Projet PLUMES
Plateforme Logicielle Unifiée de Modélisation pour l'Efficacité énergétique du bâtiment et de ses Systèmes

Interfaces du composant MUSE

Tâche 2

Projet financé par l'Agence Nationale de la Recherche

Acronyme projet: PLUMES
Titre: Plateforme Logicielle Unifiée de Modélisation pour l'Efficacité énergétique du bâtiment et de ses Systèmes

Livrable :
Date de livraison prévue: 30/10/2011
Date de livraison: 29/06/2012

Résumé:

Dans ce document, complémentaire du livrable 2.1 (spécifications du composant MUSE), on définit les interfaces informatiques standards que devront respecter les composants MUSE afin de garantir leur interopérabilité. Sont décrites les interfaces génériques (permettant notamment de réaliser l'introspection des composants), et les interfaces spécifiques aux différents domains métier visés (simulation, dimensionnement, gestion). Les mécanismes et technologies sous-jacents permettant de gérer la composition et de faciliter le déploiement sont également décrits.

Auteur: Benoît Delinchant (G2ELab)
Contributeurs: S. Ploix, F. Verdière, P. Enciu, S. Bergeon
Diffusion: Consortium PLUMES, ANR

Evolution du document			
Version	Date	Auteur, Affiliation	Modification effectuée
1.0	30/04/2012	B. DELINCHANT	Structure du document
1.1	07/05/2012	PLOIX & DELINCHANT	Structure des méta-données
1.2	08/05/2012	B. DELINCHANT	Simulation / Dimensionnement
1.3	10/05/2012	F. VERDIERE	Structure des méta-données
1.4	15/05/2012	B. DELINCHANT	OSGi
1.5	21/06/2012	DELINCHANT / ENCIU BERGEON / VERDIERE	Convergence meta-données / structure OSGi / lien MAVEN / facette de gestion
1.6	28/06/2012	DELINCHANT	Version finale

Relecture: S. Robert (CEA LIST)

1. Description d'un composant MUSE

Un composant MUSE offre des services dédiés à :

- l'introspection : par lecture d'un fichier XML standardisé
- l'accès aux ports et aux services métiers (« business » : simulation, dimensionnement, gestion, …) par des API (interface de programmation)
- l'accès à des ressources contenues dans le composant (fichiers)

Un composant MUSE se présente sous la forme d'un fichier avec l'extension « .muse ». Ce fichier est au format JAR (Java ARchive, développé par Sun Microsystems). C'est en fait un fichier d'archives (c'est-à-dire contenant plusieurs fichiers) compressés (format ZIP développé par PKWare) exploitable par tout outil indépendamment du langage de programmation et du système d'exploitation.

Un MUSE se déploie de manière simple dans différents types d'environnements grâce à la notion de plug'in. Un plug'in est un code informatique spécifique developpé pour chaque environnement et exploitant des API fournies par le framework MUSE[1]. Il peut être autonome ou dépendant d'autres composants MUSE. Les services qu'il contient peuvent être gérés dynamiquement (suppression / ajout).

Un composant MUSE peut être atomique ou composite, il peut dans les deux cas disposer de plusieurs types de facettes, chacune de ces dernières possédant des méta-données, des API et des ressources. Les ports d'un composant atomique correspondent exactement aux ports de son unique facette. Dans le cas de composants multimétiers ou composites, les ports du composant sont définis par des références aux ports des facettes.

- Dans le cas d'un composant atomique mono-métier Figure 1 - cas des outils de modélisation exportant un composant MUSE par plug'out -, la facette est directement exploitable via ses meta-données, API et ressources. Des API génériques d'instanciation (MuseFactory) et d'introspection (MuseIntrospector) seront directement exploitables à partir du MuseFramework.

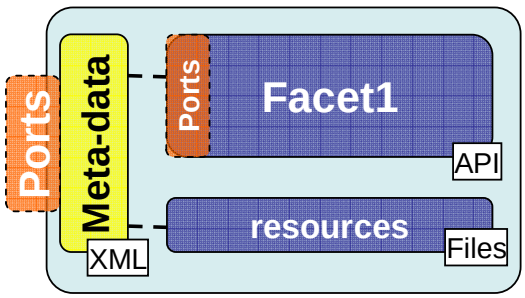

Figure 1 : Vue structurelle d'un composant MUSE atomique

- Dans le cas d'un composant multi-facettes (Figure 2), un point d'entrée commun est défini pour centraliser les meta-données, et potentiellement des API et ressources (figure 2) spécifiques globales.

[1] API du Framework MUSE : bibliothèques de fonction JAVA permettant de charger, introspecter et exploiter un composant MUSE.

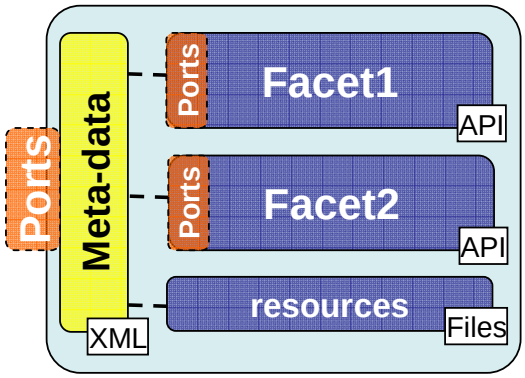

Figure 2 : Vue structurelle d'un composant MUSE multi-facettes

- Dans le cas d'un composant MUSE composite (Figure 3), on conserve une structure hiérarchique qui peut être parcourue si nécessaire. Le composant présente les facettes du système global, chacune interagissant avec les facettes filles des composants. Ces composants peuvent être intégré en resources ou définies en dépendances ce qui implique qu'ils doivent être accessibles (par exemple par référence à une bibliothèque de composants publique).

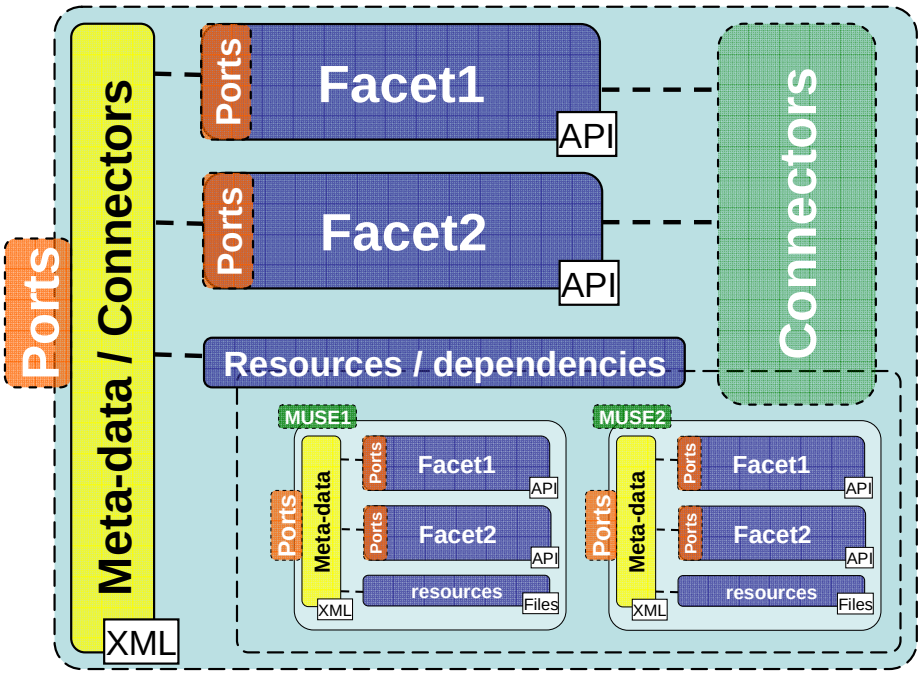

Figure 3 : vue structurelle d'un MUSE multi-facettes avec composition

A noter qu'il n'y a pas de différence de spécification entre ces trois niveaux (atomique/composite, mono/multi-métiers, composite) : elle est générique et permet de traiter les trois cas. C'est à l'aide des interfaces d'introspection que l'on peut connaître la nature d'un composant donné.

Une caractéristique forte des composants MUSE réside dans leur modularité, la plateforme d'exécution[2] étant dédiée à la gestion de tous ces modules (installation à distance, gestion de dépendances, utilisations multiples, mise à jour, …)

[2] Différentes plateformes d'execution sont envisagées car le composant MUSE pourra compatible OSGi et WebService. Une plateforme OSGi pourra par exemple être Felix (bibliothèque très légère) ou un serveur Tomcat (architecture plus conséquente).

