Requirements Modelling

CAMEL 2.0 enables the modelling of various types of requirements. Such requirements can be either soft or hard. Soft requirements can be seen as directives towards the platform in order to satisfy them in a best-effort basis. On the other hand, hard requirements need to be satisfied by the platform at all cost. All the deployment requirements, related to the deployment of one or more application components, that can be specified via CAMEL are hard. On the other hand, CAMEL can be used to also specify non-functional requirements, which can be either soft or hard. Usually, only one soft non-functional requirement can be specified mapping to the utility function whose values need to be optimised. While multiple hard non-functional requirements, related to the performance or security of application components or the whole application, can be specified in terms of Service Level Objectives (SLOs) and security control constraints. All application requirements need to be included in one overall model which is called a requirement model, acting as a single container of these requirements. This model in turn should be included within the CAMEL model of the respective application.

For instance, by relying of the CAMEL's textual syntax, a requirement model will have the following form:

```
requirement model reqModelName{
        resource requirement resReq1{
                ...
        }
        ...
        resource requirement resReqN{
                ...
        }
        ...             here we could have other kinds of deployment requirements like OS or platform ones
        slo SLO1 ...
        ...
        slo SLON ...
        ...             here we could also have security (control) requirements
        optimisation requirement OptReq{        only one optimisation requirement is meaningful to be specified
                ...
        }
}
```

In the following, we analyse all kinds of requirements that can be specified via CAMEL. To assist the better comprehension of this analysis, fragments of respective CAMEL textual syntax specifications of the related requirements in terms of Melodic use cases are specified.

Hard Requirements

Deployment Requirements

Resource Requirements

Resource requirements can be considered as constraints over the resources that can be used to host application components. Such resources can be either VMs or containers or the respective sub-resources (e.g., cores) that VMs or containers might comprise. The constraints on these resources are applied over their characteristics which can be drawn from the meta-data schema (MDS). For instance, if we desire to specify constraints over the minimum and maximum number of cores of a resource, we should specify respective attributes in CAMEL 2.0 which relate to these characteristics (i.e., hasMinNumberofCores and hasMaxNumberofCores) and the values that they should take. The following fragment shows the resource requirements that have been specified in the CRM use case for the SmartDesign component:

```
resource requirement SmartDesignResourceReqs{

    feature cpu{  we indicate here that we specify a group of constraints for the CPU sub-resource

        [ MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU ]  this is annotation from the
        MDS pointing to the CPU sub-resource

        attribute mincores  this is the actual constraint over the min number of cores of the CPU sub-resource

            [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.CPU.hasMinNumberofCores] :
            this is the annotation pointing to the respective characteristic in the MDS
            double 2.0 UnitTemplateCamelModel.UnitTemplateModel.Cores  this defines the value of the characteristic and the
            respective unit of measurement

        attribute minram  this is the constraint over the min RAM size of the CPU sub-resource

            [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.RAM.TotalMemory.
            totalMemoryHasMin] :  this is the annotation pointing to the respective characteristic in the MDS
            int 8192 UnitTemplateCamelModel.UnitTemplateModel.MegaBytes  this defines the value of the characteristic and the
            respective unit of measurement

    }

}
```

In the above fragment, we specify that the CPU of a resource should have at least 2 number of cores and at least 8192 MBs of RAM.

More details about how resource requirements can be modelled are given in a sub-page of this page.

Platform Requirements

Platform requirements can be regarded as constraints over the characteristics of PaaS services or serverless platforms or of their sub-components. Such requirements are actually specified in exactly the same way as resource requirements. That is we specify a constraint in the form of a mapping between a characteristic and the value that it can take. Such a constraint is specified as a CAMEL attribute. When attributes refer to the same PaaS / serverless component, they can be grouped over that component via a CAMEL feature. The following CAMEL textual fragment refers to the platform requirements concerning an imaginery use case over a serverless application:

```
paas requirement ServerlessCompPaasReq{

feature environment{  we indicate here that we specify a group of constraints over the environment of the PaaS / serverless platform

        [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.CloudService.PaaS.Environment]  this is a pointer to
        the respective environment element in the MDS

        attribute runtime [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.CloudService.PaaS.Environment.
        Runtime] : string "JAVA"  we specify that the environment's runtime should be Java

        attribute MinVersion [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.CloudService.PaaS.Environment.
        Runtime.minRuntimeVersion] : string "1.8"  we specify that the minimum runtime version should be 1.8

    }

    feature limits{  we indicate here that we specify a group of constraints over certain limited characteristics of serverless platforms

        [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.CloudService.PaaS.Serverless.Limits]  this is a
        pointer to the respective environment element in the MDS

        attribute maxExecTime [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.CloudService.PaaS.Serverless.
        Limits.maxDuration] : int 600 UnitTemplateCamelModel.UnitTemplateModel.Seconds  here we specify that the maximum
        execution time should be 600 seconds for the serverless component that we need to deploy

        attribute MaxHttpReqSize [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.CloudService.PaaS.
        Serverless.Limits.maxHTTPRequestSize] : int 10 UnitTemplateCamelModel.UnitTemplateModel.MegaBytes  here we specify
        that the maximum HTTP request size for the serverless component should be 10 MB

    }

    attribute CostGB_Sec [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.CloudService.PaaS.Serverless.Cost.
    costGB_Second] : double 0.0000025 UnitTemplateCamelModel.UnitTemplateModel.Dollars  here we specify a constraint over a cost
    component of a serverless platform. In particular, we indicate that the cost for GB-seconds should be 0.0000025

}
```

