Obeo SmartEA - Guide de personnalisation

v8.3.0


OBEO
SmartEA

1. Introduction

Ce manuel est à destination des personnes susceptibles de recourir aux personnalisations offertes par Obeo SmartEA.

2. Pré-requis

Les pré-requis permettant d'exploiter l'ensemble des personnalisations avancées de Obeo SmartEA sont les suivants :

- Java : niveau de programmation standard
- EMF (CDO) : connaissances de base (création d'un méta-modèle, instanciation/manipulation du méta-modèle)

Pour pouvoir exploiter les API SmartEA, vous devez installer l'environnement de développement Obeo SmartEA Developer.

3. Installer SmartEA Developer

SmartEA Developer est basé sur la plateforme Eclipse et vous permet d'utiliser facilement les API de SmartEA.

Pour l'utiliser, vous devez dézipper l'archive nommée `Obeo-SmartEA-Developer.zip`.

Avant de démarrer SmartEA Developer, pensez à vérifier que vous disposez d'une machine virtuelle Java 17 64 bits. Si ce n'est pas votre version par défaut, il vous faut préciser le chemin pour la trouver dans le fichier `Obeo-SmartEA-Developer.ini`

Pour utiliser SmartEA Developer vous devez également disposer :

- d'une licence SmartEA Developer valide. Cette licence permet d'utiliser SmartEA Developer.
- d'une licence Serveur SmartEA Developer valide. Cette licence permet de démarrer un serveur local depuis SmartEA Developer pour tester vos développements.

3.1. Licence SmartEA Developer

La licence SmartEA Developer peut être d'un des deux types suivants selon l'option que vous avez retenue :

- Licence poste
- Licence utilisant un serveur de jetons (licence flottante).

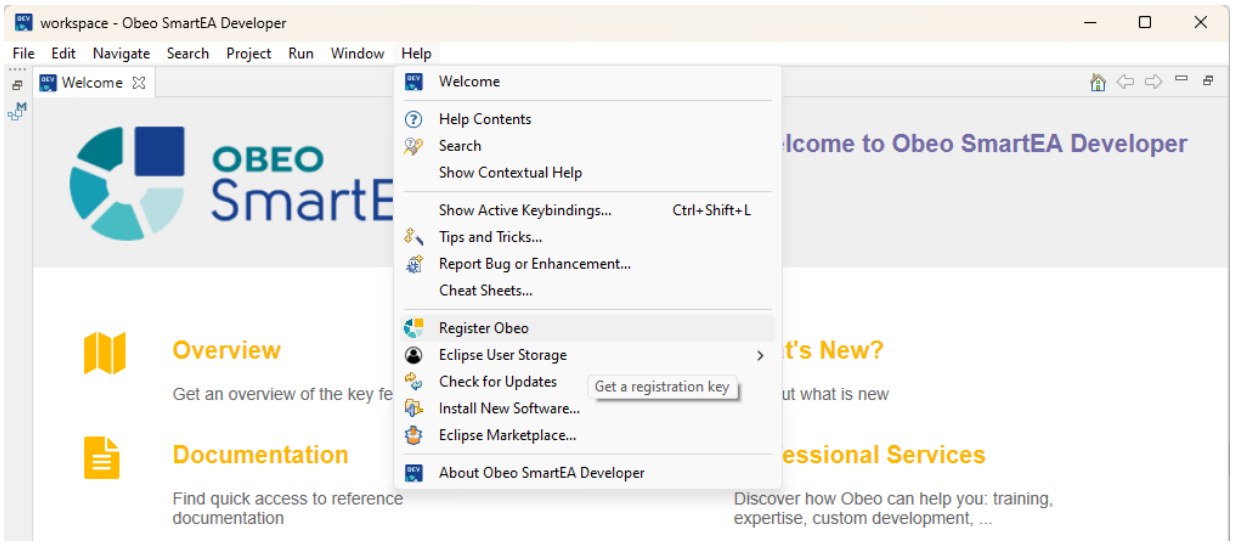
3.1.1. Licence poste

Si l'option retenue est une licence poste, vous devez générer une clé, puis installer la licence poste fournie par Obeo.

