Module Manager Specification

Fanny Boulaire Mark Utting

March 25, 2015

section *ModuleManager* **parents** *standard_toolkit*, *ModamTypesAnimate*

This specification describes the ModuleManager of MODAM.

A *SetterGetter* represents a property of an object whose value can be read and set via getter/setter methods. It has a name, an argument type (one only), a current value (for Modam, this is always a reference to another Java class), and a flag which specifies whether the property is optional or compulsory.

SetterGetter

$$
\begin{array}{l}
name : MethodName \\
argType : JavaType \\
value : ClassName \\
optional : BOOLEAN
\end{array}
$$

We start by defining the Eclipse concept of 'extension class' (or extension point?), which contains a class name plus several properties. We call these extension classes *Factory* classes, because in MODAM they are used to create assets and agents and data provider objects.

Factory

$$
\begin{array}{l}
extId : ExtId \\
className : ClassName \\
consumes : \mathbb{P}\, DataId \\
produces : DataId \\
path : Path \\
prior : \mathbb{P}\, ClassName \\
methods : \mathbb{P}\, SetterGetter \\
\hline
true
\end{array}
$$

In MODAM we use three extension points, for assets, behaviours, and data providers respectively. So we use a separate ID for each extension point: *modamAssetFactory*, *modamBehavFactory*, and *modamDataProvider*.

We also define the special (subclass) characteristics of each kind of extension point. For example, asset factories and behaviour factories may consume data providers but they do not produce data, whereas data providers produce data but do not not consume other data providers. Asset factories are the only kinds of factories whose execution has to be ordered, so we set $prior = \varnothing$ for the other factories.

$$
\begin{aligned}
AssetFactory == [&Factory \mid extId = modamAssetFactory \land \\
&produces = noDataId \land path = noPath] \\
BehavFactory == [&Factory \mid extId = modamBehavFactory \land \\
&produces = noDataId \land prior = \varnothing \land path = noPath] \\
DataProvider == [&Factory \mid extId = modamDataProvider \land \\
&consumes = \varnothing \land prior = \varnothing \land produces \neq noDataId]
\end{aligned}
$$

A Module in MODAM is essentially just a contributor ID plus a set of extension classes, which must all have unique names.

*Module*0

$$
\begin{aligned}
&contributor : Contributor \\
&classes : \mathbb{P}\, Factory
\end{aligned}
$$

For specification convenience, we extend Module to derive several mappings from factory class names to various attributes of those factories.

Module

$$
\begin{aligned}
&Module0 \\
&extensionClass : ExtId \leftrightarrow ClassName \\
&enabledClasses : \mathbb{P}\, ClassName \\
&consumes : ClassName \twoheadrightarrow \mathbb{P}\, DataId \\
&produces : ClassName \twoheadrightarrow DataId \\
&path : ClassName \twoheadrightarrow Path \\
&prior : ClassName \twoheadrightarrow \mathbb{P}\, ClassName \\
\hline
&extensionClass = \{c : classes \bullet (c.extId, c.className)\} \\
&enabledClasses = \{c : classes \bullet c.className\} \\
&consumes = \{c : classes \bullet (c.className, c.consumes)\} \\
&produces = \{c : classes \bullet (c.className, c.produces)\} \\
&path = \{c : classes \bullet (c.className, c.path)\} \\
&prior = \{c : classes \bullet (c.className, c.prior)\}
\end{aligned}
$$

The *consumes* field maps every factory (asset factories and agent factories) to its allowable input data identifiers. *Note that the implementation goes the other way around, which assumes that each input data id is required by only one class within each module.* The domain of *consumes* is made of agentFactories and assetFactories that are distinct classes, and that make the whole set of classnames.

The *produces* maps every data provider class name to the data id that it produces.

The *path* maps each data provider class name to its default path (if one has been specified).

The *contributor* field is used to access the OSGi bundle but the name will be sufficient.

The domain of *produces* is the set of all data providers. The domain of *consumes* is the set of all factories. The domain of *prior* is the asset factories, and the remaining ones in *consumes* are agent factories.

The union of the range of *prior* is within the domain of prior, which is the set of assetFactories.

TODO: simplify this to just a set of modules?