In the above fragment we specify PaaS / serverless platform requirements for one serverless components. In particular, we indicate that the runtime for that component should be at least JAVA 1.8. We also specify that the component should be given 600 seconds to execute and that its HTTP request size should reach 10 MBs. Finally, we indicate that the cost for deploying and running that component in the serverless platform in terms of GB-secs should be 0.0000025 dollars.

Operating System (OS) Requirements

OS requirements can be used to restrain the OS of the VM / container that hosts an application component. Currently, such a requirement can specify the actual OS to be used as well as whether it is 64-bit or not. The following fragment from the CRM use case defines a requirement for an ubuntu 64-bit OS:

```
os requirement UbuntuReq{

        os 'ubuntu'  here we specify the OS needed

        64os  here we specify that is 64-bit. Absence of this indicates a negative statement, i.e., the OS should not be 64-bit

}
```

Image Requirements

Image requirements can be used to restrain the image of the VM that hosts an application component. Currently, such a requirement can specify a set of VM images pertaining to an application component mapping to different cloud providers in the context of multi-cloud application deployment. To be noted that this is a kind of implicit provider requirement in the sense that the provider space is restrained only to include those providers for which the respective images referenced by this requirement have been defined. The following fragment showcases the form that such a requirement can take:

```
image requirement ImageReq{

        images ['Img1', 'Img2']  here we specify the ids of the images that can be used for the application component deployment

}
```

Location Requirements

A location requirement indicates a constraint over the placement location of an application component. Such a constraint is actually specified through the identification of all the locations that are allowed for the component's placement. Locations can be either physical (either continents, sub-continents or countries) or cloud-based (e.g., a specific region in a cloud provider). Location requirements are mainly meaningful to be provided for both data management/storage and processing components. Data management / storage components encapsulate and manage data sources. As such, the respective constraints actually restrains the location of such data sources. On the other hand, data processing components perform computations over data. Via location requirements over them we then specify constraints over the location of processing of the respective data.

It should be noted that the list of locations supplied in such a constraint can map to a conjunctive or a disjunctive requirement. By default, we deal with a disjunctive requirement which indicates that one of the locations is needed for the hosting of the respective component. However, if the modeller specifies also the value of "allRequired" in the location requirement, this requirement then becomes conjunctive meaning that at least one instance of the respective component should be deployed in all the locations specified. This latter requirement kind can be quite useful in case that an application needs to be deployed in multiple locations and not just one to be closer to its clients and the way they are partitioned across the world.

The following fragment specifies a location constraint over the Hive component of the PeopleFlow use case:

```
location requirement CzReq [Locations.CZ]  here we have a pointer to a location that is specified in the Locations location model which maps to the country of Czech Republic
```

From the above fragment, we see that a reference to only one location is supplied as the data managed by the Hive component are sensitive and need to be stored only within the country of Czech Republic.

It should be noted that a template location model has been generated from the United Nations' Food and Agriculture Organisation (FAO) geopolitical ontology comprising a three-level hierarchy of physical locations (continents, sub-continents and countries). This location model can be thus re-used for the specification of location requirements. It can be supported by the textual CAMEL editor, if it is placed in the respective workspace, and by the CAMEL's web editor, if it is imported in the CDO model repository.

Provider Requirements

A provider requirement indicates a constraint over the providers that can be used for the hosting of one or more application components. Such a constraint can be specified through identifying the list of providers that are required, supplied in the form of String names (e.g., "Amazon"). In addition, the user can specify whether the type of the cloud offered by the providers should be private or public. The latter specification can be also used in isolation and could be combined via a location constraint. For instance, we can specify that we need a private cloud and such a cloud should be situated in Germany. This will then map to the selection of German cloud providers like ProfitBricks. Alternatively, if only ProfitBricks is needed, then the modeller could just specify the name of that provider without providing the respective cloud type.

The following fragment specifies the provider requirement for the SparkOnDemand component of the PeopleFlow use-case:

provider requirement EUProvs [Amazon, Google, ProfitBricks] here we specify that the component deployment should be restrained over these 3 cloud providers

This component is responsible for processing the sensitive data stored in Czech Republic. The processing of such data should be performed only in Europe. To this end, the modeller specifies here providers which do have data centres in Europe. To also guarantee that the processing will definitely stay in Europe, the modeller will have to specify a location requirement for the SparkOnDemand component. This would not be actually required if only ProfitBricks was supplied as this provider has only data centres in Europe.