2. Architecture informatique

2.1. La norme MUSE se base sur la norme OSGi

Un muse est une archive contenant ou faisant référence à un certain nombre de "bundle" OSGi[3]. Un bundle est tout simplement le nom donnée à une archive java (fichier ZIP) contenant des meta-données le renseignant, et conforme à la spécification du standard OSGi. Un bundle principal représente le composant MUSE (point d'entrée), et un bundle est associé à chaque service métier. De la même manière, un composant composite contient un bundle décrivant la composition (dépendance vers d'autres bundles) et les sous composants MUSE (bundle principal et bundles de service). La plateforme OSGi se charge alors de gérer le chargement et les dépendances automatiquement et nous autorise ainsi une gestion dynamique des services disponibles, ainsi que des sous-composants d'une composition.

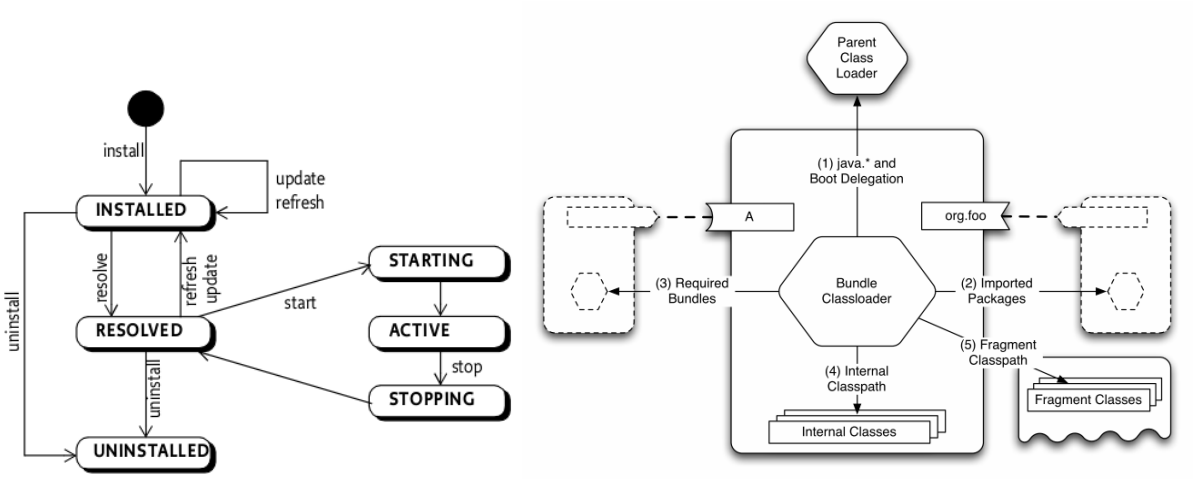

Figure 4 : à gauche : cycle de vie d'un bundle OSGi – la plateforme OSGi gère l'installation et l'exécution des bundles ; à droite : le « classloader » gère les dépendances (paquetages, bundles) du bundle

Durant la phase d'installation, le composant va installer les bundles internes tels que les services métiers ou les sous-composants dans le cas d'un composite. Le mécanisme interne permettant de lier les sous-bundles au bundle parent correspond au patron fournisseur/consommateur des architectures orientées services (SOA). Un ensemble de fournisseurs de services (e.g. un service de calcul) se met à disposition d'éventuels consommateurs. Le mécanisme permettant d'atteindre cette fonctionnalité dynamique est celui du triptyque « publication, découverte et invocation » comme le montre la Figure 5. Les services se publient quand ils sont disponibles, puis, lorsqu'un consommateur a besoin d'un service particulier, il le découvre dans le registre de services via un contrat de service, puis il l'invoque pour l'exploiter.

OSGi, et plus particulièrement iPOJO[4] pour la composition structurelle de services OSGi peut être vue comme une déclinaison de l'approche SOA en y apportant la contribution de l'approche à composants, et fusionner l'aspect dynamique de découverte des services et l'approche de composition récursive de composants. Ces concepts sont formalisés par le modèle SCA (Service

[3] http://www.osgi.org/Technology/HomePage

[4] iPOJO : injected POJO, http://cwiki.apache.org/confluence/display/FELIX/iPOJO

Component Architecture). C'est par exemple aussi le cas dans l'approche SOMA d'IBM[5] décrite dans la Figure 6.

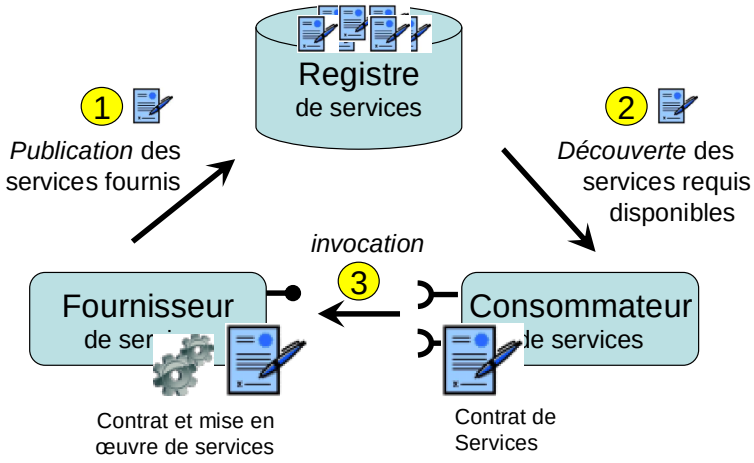

Figure 5 : tryptique "publication, découverte, invocation".

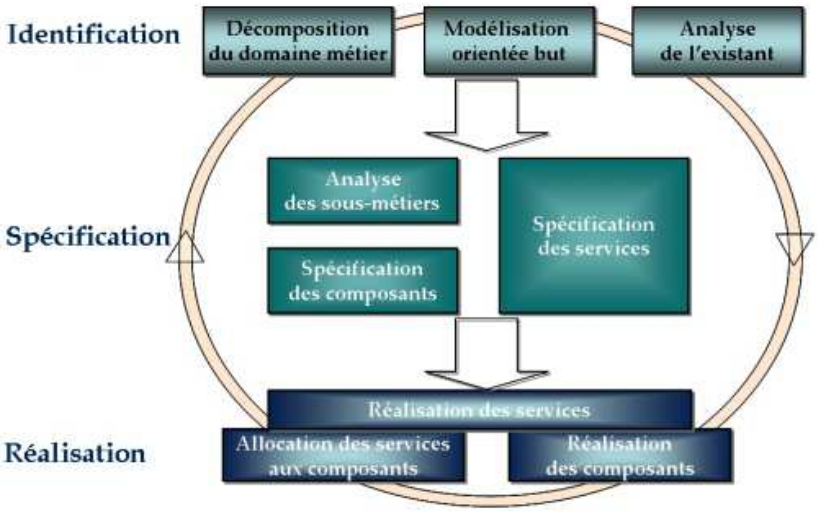

Figure 6 : architecture et modélisation orientée services (SOMA)

Un mécanisme par abonnement peut également faciliter la découverte dynamique. Par exemple, lorsqu'un composant MUSE est installé, il s'enregistre sur la plateforme et tous les outils abonnés aux composants MUSE sont informés qu'un nouveau composant est disponible.

Une fois le composant actif, les interfaces Java permettent d'exploiter les services offerts par le composant via leurs API. Ils permettent aussi d'introspecter le composant via des API, pouvant offrir les mêmes informations que la méthode plus souple par fichier XML.

2.2. La norme MUSE s'appuie sur la notion d'artefact

Le référencement des composants MUSE est décrit à l'aide d'"artefacts", qui constituent en quelque sorte la carte d'identité du composant. La notion d'artefact est à la base des systèmes de

[5] http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf

gestion de dépendances (tels que MAVEN[6], IVY[7], GRAPE[8] et GRADLE[9]). De manière générale, ces systèmes définissent un artefact par:

- Un identifiant unique d'éditeur, d'organisation ou de vendeur;
- Un identifiant unique nom de l'artefact;
- Une version unique de l'artefact.

Un composant MUSE ainsi référencé par son artefact pourra être déposé sur un repository distant, puis rendu accessible à la communauté. Différentes solutions logicielles sont aujourd'hui disponibles pour la gestion de tels repositories d'artefacts: ARCHIVA[10] (open source), ARTIFACTORY[11], NEXUS[12] …

De la meme manière, un composant MUSE composite décrit ses composants internes (enfants) sous la forme d'artefacts (i.e. dans les méta-données, chaque enfant du composant MUSE est défini par son artefact). A titre d'illustration, ci-dessous la représentation XML (MAVEN) :

```
<groupId>vendor.identifier</groupId>
<artifactId>artifact.name</artifactId>
<version>artifact.version</version>
```

[6] MAVEN : http://maven.apache.org/

[7] IVY : http://ant.apache.org/ivy/

[8] GRAPE : http://groovy.codehaus.org/Grape

[9] GRADLE : http://gradle.org

[10] ARCHIVA : http://archiva.apache.org/

[11] ARTIFACTORY : http://www.jfrog.com/products.php

[12] NEXUS : http://www.sonatype.org/nexus/

3. Méta-données ou données descriptives du composant

Les méta-données permettent de renseigner l'utilisateur du composant sur les services offerts sans nécessiter d'instancier le composant. Cette introspection « légère » est nécessaire par exemple pour obtenir certaines informations tels que les identifiants des ports du composant et des facettes nécessaires ensuite pour l'utilisation des API. Les méta-données fournissent également des informations complémentaires sur la nature des ports, l'origine des facettes…

3.1. Accès aux méta-données

Le fichier XML décrivant les méta-données du composant est accessible directement dans l'archive du composant, depuis la racine. Il peut être récupéré facilement, puis analysé par un parseur XML. Les contenus de ce fichier XML sont contraints par une grammaire XSD (fournie en annexe 1), que nous détaillons dans cette partie.

