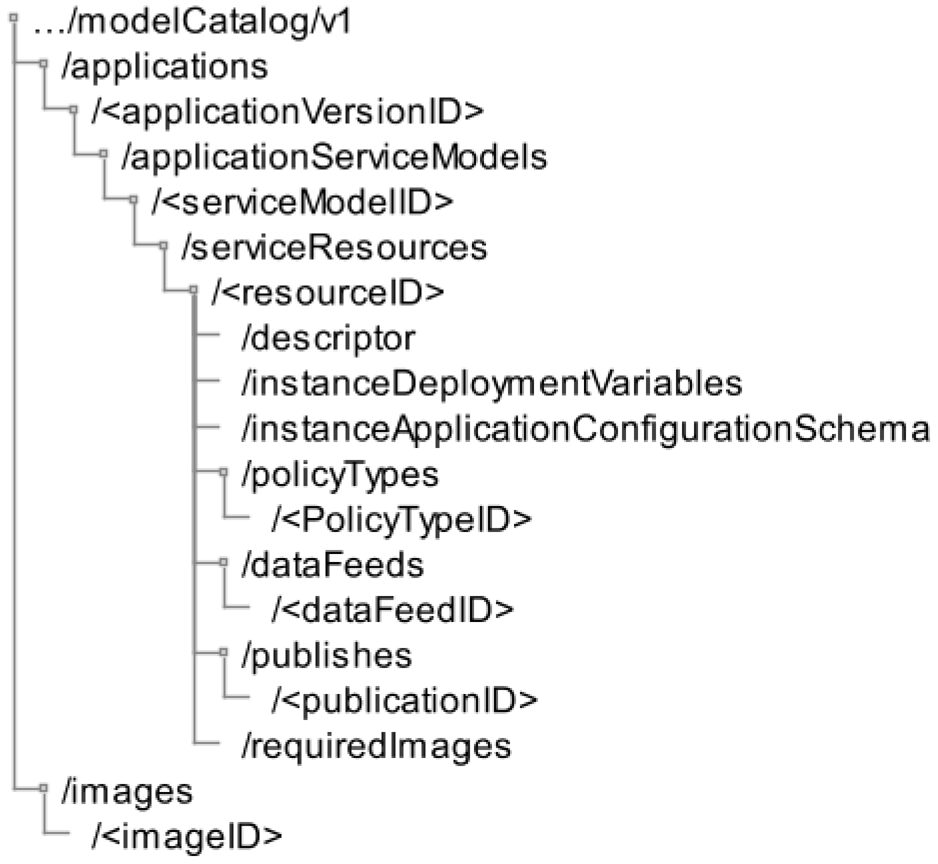
Cherry - <<SMO>> ModelCatalog

SMO - Model Catalog

The Model Catalog provides a list of Applications that are onboarded to the SMO after delivery of an "Application Package" from a vendor. It presents a Model-View-Controller for management of the data.

