Data Modelling

To support the modelling of the data aspect, various changes have been performed in CAMEL which also led to an update with respect to its design philosophy. All these changes are reported in the following confluence page: Changes in CAMEL to support Data Modelling. In this page, we will shortly analyse how a data model in CAMEL can be specified.

Two types of data models exist, a data type and instance model. This indicates that the well-known type-instance pattern is followed, where the type model provides for the types of elements while the instance model cater for the instances of these types. Both kinds of data models act as containers of data-related elements. A data type model contains two kinds of elements, data sources and data (sets). A data source is a means of encapsulating data. Further, one data source can encapsulate more than one data (set). A data source can be either internal or external. An external data source is managed by a component which is external to the platform, i.e., the platform cannot control it. However, such an external data source can be exploited by a user application primarly as a kind of an input data source. However, we do not exclude the case where this data source can become the place where the data produced from the application execution are stored. On the other hard, an internal data source maps to a (data) application component that is responsible for its management. Such a component could be, for example, a database (server). In the following, we supply an example of two data sources, one internal and one external, originating from the PeopleFlow use case. These data sources are modelled according to the following CAMEL textual fragment:

```
data source S3DataSource{

    [MetaDataModel.MELODICMetadataSchema.ApplicationPlacementModel.IaaS.Processing.Storage.Off_InstanceStorage.S3]
    annotation indicating that we are dealing with an S3 storage

    external  this indicates that we are dealing with an external S3 data source

}

data source HiveDS{

    component PeopleFlowDepModel.Hive  here we deal with an internal data source managed by the Hive application component
    referenced

}
```

The first data source is external and maps to an S3 storage (as given by the annotation). To indicate that it is external, we supply the "external" keyword. The second data source is internal and managed by the Hive data component of the PeopleFlow application. Please note here that we deal with data source types. In other words, the S3 data source, for example, could then map to multiple data source instances which might be located in different locations of the Amazon cloud.

As can be understood, data sources can be annotated from the meta-data schema (MDS). In fact, the same can hold also for the data themselves. In addition, both data and data sources can also be described via features and attributes mapping to the use of the extension mechanism of CAMEL (see more details in Changes in CAMEL to support Data Modelling). The features can be considered as kinds of groupings of attributes that refer to a certain notion. While the attributes map directly to some characteristics of these notions that can be regarded as certain aspects of the respective data or data sources. We will come back to this below by supplying a certain example on how to specify data.

The notion of Data in CAMEL represents a certain data set type. Such a dataset type is always encapsulated by a certain data source (type). In addition, a data set can also include other data sets. This could be quite useful for the modelling of composite data sets comprising data set pieces that can be managed separately by the platform. A component dataset does not require in this way to refer to its data source as this is already done by its parent data set. A data set can be characterised by certain attributes or features while it could be also annotated by the meta-data schema. For instance, we could specify that a certain dataset has an (input) velocity of 50 GB per day. As another example, a dataset could be have a tabular form of key-value pairs. Both of these examples are showcased in the following CAMEL fragment drawn from the PeopleFlow application:

```
data TableData{

    [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataVariety.Format.Key_ValuePairs]  annotation
    that the table-based data set is in form of key-value pairs

}

data StreamData {

    source S3DataSource  the stream data come from the S3DataSource

    feature InputVelocity{

        [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataDensity.Velocity]  this is the velocity
        notion from the meta-data schema

        attribute isInput
        [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataDensity.Velocity.isInputVelocity] :
        boolean true  we deal with an input velocity for the data

        attribute rate
        [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataDensity.Velocity.hasRate] : int 50
        UnitTemplateCamelModel.UnitTemplateModel.GigaBytesPerDays  we indicate that this input velocity is 50 GB per day

    }

}
```

As it can be seen from this fragment, the first dataset maps to table data that have a key-value pair form; this latter fact is given in the form of a MDS annotation. For the second data set, we refer to its data source, which is external, while we define a feature for it. This feature maps to the velocity notion from the MDS and includes the modelling of two attributes, which actually characterise this notion. The first indicates that the velocity is an input one. To indicate this, we firstly annotate the attribute and then we provide the "true" boolean value for it. The second indicates the rate of the velocity. This attribute is also annotated from the MDS while we supply the int value of 50 for it and we associate it with the unit of GigaBytesPerDay. This ends our intuition to specify that the second dataset should have an input velocity of 50 GBs per day.

Specifying characteristics of a data set is performed at the type level in this way. This indicates that any true instance of this data set will comply with this characteristic. However, such an instance might be also characterised by other attributes which could be more dynamic in nature. For instance, all instances of a certain data set might have a different size. Moreover, some of these instances might have a dynamic size, if it is updated regularly, either from an external application, or the actual application managed by the respective platform like Melodic. In this respect, a data set instance should always have a type and might also refer to an instance of the data source that encapsulates it. It can also include other data (set) instances, similarly to the composition capability implanted in its respective type. In the following fragment, we supply one instance from the second of the above data set types (see previous fragment) which has a certain size.

```
data instance StreamDataInstance{

    type StreamData  a reference to the type of this data instance

    source instance S3DataSourceInstance  a reference to the source instance of this data instance

    attribute size [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataDensity.Volume.hasSize] : int
    100 UnitTemplateCamelModel.UnitTemplateModel.GigaBytes  here we indicate that the size of this data instance is 100 GB

}
```

This fragment indicates that the type of this data instance is StreamData, that its source instance is S3DataSourceInstance (see example below) and that it has an attribute named size, annotated from the MDS, which has the int value of 100 and the unit of GigaBytes. Thus, we indicate that this data set instance has a size of 100 GBs.