3.2. Données descriptives au niveau composant

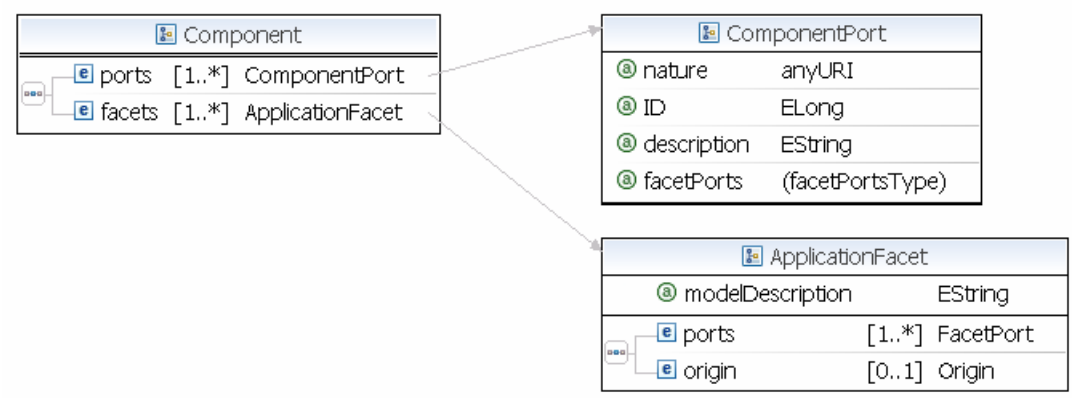

Figure 7 : Structure des méta-données : description d'un composant MUSE

Les méta-données permettent tout d'abord de réaliser l'introspection du composant en décrivant la liste de ses Ports et de ses Facettes.

Un composant pouvant être composite, il est possible de connaître la liste des ses composants internes et des connections.

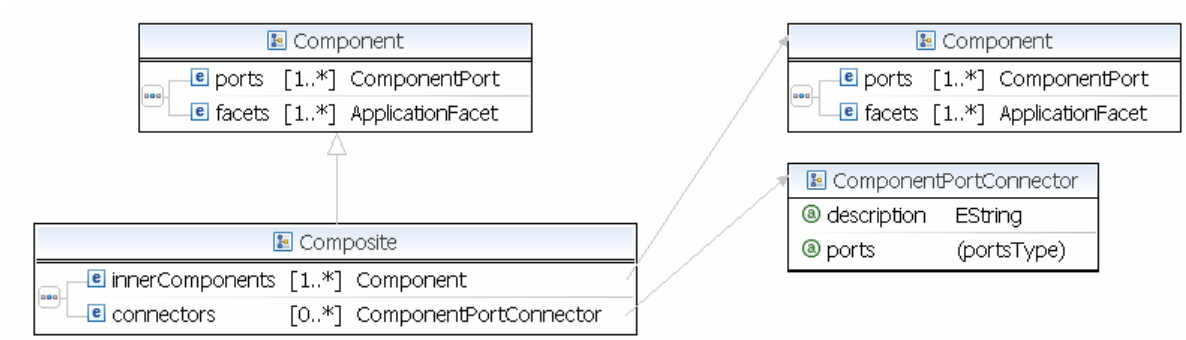

Figure 8 : Structure des méta-données : description d'un composant composite

3.3. Données descriptives au niveau facette métier

Une facette décrit les ports qu'elle exploite en référence aux ports définis dans le composant. Elle définit également son origine (langage, logiciel, fichiers sources,…)

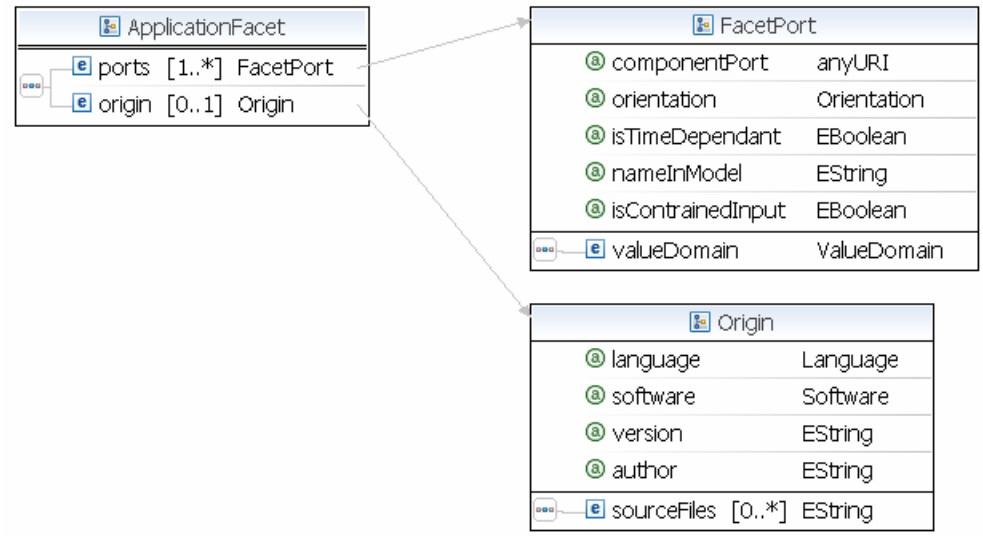

Figure 9: Structure des méta-données : description d'une facette

Les ports de la facette référencent des ports du composant. Ils complètent leur description par des spécificités métier (orientation, dépendant du temps, domaine de valeur,…)

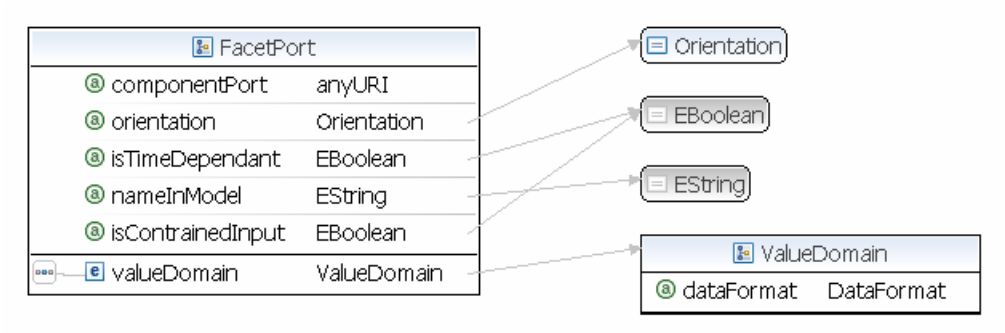

Figure 10 : Structure des méta-données : description d'un port de facette

Enfin, chaque facette métier est libre de définir sa structure de méta-données. Dans le cas de la simulation, le niveau de modélisation et les types de solver autorisés sont fournis.

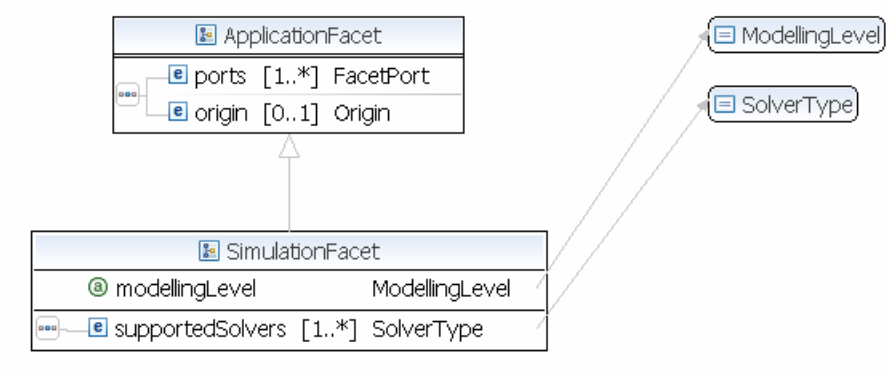

Figure 11 : Structure des méta-données : description d'une facette métier de simulation