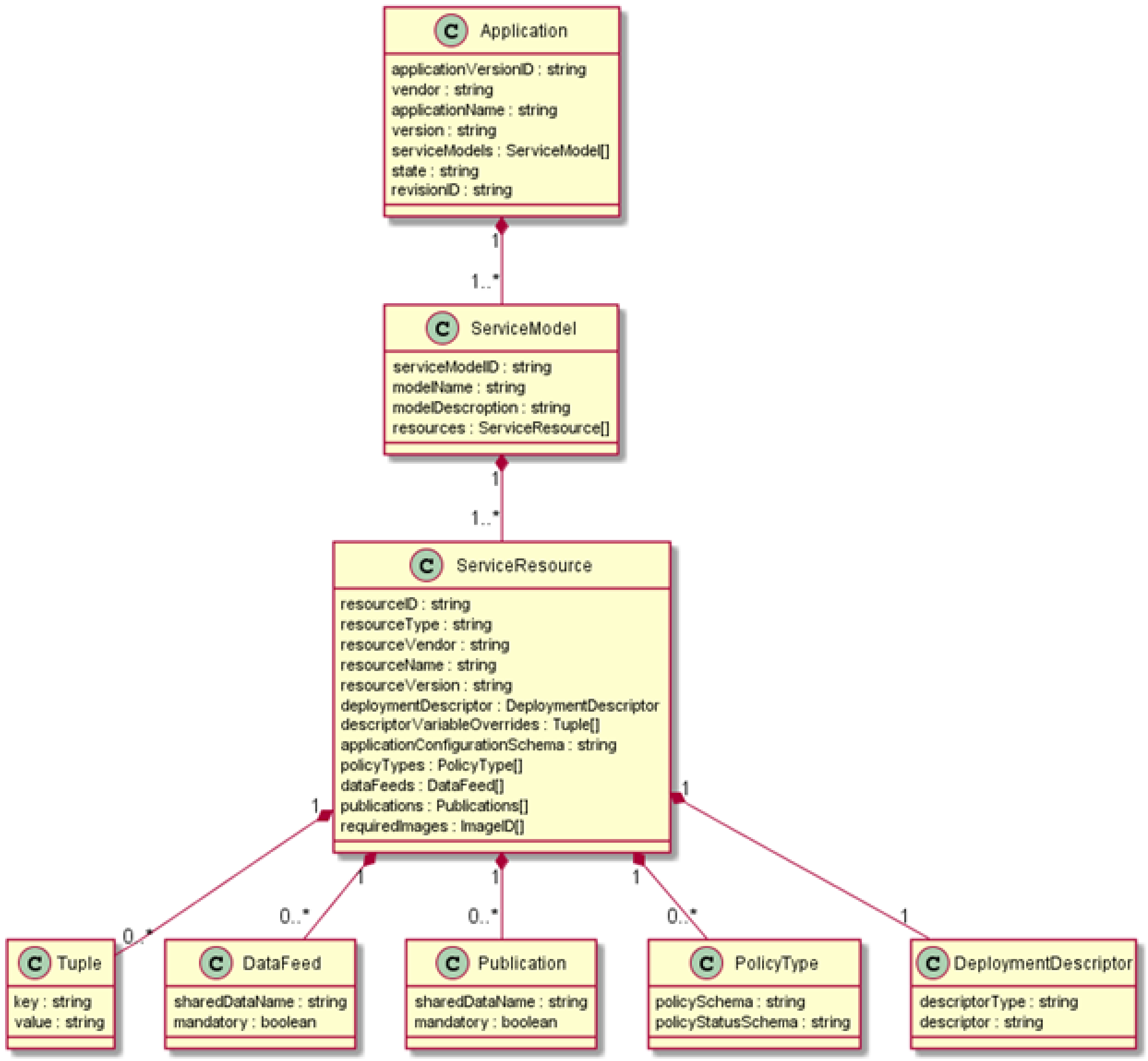
For the purposes of this implementation we will assume the following:

- The Controller will be RESTful from the service endpoint of {apiRoot}/ModelCatalog/v1. The initial REST resource tree is show below. Additional resources may be added to represent aspect of ML Applications which have additional data elements.



```
@startsalt
{
scale 1.5
{T
+…/modelCatalog/v1
++/applications
+++/<applicationVersionID>
++++/applicationServiceModels
+++++/<serviceModelID>
++++++/serviceResources
+++++++/<resourceID>
++++++++/descriptor
++++++++/instanceDeploymentVariables
++++++++/instanceApplicationConfigurationSchema
++++++++/policyTypes
+++++++++/<PolicyTypeID>
++++++++/dataFeeds
+++++++++/<dataFeedID>
++++++++/publishes
+++++++++/<publicationID>
++++++++/requiredImages
++/images
+++/<imageID>
}
}
@endsalt
```