A data source instance is a certain instantiation of a data source type. For instance, in case of the S3 data source, this could be a certain source instance (e.g., a bucket) which is situated in a certain location. A data source instance has certainly a type (a data source (type)), while it is situated in a certain location, usually cloud-based. Further, it could be managed / encapsulated by an instance of a (data) component that is part of the user application being executed. Continuing our example, the following fragment showcases the data source instance for the S3 data source type (which is also referenced from the example about of a data set instance).

```
data source instance S3DataSourceInstance{

    type S3DataSource  here we refer to the type of the data source instance

    location Locations.DE  here we refer to the location of this data source

}
```

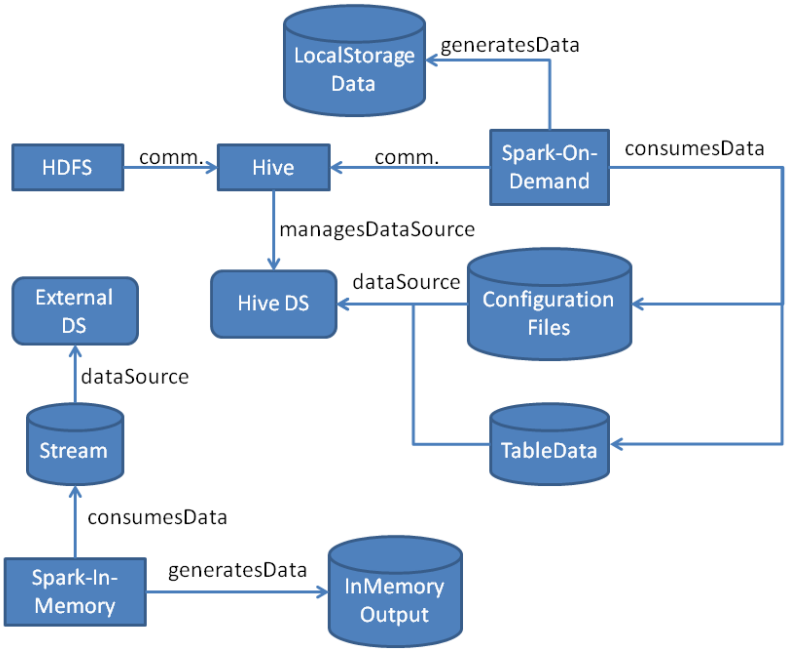
As it can be seen, this data source instance refers to its actual type while it is also associated to a certain location, which is Germany in our case (identified by its ISO code).

In general, a data type model should be specified by the modeller whenever a data processing application needs to be modelled. On the other hand, the data instance model is not always needed to be specified. This depends on the initial situation of the user application before it is being deployed and especially of its data. This actually indicates that we should provide the data instance model only when an external data source is involved for which we need to clearly declare its instance and its actual location. This can then enable the platform to perform deployment reasoning based also on the data aspect such that it can place the data processing components which should process the data of this data source close to this data source. However, we do not exclude the situation that some user nodes are up and running and are registered in the system like in a "bring your own node" fashion. Such nodes then will be manageable by the platform and could be used to host data sources. Thus, we foresee that only in this additional case the modeller should specify a data instance model. However, we should highlight that this data instance model will be eventually generated once the application is being deployed and executed by the platform. This model will be both maintained and expanded by the platform as new data instances are generated.

In the following, we supply a complete example of a data type and instance model coming from the PeopleFlow use case.

Example

In order to explicate the semantics of the newly introduced CAMEL extension that covers the data aspect, we provide now a certain example which attempts to exemplify how a big data application can be modelled in CAMEL. This example has been adopted from the PeopleFlow use case of CET while focuses mainly on deployment and data aspects and not on others like metric and scalability ones. It is graphically depicted in the figure below.



In this example, the modeller would need to specify for the respective CAMEL model 3 sub-models: (a) a deployment type model; (b) a data type model; (c) a data instance model. A unit model could have needed to be also specified but, fortunately, a unit template model is already offered by the CAMEL editor comprising units which suffice for the modelling of this use case. The deployment type model specifies the provider-independent deployment model of the application according to the aforementioned vision. This model includes the description of 4 application components, 2 communication connections between them and 1 component coupling.