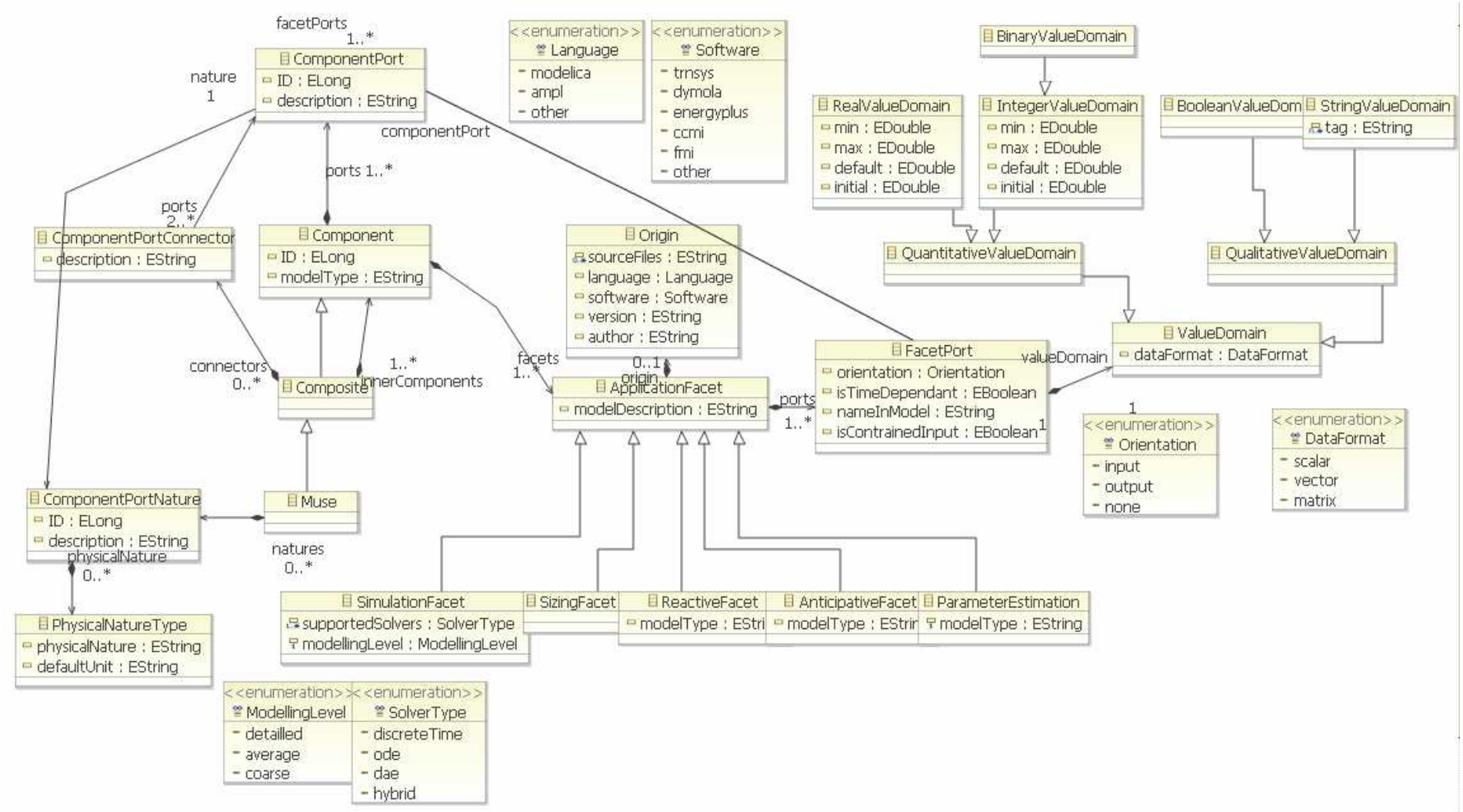

Figure 12 : Structure des méta-données : description de la structure des méta-donné

4. Interfaces de programmation – API – MUSE

4.1. Accès aux API

Le mécanisme OSGi est utilisé pour le chargement du composant, installant de manière transparente toutes les facettes et validant les dépendances existantes. L'installation peut se faire par un téléchargement des artefacts qui ne sont pas embarqués dans le composant. Ensuite, une simple API permet d'accéder aux facettes souhaitées puis d'accéder directement aux interfaces des objets.

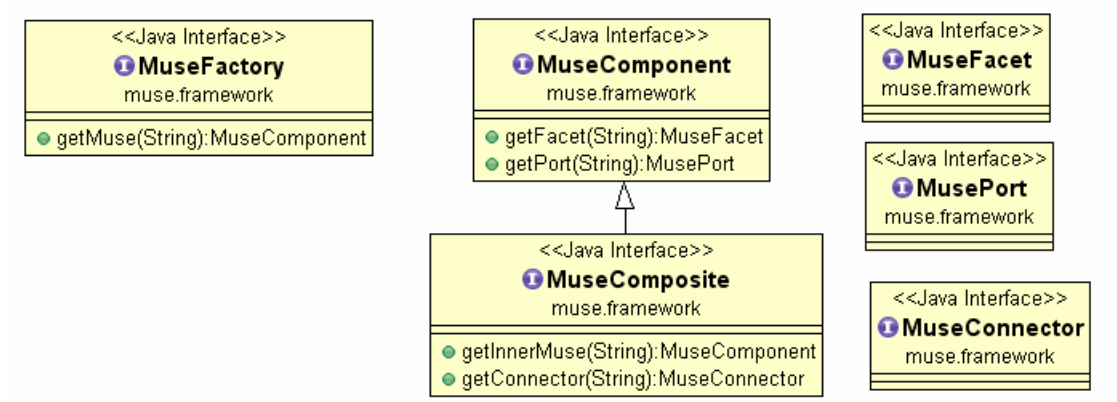

Figure 13 : Diagramme UML des API principales pour créer et utiliser un composant Muse

4.2. Facette de simulation

4.2.1. Les modèles mathématiques sous-jacents

La norme MUSE permet de décrire des modèles de type :

- **discret** permettant de gérer l'intégration en pas de temps fixe,
- **continu** sous la forme d'un système d'ODE (Ordinary Differential Equations),
- **hybride**, c'est-à-dire continu avec événements de type PDI (prochaine date importante) ou franchissement de seuil (zero crossing).

Le système ODE d'ordre N est traité en se ramenant à un système de N équation du premier ordre, explicite, linéaire ou non linéaire :

$$\dot{X}(t) = f(X(t), U(t), p, t) \qquad (1)$$

avec :

X : vecteur d'états
U : vecteur des sources
p : les paramètres
t : la variable temps.

et d'une équation vectorielle algébrique définissant les sorties du modèle en fonction du vecteur d'état :

$$Y(t) = g(X(t), U(t), p, t) \tag{2}$$

avec :

Y : vecteur des sorties

En particulier, les composants ICAr sont à même de définir des modèles dynamiques linéaires tels que :

$$\begin{aligned} \dot{X}(t) &= A \cdot X(t) + B \cdot U(t) \\ Y(t) &= C \cdot X(t) + D \cdot U(t) \end{aligned} \tag{3}$$

avec :

A, B, C, D : les matrices définissant le système dynamique linéaire.

Cette définition du système différentiel est classiquement adoptée dans les environnements de simulation tels que Simulink ou par les environnements traitant du langage Modelica (Dymola, AMESim, OpenModelica, …).

4.2.2. Diagramme UML

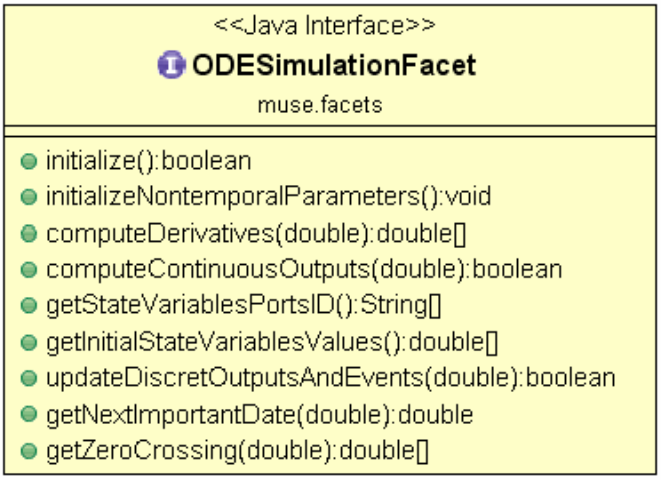

Figure 14 : Interface pour la simulation dynamique

4.2.3. Description des méthodes

Nom de la fonction	Description
`boolean initialize()`	**Initialize the state system. This method is only called once before simulation starts.** Returns **false if initialization failed, true else.**
`void initializeNontemporalParameters()`	**Initialize all the parameters which are independant of time**
`double[] computeDerivatives (double date)`	**Computes state variable derivatives** **dX/dt = f (X,U, P, t)** date **: the current simulation date** Returns **derivatives**

`boolean computeContinuousOutputs(double date)`	**Computes outputs Y = g(X, U, P, t)** date **: the current simulation date** Returns **true if no error occurs.**
`String[] getStateVariablePortsID()`	Returns **the array containing port ID of state variables**
`double[] getInitialStateVariablesValues ()`	Returns **the array containing initial values of state variables**
`boolean updateDiscretOutputsAndEvents (double date)`	**Update the state System. This method is called after each integration step. Usefull to update discret parameters. Because updates could make changes in the state system, and so change time constant.** date **: the current simulation date** Returns **true if dynamics are changed.**
`double getNextImportantDate(double date)`	**Compute the time value for the next major step in function of the system dynamic and discret state values** date **: the current simulation date** Returns **the time value of the next important date**
`double[] getZeroCrossing(double date)`	**Return the vector containing only variables to watch for a zerocrossing at the current time. This vector has to be reinitialized when an event occurs. The size can change.** date **: the current simulation date** Returns **the time value of zerocrossing date**

4.2.4. Utilisation du modèle par le solver

Pour aider le concepteur à mieux comprendre le fonctionnement des facettes pour la simulation, la Figure 15 présente le déroulement typique de leur utilisation au cours d'une simulation.