- The Model is represented in the class diagram below

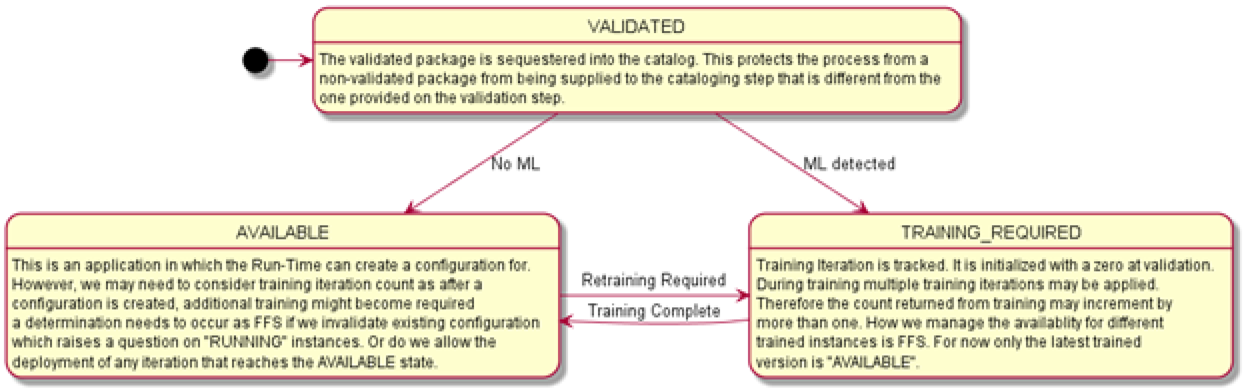


```
@startuml

Class Application {
   applicationVersionID : string
   vendor : string
   applicationName : string
   version : string
   serviceModels : ServiceModel[]
   state : string
   revisionID : string
}

Class ServiceModel {
   serviceModelID : string
   modelName : string
   modelDescroption : string
   resources : ServiceResource[]
}

Class ServiceResource {
   resourceID : string
   resourceType : string
   resourceVendor : string
   resourceName : string
   resourceVersion : string
   deploymentDescriptor : DeploymentDescriptor
   descriptorVariableOverrides : Tuple[]
   applicationConfigurationSchema : string
   policyTypes : PolicyType[]
   dataFeeds : DataFeed[]
   publications : Publications[]
   requiredImages : ImageID[]
}

Class Tuple {
   key : string
   value : string
}

Class DataFeed {
   sharedDataName : string
   mandatory : boolean
}

Class Publication {
   sharedDataName : string
   mandatory : boolean
}

Class PolicyType {
   policySchema : string
   policyStatusSchema : string
}

Class DeploymentDescriptor {
   descriptorType : string
   descriptor : string
}

Application "1" *-- "1..*" ServiceModel
ServiceModel "1" *-- "1..*" ServiceResource
ServiceResource "1" *-down- "1" DeploymentDescriptor
ServiceResource "1" *-down- "0..*" Tuple
ServiceResource "1" *-down- "0..*" PolicyType
ServiceResource "1" *-down- "0..*" DataFeed
ServiceResource "1" *-down- "0..*" Publication

@enduml
```

- The View will be JSON.

The Application record in the Model Catalog follows a Stateful lifecycle. The State can be updated with a partial update (PUT) as long as the current revisionID is supplied as a query parameter. Upon a successful update the revisionID will be changed by the system to a newly generated value.

The valid values for State are "VALIDATED", "TRAINING_REQUIRED", and "AVAILABLE". The state transitions allowed are:



The validated package is sequestered into the catalog. This protects the process from a non-validated package from being supplied to the cataloging step that is different from the one provided on the validation step.

This is an application in which the Run-Time can create a configuration for. However, we may need to consider training iteration count as after a configuration is created, additional training might become required a determination needs to occur as FFS if we invalidate existing configuration which raises a question on "RUNNING" instances. Or do we allow the deployment of any iteration that reaches the AVAILABLE state.

Training Iteration is tracked. It is initialized with a zero at validation. During training multiple training iterations may be applied. Therefore the count returned from training may increment by more than one. How we manage the availablity for different trained instances is FFS. For now only the latest trained version is "AVAILABLE".

```
@startuml
[*] -> VALIDATED
VALIDATED : The validated package is sequestered into the catalog. This protects the process from a
VALIDATED :  non-validated package from being supplied to the cataloging step that is different from the
VALIDATED :  one provided on the validation step.

VALIDATED -down-> TRAINING_REQUIRED : ML detected
VALIDATED -> AVAILABLE : No ML
TRAINING_REQUIRED -> AVAILABLE : Training Complete
AVAILABLE -> TRAINING_REQUIRED : Retraining Required

TRAINING_REQUIRED : Training Iteration is tracked. It is initialized with a zero at validation.
TRAINING_REQUIRED : During training multiple training iterations may be applied.
TRAINING_REQUIRED : Therefore the count returned from training may increment by
TRAINING_REQUIRED : more than one. How we manage the availablity for different
TRAINING_REQUIRED : trained instances is FFS. For now only the latest trained
TRAINING_REQUIRED : version is "AVAILABLE".

AVAILABLE : This is an application in which the Run-Time can create a configuration for.
AVAILABLE : However, we may need to consider training iteration count as after a
AVAILABLE : configuration is created, additional training might become required
AVAILABLE : a determination needs to occur as FFS if we invalidate existing configuration
AVAILABLE : which raises a question on "RUNNING" instances. Or do we allow the
AVAILABLE : deployment of any iteration that reaches the AVAILABLE state.

@enduml
```