The information about the 4 application components is as follows:

- HDFS: this is a long-lived internal distributed file system component mapping to the *HDFS* annotation in the meta-data schema. This component provides a certain communication port allowing the Hive component to connect to it.
- Hive: this is a long-lived internal data warehouse component which sits on top of HDFS. In this sense, it needs to communicate with it. So, it exposes a respective required communication port. In addition, it includes a provided communication port which enables the external component of the application (which is not modelled) to connect and store data on it. This component manages the *TableData* D ata (see description of the data model below).
- SparkOnDemand: this is a short-lived internal application component mapping to a big data computing task. This component is annotated with the SPARK concept from the meta-data schema instance model. It takes as input *TableData* and *ConfigurationFiles* data and produces as output *LocalStorageData* data. It exposes one communication port in order to connect to Hive and obtain the respective *Tabl eData* and *ConfigurationFiles* from there.
- SparkInMemory: this is a short-lived internal application component / task that has the same annotation as the previous one. It takes as input the *Stream* data. It does produce output data (named as InMemoryOutput) which are sent back to the external data source. It exposes one provided communication port allowing the external data dource to connect to it and send the respective stream and one required communication port in order to connect to the external data source and send back the data produced.

The sole component coupling is more or less obvious by also considering the way components are connected. It highlights that Hive and HDFS

need to be coupled together in the same host. This is normal as a lot of communication is involved between these two components. For the sake of brevity, we do not supply the textual content of this deployment model as this is the subject of another Confluence page. Thus, our focus is more on the data model part.

The data type model attempts to model the data that are managed or processed by the internal application components as well as the respective data sources. In this respect, we have 5 data items which have been modelled:

- *Stream*: this is a near-real time feed (see respective concept in the meta-data schema) which is managed by the external (data) source. This data item has an input velocity of 50 GBs/day. This characteristic has been modelled through the assistance of the meta-data schema instance model based on a certain feature named as *InputVelocity* that maps to the *Velocity* concept in that model. This feature then has two attributes: (a) one named as *isInputVelocity* annotated with equivalently named property in meta-data schema and taking the value of true. This indicates that the velocity is an input one; (b) one named as *rate* annotated with the *hasRate* property from the meta-data schema instance model. This attribute takes the value of 50 and maps to the *GigaBytesPerDay* unit which has been defined in the unit sub-model of the CAMEL model. This has been already shown in the previous section of this page but it is repeated for completeness reasons.

```
data StreamData {

    source S3DataSource  the stream data come from the S3DataSource

    feature InputVelocity{

        [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataDensity.Velocity]  this is the velocity notion from the meta-data schema

        attribute isInput [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataDensity.Velocity.isInputVelocity] : boolean true  we deal with an input velocity for the data

        attribute rate [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataDensity.Velocity.hasRate] : int 50 UnitTemplateCamelModel.UnitTemplateModel.GigaBytesPerDays  we indicate that this input velocity is 50 GB per day

    }

}
```

- *LocalStorageData*: these are just local storage data which are generated by the *SparkOnDemand* component. They take the form of CSV files and thus the respective data item has been annotated with the concepts *FileFormat* and *CSV* from the meta-data schema instance model.

```
data LocalStorageData {

    [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataVariety.Format.FileFormat]

    [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataVariety.Format.CSV]

}
```

- *TableData*: these are tabular data in the form of key-value pairs, so they obtain the corresponding annotation from the MDS. These table data are encapsulated by the Hive data source.

```
data TableData{

    [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataVariety.Format.Key_ValuePairs]

    source HiveDataSource

}
```

- *ConfigurationFiles*: these are configuration files required for the execution of the *SparkOnDemand* application component. The *FileFormat* annotation from MDS was used to decorate them. These configuration files are encapsulated by the Hive data source.

```
data ConfigurationFiles {

    [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataVariety.Format.FileFormat]

    source HiveDataSource

}
```

- *InMemoryOutput*: output data in the form of a file which are sent to the external data source.

```
data InMemoryOutput {

    [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataVariety.Format.FileFormat]

}
```

The respective data sources that have been modelled with the data type model are the following:

- *S3DataSource*: this is the external data source which supplies the *Stream* and ConfigurationFiles Data.

```
data source S3DataSource{

    external

}
```

- *HiveDataSource*: this is the data source that is encapsulated by the Hive data component of the application. Such data source manages the *TableData* and *ConfigurationFiles* Data.

```
data source HiveDS{

    component PeopleFlowDepModel.Hive

}
```

The data instance model needs to contain just the instances of the external data source and its data. This maps to the following fragment in CAMEL 2.0:

```
data source instance S3DataSourceInstance{

    type S3DataSource  here we refer to the type of the data source instance

    location Locations.DE  here we refer to the location of this data source

}

data instance StreamDataInstance{

    type StreamData  a reference to the type of this data instance

    source instance S3DataSourceInstance  a reference to the source instance of this data instance

    attribute size [MetaDataModel.MELODICMetadataSchema.Big_DataModel.Big_DataAspects.DataDensity.Volume.hasSize] : int
    100 UnitTemplateCamelModel.UnitTemplateModel.GigaBytes  here we indicate that the size of this data instance is 100 GB

}
```

In this fragment, we specify that the type of the data source instance is S3DataSource and its location is Germany. In the case of the data instance, we refer to its type, its data source instance which is the one that has been modelled while we also specify its actual size in GBs.