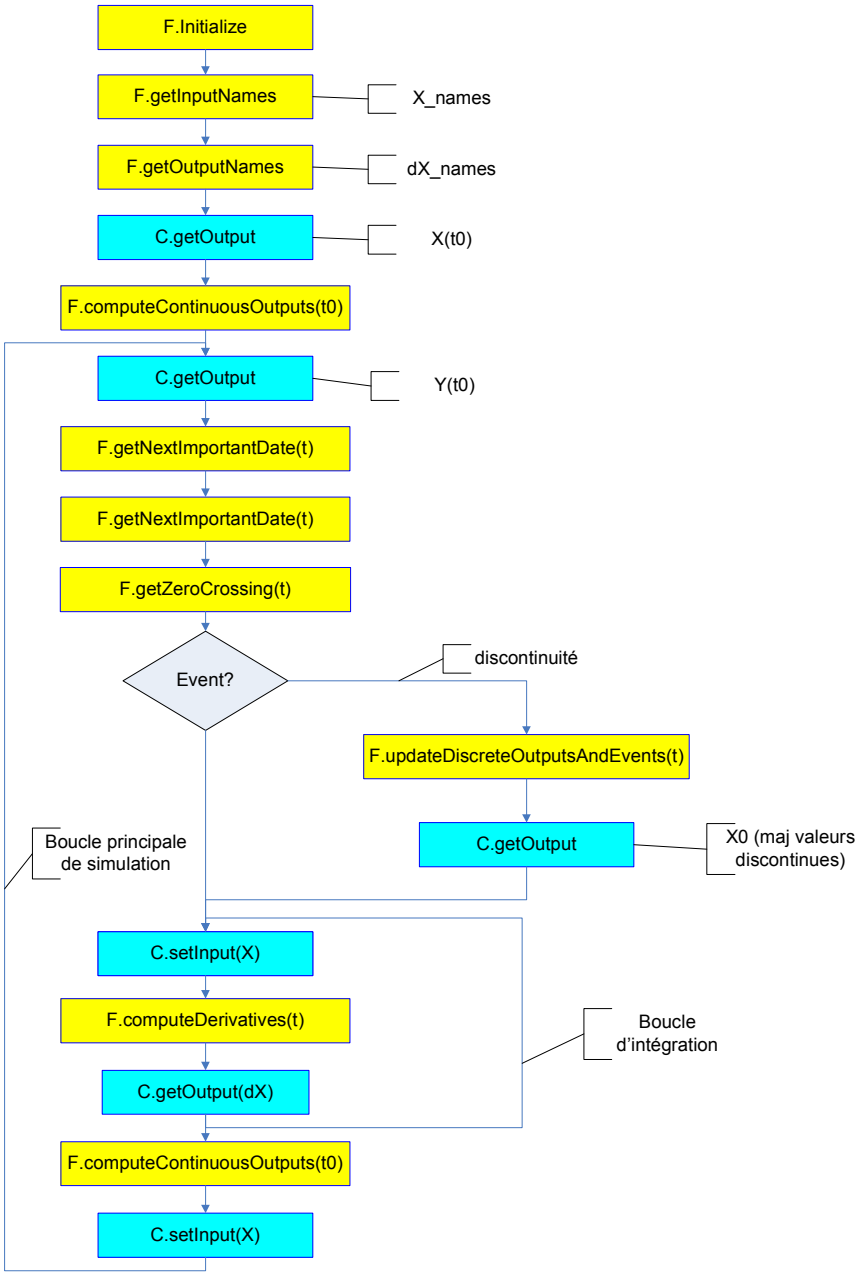

Figure 15 : Algorigramme définissant la séquence d'appels d'un modèle hybride continu/discret

4.3. Facette de dimensionnement

4.3.1. Les modèles mathématiques sous-jacents

On définit de manière classique le problème d'analyse ou de simulation comme un problème direct. Il calcule les performances (sorties : forces, vitesse, …) d'un système donné (entrées : géométrie, propriétés physiques, …). Réaliser un dimensionnement revient à résoudre le problème inverse qui consiste à définir le système (entrées : géométrie, propriétés physiques, …) pour atteindre des performances souhaitées (sorties : forces, vitesse, …).

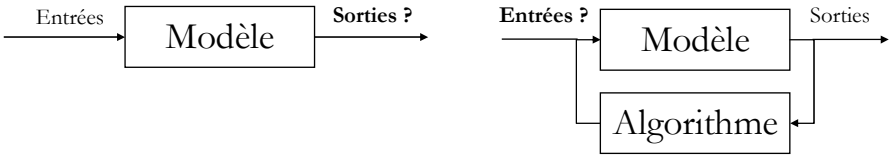

Figure 16 : Problème direct (simulation) et problème inverse (optimisation)

Ce problème peut se formuler comme un problème d'optimisation sous contraintes :

$$\begin{cases} \text{- minimize} & F(X) \\ \text{with} & X = \{x_1, x_2, \ldots x_n\} \in \Re^n \\ & x_i^{min} \le x_i \le x_i^{max} \quad i = 1, \ldots, n \\ \text{- subject to} & G_j(X) = 0, \quad j = 1, \ldots, m \\ & H_k(X) \le 0, \quad k = 1, \ldots, p \end{cases}$$

Des algorithmes d'optimisation existent et exploitent différentes classes de modèles. Par exemple, certains algorithmes nécessitent des modèles à variables purement discrètes ou purement continues, certains ne traitent que de modèles ou contraintes linéaires, d'autres exigent que le modèle fournisse le Jacobien des sorties par rapport aux entrées, d'autres prennent en considération des informations d'encadrement de la solution (optimisation par intervalle). Une typologie des algorithmes d'optimisation non exhaustive est esquissée dans la figure suivante, avec comme fil conducteur les problèmes d'optimisation continus, non linéaires,

- à convergence globale par approche meta-heuristique distribuée,
- à convergence locale exploitant l'information du gradient.

Avec la possibilité d'hybrider ces algorithmes.

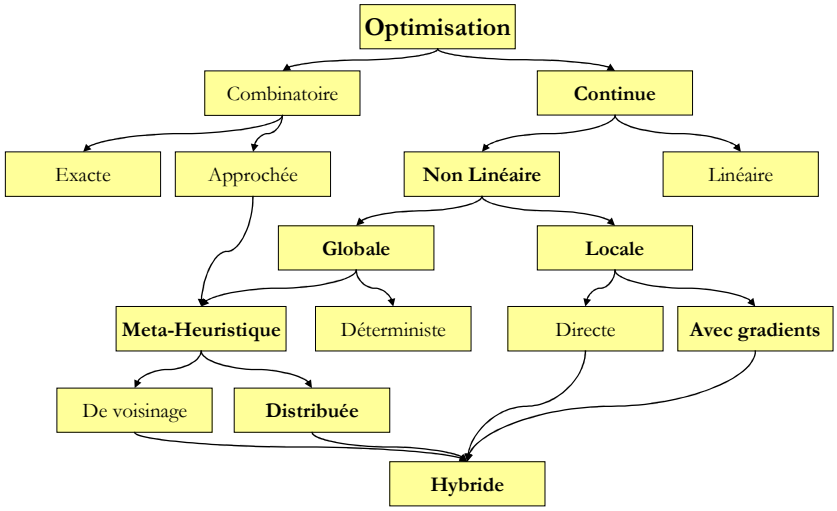

Figure 17 : Une typologie d'algorithmes d'optimisation [NEOS Guide[13]]

Nous définissons la facette de dimensionnement pour un modèle orienté, non fonction du temps. Le couplage de la facette de dimensionnement à l'algorithme d'optimisation sera fonction de la nature des ports (continus / discret) mais aussi de la disponibilité d'un calcul de Jacobien.

