that are needed. I would prefer the first choice which seems more neat and leads to less lengthier models. Anyway, both ways are possible in CAMEL's textual (and abstract) syntax. So, here there is also a small technical decision to take.