Horizontal Scale Requirements

Horizontal scale requirements restrain the number of instances that can be created for a certain component within a certain range. Such a range can go to positive infinity, if its upper bound equals to -1. Such constraints can be regarded also as scaling policies as they restrain the way a certain component can scale. In the following, we depict a CAMEL textual fragment mapping to an horizontal scale requirement for the SmartDesign component of the CRM use case:

horizontal scale requirement SmartDesignHReq [2,6] here we specify that the number of instances of the component should be from 2 to 6

The above requirement indicates that the number of instances of that component should be in the [2,6] range. Please note that the lower bound indicates that at any moment during the execution of the respective application, the SmartDesign component should not have less than 2 instances being deployed. However, if 2 was replaced with 0, this could mean that the modeller allows that component to not have any instances at a particular moment in time or time period. This could be useful, for example, in case of components which need to be conditionally deployed for an application, like load balancers.

Non-Functional Requirements

Service Level Objectives (SLOs)

SLOs are constraints over non-functional terms, which in our case are metrics or metric variables. In the first case, such constraints can be imposed over the performance of an application. For instance, we could specify that the average response time of the application should be at most 2 minutes. In the second case, such constraints can be imposed over the characteristics of the application configuration. For instance, we could specify a constraint that signifies that the sum of the number of instances of two application components should not be greater than 10. To formulate an SLO, the modeller needs to just refer to the respective constraint element in CAMEL that specifies the respective constraint. For instance, please see the next fragment mapping to a constraint over the FinishTimeMargin metric for the TrafficSimulation use case:

slo SimulationOnTime constraint TrafficSimConstrModel.FinishTimeMarginCondition here we specify the name of the SLO and a reference to the respective constraint that should hold

The respective constraint referenced by this SLO indicates that the finish time margin (the difference between the expected and the required execution time) of the application should be negative or equal to zero.

This is clarified in the next fragment:

metric constraint FinishTimeMarginCondition: [TrafficSimMetrModel.FinishTimeMarginContext] <= 0.0

More details about how constraints can be modelled in CAMEL 2.0 are supplied in: Modelling of Utility Function and Constraints in Camel Model 2.0

Security Requirements

Security requirements are constraints over security controls which need to be supported by the clouds on which one or more application components should be deployed. Such security controls represent high-level security mechanisms that should have been realised by these cloud providers. An extensive list of security controls has been defined in the Cloud Control Matrix (CCM) of the Cloud Security Alliance. This list has been transformed into a security model in CAMEL from which the required security controls for an application can be picked up. This template model can be copied in the workspace of the CAMEL's textual editor and it can be also imported in the CDO model repository for being re-used by the CAMEL web-based editor. The following fragment supplies a security requirement for the Hive application component in the PeopleFlow use case:

```
security requirement SecReq{

    security controls [Basic_Security_Model.DCS_08, Basic_SecurityModel.EKM_03]  here we provider a reference to all the security controls that need to be realised

}
```

In the above fragment, two security controls need to be realised by the cloud providers that should host the Hive component: (a) DCS_08: this is related to the monitoring and control of unauthorised personnel in ingress or egress areas around data storage facilities; (b) EKM_03: this is related to the establishment of policies, procedures and business processes for the use of appropriate encryption mechanisms / protocols for the protection of data. Both these security controls are referenced from the Basic_Security_Model security model that has been generated from CCM.

Soft Requirements

Optimisation Requirements

Optimisation requirements are soft requirements over a metric variable that represents a utility function. Such a requirement indicates that the platform should attempt to optimise (either maximise or minimise) the utility function in a best-effort basis. The way the utility function can be specified is explained in the following page in Confluence: Modelling of Utility Function and Constraints in Camel Model 2.0. In principle, a utility function is a mathematical expression that can take values from the domain [0.0,1.0] of real values, where, normally speaking, the value of 0.0 indicates the least preferred value and 1.0 the most preferred one. Such a function can be specified over one or more metric variables or metrics. In this respect, it maps to one objective which could be derived from multiple ones by following a method such as the Simple Additive Weighting (SAW) one. In the following, we supply a fragment depicting the optimisation requirement for the CRM use case:

```
optimisation requirement maxCostUtility{

variable CRMMetricModel.CostUtility  here we refer to the metric variable to be optimised

}
```

In this fragment, we refer to a metric variable that specifies a utility function over the SmartDesign component's cost of the CRM use case. We do not indicate whether this function should be minimised or maximised as by default it is assumed to be maximised, which perfectly matches the direction of values for this function. In case that this function would require to be minimised, we would have to add the "minimise" clause/statement before the variable reference.