[13] www.neos-guide.org

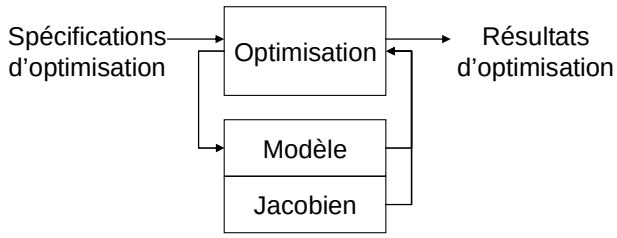

Figure 18 : Couplage de la facette de dimensionnement (modèle + Jacobien) à un algorithme d'optimisation

Le calcul exact des gradients est motivé par le fait que ce calcul, lorsqu'il est possible, conduit à une optimisation plus efficace, car en disposant du gradient, on dispose de plus de connaissance sur le comportement du modèle. Les optimisations utilisant les gradients ont donc de fortes chances d'être plus rapides, ce qui est important pour les phases d'étude de faisabilité dans lesquelles on doit itérer rapidement et de façon aussi fiable que possible.

4.3.2. Description des interfaces

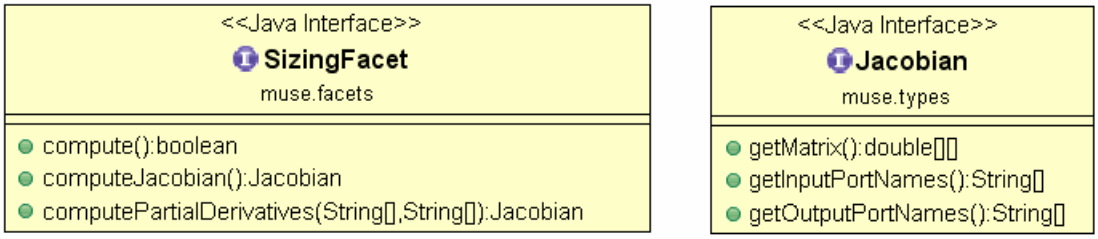

Figure 19 : Diagramme UML de la facette de dimensionnement

4.3.3. Description des méthodes

Méthodes de l'interface « SizingFacet »

Nom de la fonction	Description
`boolean compute()`	**Compute the model and define output ports value.** Returns **false if computation failed, true else.**
`Jacobian computeJacobian()`	Returns **the Jacobian object**
`Jacobian computePartialDerivatives(String[] out, String[] in)`	out **: output port names to differentiate** in **: input port names implied by the derivation** Returns **a Jacobian object, with partial derivatives.**

Méthodes de l'interface « Jacobian »

Nom de la fonction	Description
`double[][] getMatrix()`	Returns **the matrix of values ordered by [outputPortNames][inputPortNames]**
`String[] getInputPortNames()`	Returns **an array with input port names**
`String[] getOutputPortNames()`	Returns **an array with output port names**

4.4. Facette de gestion

Sur le métier « gestion », deux types d'interfaces sont à différencier : les interfaces anticipatives et les interfaces réactives. La première facette permet de réaliser une gestion « anticipative », c'est-à-dire par exemple la veille, pour déterminer à partir d'un modèle prévisionnel quelle sera la stratégie à appliquer dans les prochaines 24h. La deuxième facette est utilisée de manière « réactive » c'est-à-dire en temps réel par rapport aux données courantes pour corriger la stratégie en cours vis-à-vis d'écrat constatés par rapport à la prévision.

4.4.1. Interfaces réactives

4.4.1.1. Description des interfaces

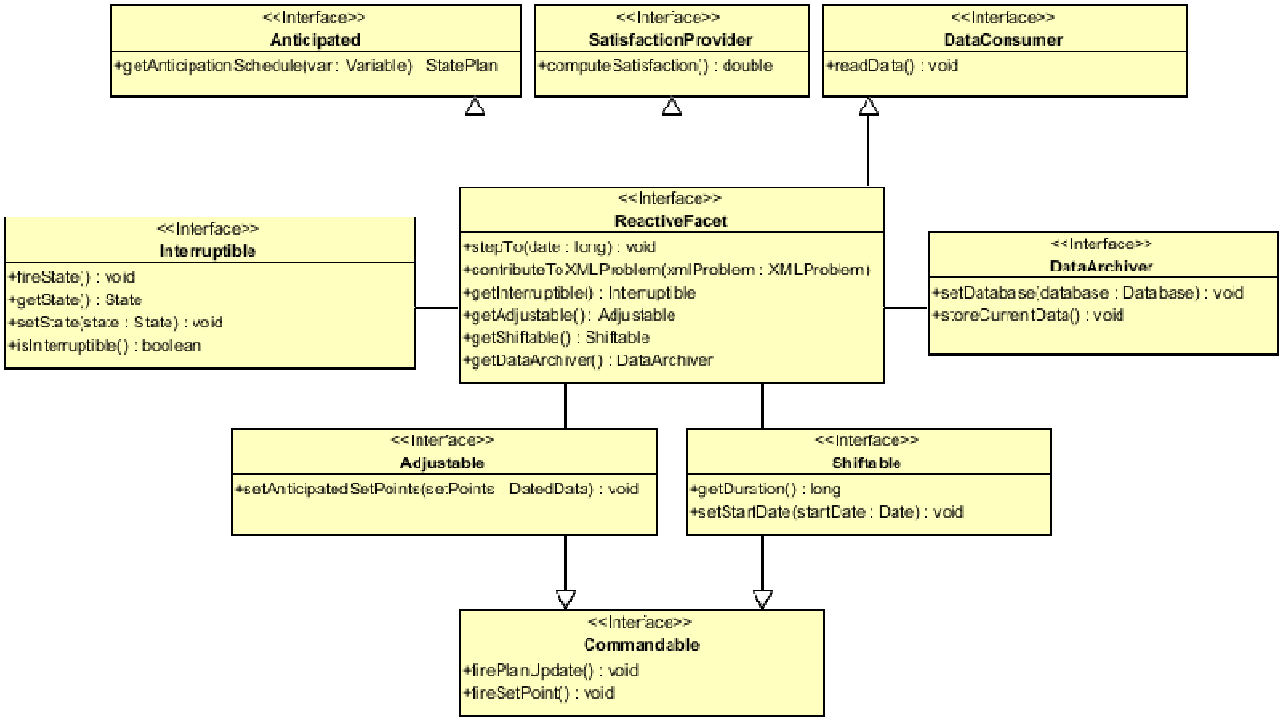

Figure 20 : Diagramme UML de l'interface réactive de la facette de gestion

4.4.1.2. Description des méthodes

Description des méthodes de l'interface Anticipated

Nom de la fonction	Description
`getAnticipationSchedule(var : Variable) : StatePlan`	Getter for the current anticipated schedule of a given variable. variable : the variable to get the anticipation schedule for returns the current anticipated schedule of the variable

Description des méthodes de l'interface SatisfactionProvider

Nom de la fonction	Description
`computeSatisfaction() : double`	Compute the satisfaction value for the service Return satisfaction value : real value 1 and above is perfect and 0 or lower is unacceptable

Description des méthodes de l'interface DataConsumer

Nom de la fonction	Description
`readData() : void`	request available data from remote sensor

Description des méthodes de l'interface ReactiveFacet

Nom de la fonction	Description
`stepTo(date : long) : void`	Go to the next time step
`contributeToXMLProblem(xmlProblem : XMLProblem)`	Gives an XML description of the component model
`getInterruptible() : Interruptible`	Gives functionnalties to interrupt the device consumption
`getAdjustable() : Adjustable`	Gives functionnalities to adjust the device consumption
`getShiftable() : Shiftable`	Gives functionnalities to shift the device consumption
`getDataArchiver() : DataArchiver`	Gives functionnalities to archive data

Description des méthodes de l'interface Interruptible

Nom de la fonction	Description
`fireState() : void`	Propagates device state modifications
`getState() : State`	Gives the device state
`setState(state : State) : void`	Defines the device state
`isInterruptible() : boolean`	Defines if the device is interruptible or not

Description des méthodes de l'interface Adjustable

Nom de la fonction	Description
`+setAnticipatedSetPoints(setPoints : DatedData) : void`	Defines the states planning of the device

Description des méthodes de l'interface Shiftable

Nom de la fonction	Description
`getDuration() : long`	Gives the duration of an active state of the device
`setStartDate(startDate : Date) : void`	Defines when the device has to start

Description des méthodes de l'interface DataArchiver

Nom de la fonction	Description
`setDatabase(database : DatabaseInterface) : void`	Defines database

Description des méthodes de l'interface Commandable

Nom de la fonction	Description
`firePlanUpdate() : void`	Propagates device state planning modifications
`fireSetPoint() : void`	Propagates device state planning definition

4.4.2. Interfaces anticipatives

La facette anticipative decrit un ensemble d'API spécifiques très dépendantes d'une méthode de résolution. Cette méthode traite des problèmes MILP (mixed linear integer programming), c'est-à-dire en programmation linéaire mixte en nombres d'entiers.

4.4.2.1. Description des interfaces

Le détail des méthodes est fourni dans le prochain paragraphe.