ModuleManager

$$modules : Contributor \pfun Module$$
$$outputid : ClassName \pfun DataId$$

$$(\forall\, n : dom\, modules \bullet (modules\, n).contributor = n)$$
$$outputid = \bigcup \{n : dom\, modules \bullet (modules\, n).produces\}$$

The *ModuleManager* has a set of *modules* which map a *Contributor* to a *Module*, and of *outputid* that map a classname to a data id. Each contributor is uniquely associated to a module. An *outputid* is the union of all the contributors which have a set of data providers

TODO: we could specify the GetMissingDependencies method too?

MissingDependencies

$$\Xi ModuleManager$$
$$missing! : \mathbb{P}\, DataId$$

$$true$$

One of the important operations in Modam is automatically analyzing each property of a requested factory to see if it can be satisfied by the available data providers. The following *SatisfySetterMethod* specifies how this is done.

SatisfySetterMethod

$m, m' : SetterGetter$
$dataProviders? : \mathbb{P}\ ClassName$
$outputid : ClassName \twoheadrightarrow DataId$
$matching! : \mathbb{P}\ ClassName$
$value! : ClassName$

$m'.name = m.name$
$m'.argType = m.argType$
$m'.optional = m.optional$
$m'.value = value!$
$matching! = \{d : dataProviders? \mid outputid\ d = m.name\}$
$(\#\ matching! = 1 \Rightarrow value! \in matching!) \land$
$(\#\ matching! > 1 \Rightarrow value! = dataProviderError) \land$
$(\#\ matching! = 0 \land m.optional = YES \Rightarrow value! = m.value) \land$
$(\#\ matching! = 0 \land m.optional = NO \Rightarrow value! = dataProviderError)$

The *SatisfySetterMethod* transforms the value of a *SetterGetter* method after having matched an *outputid* to its dataProvider from a set of input *dataProviders*. The *matching*! output set contains all the available data providers that can satisfy this property. Then there are four cases:

- the sweet path corresponds to having exactly one match, which leads to setting the value of the method to that match;
- another path where there is no match found but the matching was optional, leading to an unchanged method;
- an error is thrown if no matches are found and the property is not optional;
- an error is thrown if there are multiple matches, so it is ambiguous which data provider the user intended to be used.

SatisfyFactoryInputs

$dataProviders? : \mathbb{P}\ ClassName$
$factory?, factory! : Factory$
$\Xi ModuleManager$

$factory?.className = factory!.className$
$factory!.methods = \{m' : SetterGetter \mid (\exists\, m : factory?.methods;$
$\quad matching! : \mathbb{P}\ ClassName;\ value! : ClassName$
$\quad \bullet\ SatisfySetterMethod)\}$
$\neg\ (\exists\, m : factory!.methods \bullet m.value = dataProviderError)$

The *SatisfyFactoryInputs* operation tries to satisfy all the properties in a given factory, by using the *SatisfySetterMethod* method to set each of their

dataProviders. However, if any of those properties throw an error, then the last line in this operation ensures that the whole operation fails.

Finally, this *getMissingDependencies* operation finds all the non-optional data requirements that are not satisfied by any of the data providers. This indicates that the current model setup is incomplete, so the model cannot be run.

$$
\begin{array}{|l}
\textit{getMissingDependencies} \\
\hline
\Xi \textit{ModuleManager} \\
\textit{factories}? : \mathbb{P}\, \textit{Factory} \\
\textit{missing}! : \mathbb{P}\, \textit{DataId} \\
\hline
\textit{missing}! = \{\textit{unsatisfied} : \textit{DataId} \mid \\
\quad (\exists\, m : \textit{ran modules};\ f : \textit{factories}?;\ s : \textit{SetterGetter} \bullet \\
\qquad f.\textit{className} \in m.\textit{enabledClasses} \land \\
\qquad f.\textit{className} \in \textit{dom}\, m.\textit{consumes} \land \\
\qquad \textit{unsatisfied} \in m.\textit{consumes}\, (f.\textit{className}) \land \\
\qquad s \in f.\textit{methods} \land \\
\qquad s.\textit{name} = \textit{unsatisfied} \land \\
\qquad s.\textit{optional} = \textit{NO} \land \\
\qquad (\neg\ \exists\, m' : \textit{ran modules};\ \textit{provider} : \textit{ClassName} \bullet \\
\qquad\quad \textit{provider} \in \textit{dom}\, m'.\textit{produces} \land \\
\qquad\quad \textit{provider} \in m'.\textit{enabledClasses} \land \\
\qquad\quad m'.\textit{produces}\, \textit{provider} = s.\textit{name} \\
\qquad) \\
\quad)\} \\
\hline
\end{array}
$$