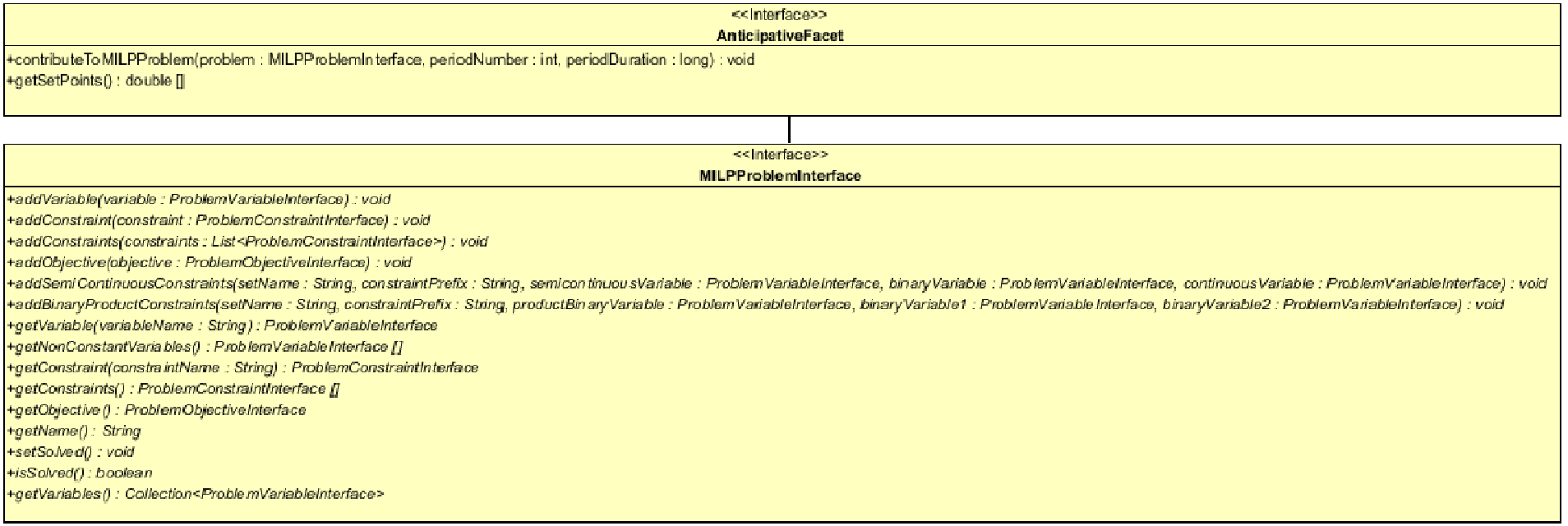

Figure 21 : Diagramme UML de l'interface anticipative de la facette de gestion

4.4.2.2. Description des méthodes

Description des méthodes de l'interface AnticipativeFacet

Nom de la fonction	Description
`contributeToMILPProblem(problem : MILPProblemInterface, periodNumber : int, periodDuration : long) : void`	Visitor for the optimization problem construction. Here is implemented the contribution of the service in the global optimization problem. problem the MILP problem to be updated with this service contribution numberOfPeriods anticipativePeriodDuration in ms
`getSetPoints() : double []`	

Description des méthodes de l'interface MILPProblemInterface

Nom de la fonction	Description
`addVariable(variable : ProblemVariableInterface) : void`	add given variable to this MILP problem. variable, the variable to be added to this problem
`addConstraint(constraint : ProblemConstraintInterface) : void`	add given constraint to this MILP problem. constraint the constraint to be added to this problem
`addObjective(objective : ProblemObjectiveInterface) : void`	add given objective to this MILP problem. objective the objective to be added to this problem
`addSemiContinuousConstraints(setName : String, constraintPrefix : String, semicontinuousVariable : ProblemVariableInterface, binaryVariable : ProblemVariableInterface, continuousVariable : ProblemVariableInterface) : void`	generate the semi-continuous constraints and add them to this MILP problem, ie. constraints such as z=d*x where d is a binary variable, x is a continuous variable and z, the resulting semi-continuous variable setName the name of the constraint set associated with the added constraints. constraintPrefix prefix for the 4 generated constraints (prefix1, prefix2, prefix3, prefix4) semicontinuousVariable z, the semi-continuous variable binaryVariable d, the binary variable continuousVariable x, the continuous variable
`addBinaryProductConstraints(setName : String, constraintPrefix : String, productBinaryVariable : ProblemVariableInterface, binaryVariable1 : ProblemVariableInterface, binaryVariable2 : ProblemVariableInterface) : void`	generate the binary product constraints and add them to this MILP problem, ie. constraints such as z=d1*d2 where d1 and d2 are binary variables and z, the

	resulting binary-product variable setName the name of the constraint set associated with the added constraints. constraintPrefix prefix for the generated constraints productBinaryVariable z, the resulting binary-product variable binaryVariable1 d1, the binary variable binaryVariable2 d2, the binary variable
`getVariable(variableName : String) : ProblemVariableInterface`	retrieve from this problem the variable with given name. variableName the name of the variable to be retrieved
`getConstraint(constraintName : String) : ProblemConstraintInterface`	retrieve from this problem the constraint with given name. constraintName the name of the constraint to be retrieved
`getConstraints() : ProblemConstraintInterface []`	getter for all constraints defined in this MILP problem. return all constraints defined in this problem
`getObjective() : ProblemObjectiveInterface`	getter for the objective defined for this MILP problem. return the objective defined for this MILP problem
`getName() : String`	getter for the name of this MILP problem. return the name of this MILP problem
`setSolved() : void`	make this problem solved.
`isSolved() : boolean`	check if this problem is solved. return true if this problem is solved, false otherwise.
`getVariables() : java.util.Collection<ProblemVariableInterface>`	getter for all variables defined in this MILP problem. return all variables defined in this problem

5. Accès aux ressources

Certains services ne nécessitent pas l'utilisation des API et peuvent réaliser des traitements directement avec des ressources disponibles dans le composant. C'est par exemple le cas d'un modèle disposant des sources Modelica qui seront exploités lors d'une composition. C'est également le cas d'un composant FMI embarqué qui peut être directement exploité comme « boite noire » sans charger de machine virtuelle Java.
Les ressources sont placées par convention dans le dossier « /RES/ » le nom complet de la ressource doit être décrit dans le fichier XML de méta-données ou disponible par les API des facettes pour être utilisé. Les ressources peuvent également être à usage interne uniquement, les facettes les exploitent pour fournir leur service.

6. Annexe 1 : Schéma XSD des méta-données XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" xmlns:museapi="http://museapi/1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" ecore:nsPrefix="museapi" ecore:package="museapi"
targetNamespace="http://museapi/1.0">
  <xsd:import namespace="http://www.eclipse.org/emf/2002/Ecore"
schemaLocation="platform:/plugin/org.eclipse.emf.ecore/model/Ecore.xsd"/>
  <xsd:element ecore:ignore="true" name="Component" type="museapi:Component"/>
  <xsd:element ecore:ignore="true" name="ComponentPort" type="museapi:ComponentPort"/>
  <xsd:element ecore:ignore="true" name="ApplicationFacet" type="museapi:ApplicationFacet"/>
  <xsd:element ecore:ignore="true" name="FacetPort" type="museapi:FacetPort"/>
  <xsd:element ecore:ignore="true" name="ValueDomain" type="museapi:ValueDomain"/>
  <xsd:element ecore:ignore="true" name="Origin" type="museapi:Origin"/>
  <xsd:element ecore:ignore="true" name="ComponentPortConnector"
type="museapi:ComponentPortConnector"/>
  <xsd:element ecore:ignore="true" name="ComponentPortNature" type="museapi:ComponentPortNature"/>
  <xsd:element ecore:ignore="true" name="Muse" type="museapi:Muse"/>
  <xsd:element ecore:ignore="true" name="PhysicalNatureType" type="museapi:PhysicalNatureType"/>
  <xsd:element ecore:ignore="true" name="RealValueDomain" type="museapi:RealValueDomain"/>
  <xsd:element ecore:ignore="true" name="IntegerValueDomain" type="museapi:IntegerValueDomain"/>
  <xsd:element ecore:ignore="true" name="BooleanValueDomain" type="museapi:BooleanValueDomain"/>
  <xsd:element ecore:ignore="true" name="StringValueDomain" type="museapi:StringValueDomain"/>
  <xsd:element ecore:ignore="true" name="BinaryValueDomain" type="museapi:BinaryValueDomain"/>
  <xsd:element ecore:ignore="true" name="QuantitativeValueDomain"
type="museapi:QuantitativeValueDomain"/>
  <xsd:element ecore:ignore="true" name="QualitativeValueDomain"
type="museapi:QualitativeValueDomain"/>
  <xsd:element ecore:ignore="true" name="SimulationFacet" type="museapi:SimulationFacet"/>
  <xsd:element ecore:ignore="true" name="SizingFacet" type="museapi:SizingFacet"/>
  <xsd:element ecore:ignore="true" name="ReactiveFacet" type="museapi:ReactiveFacet"/>
  <xsd:element ecore:ignore="true" name="AnticipativeFacet" type="museapi:AnticipativeFacet"/>
  <xsd:element ecore:ignore="true" name="ParameterEstimation" type="museapi:ParameterEstimation"/>
  <xsd:element ecore:ignore="true" name="Composite" type="museapi:Composite"/>
  <xsd:complexType name="Component">
    <xsd:sequence>
      <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" name="ports"
type="museapi:ComponentPort"/>
      <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" name="facets"
type="museapi:ApplicationFacet"/>
    </xsd:sequence>
    <xsd:attribute ecore:id="true" ecore:name="ID" ecore:unsettable="false" name="ID"
type="ecore:ELong"/>
    <xsd:attribute name="modelType" type="ecore:EString"/>
  </xsd:complexType>
  <xsd:complexType name="ComponentPort">
    <xsd:attribute ecore:reference="museapi:ComponentPortNature" name="nature" type="xsd:anyURI"
use="required"/>
    <xsd:attribute ecore:name="ID" ecore:unsettable="false" name="ID" type="ecore:ELong"/>
    <xsd:attribute name="description" type="ecore:EString"/>
    <xsd:attribute ecore:opposite="componentPort" ecore:reference="museapi:FacetPort"
name="facetPorts" use="required">
      <xsd:simpleType>
        <xsd:list itemType="xsd:anyURI"/>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="ApplicationFacet">
    <xsd:sequence>
      <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" name="ports"
type="museapi:FacetPort"/>
      <xsd:element ecore:resolveProxies="true" minOccurs="0" name="origin" type="museapi:Origin"/>
    </xsd:sequence>
    <xsd:attribute name="modelDescription" type="ecore:EString"/>
  </xsd:complexType>
  <xsd:complexType name="FacetPort">
    <xsd:sequence>
      <xsd:element ecore:resolveProxies="true" name="valueDomain" type="museapi:ValueDomain"/>
    </xsd:sequence>
    <xsd:attribute ecore:opposite="facetPorts" ecore:reference="museapi:ComponentPort"
name="componentPort" type="xsd:anyURI" use="required"/>
    <xsd:attribute ecore:unsettable="false" name="orientation" type="museapi:Orientation"/>
    <xsd:attribute ecore:unsettable="false" name="isTimeDependant" type="ecore:EBoolean"/>
    <xsd:attribute name="nameInModel" type="ecore:EString"/>
```

```
    <xsd:attribute ecore:unsettable="false" name="isContrainedInput" type="ecore:EBoolean"/>
  </xsd:complexType>
  <xsd:complexType name="ValueDomain">
    <xsd:attribute ecore:unsettable="false" name="dataFormat" type="museapi:DataFormat"/>
  </xsd:complexType>
  <xsd:complexType name="Origin">
    <xsd:sequence>
      <xsd:element ecore:unique="true" maxOccurs="unbounded" minOccurs="0" name="sourceFiles"
nillable="true" type="ecore:EString"/>
    </xsd:sequence>
    <xsd:attribute ecore:unsettable="false" name="language" type="museapi:Language"/>
    <xsd:attribute ecore:unsettable="false" name="software" type="museapi:Software"/>
    <xsd:attribute name="version" type="ecore:EString"/>
    <xsd:attribute name="author" type="ecore:EString"/>
  </xsd:complexType>
  <xsd:complexType name="ComponentPortConnector">
    <xsd:attribute name="description" type="ecore:EString"/>
    <xsd:attribute ecore:reference="museapi:ComponentPort" name="ports" use="required">
      <xsd:simpleType>
        <xsd:restriction>
          <xsd:simpleType>
            <xsd:list itemType="xsd:anyURI"/>
          </xsd:simpleType>
          <xsd:minLength value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="ComponentPortNature">
    <xsd:sequence>
      <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" minOccurs="0"
name="physicalNature" type="museapi:PhysicalNatureType"/>
    </xsd:sequence>
    <xsd:attribute ecore:id="true" ecore:name="ID" ecore:unsettable="false" name="ID"
type="ecore:ELong"/>
    <xsd:attribute name="description" type="ecore:EString"/>
  </xsd:complexType>
  <xsd:complexType name="Muse">
    <xsd:complexContent>
      <xsd:extension base="museapi:Composite">
        <xsd:sequence>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" minOccurs="0"
name="natures" type="museapi:ComponentPortNature"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="PhysicalNatureType">
    <xsd:attribute name="physicalNature" type="ecore:EString"/>
    <xsd:attribute name="defaultUnit" type="ecore:EString"/>
  </xsd:complexType>
  <xsd:simpleType name="Language">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="modelica"/>
      <xsd:enumeration value="ampl"/>
      <xsd:enumeration value="other"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="Software">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="trnsys"/>
      <xsd:enumeration value="dymola"/>
      <xsd:enumeration value="energyplus"/>
      <xsd:enumeration value="ccmi"/>
      <xsd:enumeration value="fmi"/>
      <xsd:enumeration value="other"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="Orientation">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="input"/>
      <xsd:enumeration value="output"/>
      <xsd:enumeration value="none"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="DataFormat">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="scalar"/>
```

```
      <xsd:enumeration value="vector"/>
      <xsd:enumeration value="matrix"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType ecore:implements="museapi:QuantitativeValueDomain" name="RealValueDomain">
    <xsd:complexContent>
      <xsd:extension base="museapi:ValueDomain">
        <xsd:attribute ecore:unsettable="false" name="min" type="ecore:EDouble"/>
        <xsd:attribute ecore:unsettable="false" name="max" type="ecore:EDouble"/>
        <xsd:attribute ecore:unsettable="false" name="default" type="ecore:EDouble"/>
        <xsd:attribute ecore:unsettable="false" name="initial" type="ecore:EDouble"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType ecore:implements="museapi:QuantitativeValueDomain" name="IntegerValueDomain">
    <xsd:complexContent>
      <xsd:extension base="museapi:ValueDomain">
        <xsd:attribute ecore:unsettable="false" name="min" type="ecore:EDouble"/>
        <xsd:attribute ecore:unsettable="false" name="max" type="ecore:EDouble"/>
        <xsd:attribute ecore:unsettable="false" name="default" type="ecore:EDouble"/>
        <xsd:attribute ecore:unsettable="false" name="initial" type="ecore:EDouble"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType ecore:implements="museapi:QualitativeValueDomain" name="BooleanValueDomain">
    <xsd:complexContent>
      <xsd:extension base="museapi:ValueDomain"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType ecore:implements="museapi:QualitativeValueDomain" name="StringValueDomain">
    <xsd:complexContent>
      <xsd:extension base="museapi:ValueDomain">
        <xsd:sequence>
          <xsd:element ecore:unique="true" maxOccurs="unbounded" minOccurs="2" name="tag"
nillable="true" type="ecore:EString"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="BinaryValueDomain">
    <xsd:complexContent>
      <xsd:extension base="museapi:IntegerValueDomain"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="QuantitativeValueDomain">
    <xsd:complexContent>
      <xsd:extension base="museapi:ValueDomain"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="QualitativeValueDomain">
    <xsd:complexContent>
      <xsd:extension base="museapi:ValueDomain"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="SimulationFacet">
    <xsd:complexContent>
      <xsd:extension base="museapi:ApplicationFacet">
        <xsd:sequence>
          <xsd:element ecore:unique="true" maxOccurs="unbounded" name="supportedSolvers"
type="museapi:SolverType"/>
        </xsd:sequence>
        <xsd:attribute ecore:unsettable="false" name="modellingLevel" type="museapi:ModellingLevel"
use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="ModellingLevel">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="detailled"/>
      <xsd:enumeration value="average"/>
      <xsd:enumeration value="coarse"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="SolverType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration ecore:value="3" value="hybrid"/>
      <xsd:enumeration ecore:value="0" value="discreteTime"/>
      <xsd:enumeration ecore:value="1" value="ode"/>
```

```
      <xsd:enumeration ecore:value="2" value="dae"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="SizingFacet">
    <xsd:complexContent>
      <xsd:extension base="museapi:ApplicationFacet"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="ReactiveFacet">
    <xsd:complexContent>
      <xsd:extension base="museapi:ApplicationFacet">
        <xsd:attribute name="modelType" type="ecore:EString"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="AnticipativeFacet">
    <xsd:complexContent>
      <xsd:extension base="museapi:ApplicationFacet">
        <xsd:attribute name="modelType" type="ecore:EString"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="ParameterEstimation">
    <xsd:complexContent>
      <xsd:extension base="museapi:ApplicationFacet">
        <xsd:attribute name="modelType" type="ecore:EString" use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="Composite">
    <xsd:complexContent>
      <xsd:extension base="museapi:Component">
        <xsd:sequence>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" name="innerComponents"
type="museapi:Component"/>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" minOccurs="0"
name="connectors" type="museapi:ComponentPortConnector"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```