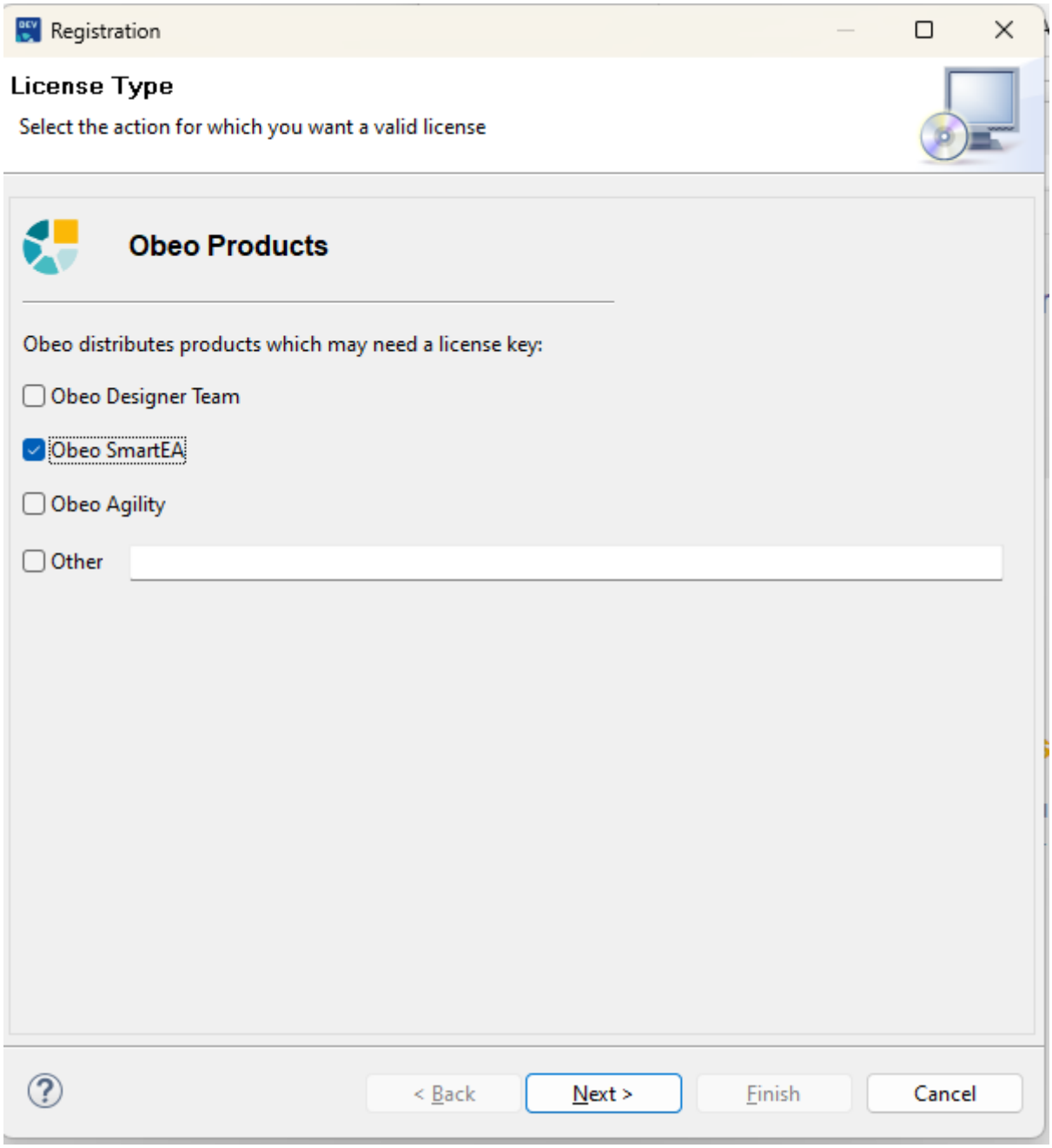
3.1.1.1. Génération d'une clé

Pour générer une clé de licence :

- Démarrez SmartEA Developer,
- Ouvrez le menu `Help` puis
- Cliquez sur `Register Obeo`.



Dans le wizard qui s'affiche, sélectionnez SmartEA, puis cliquez sur le bouton `Next`.



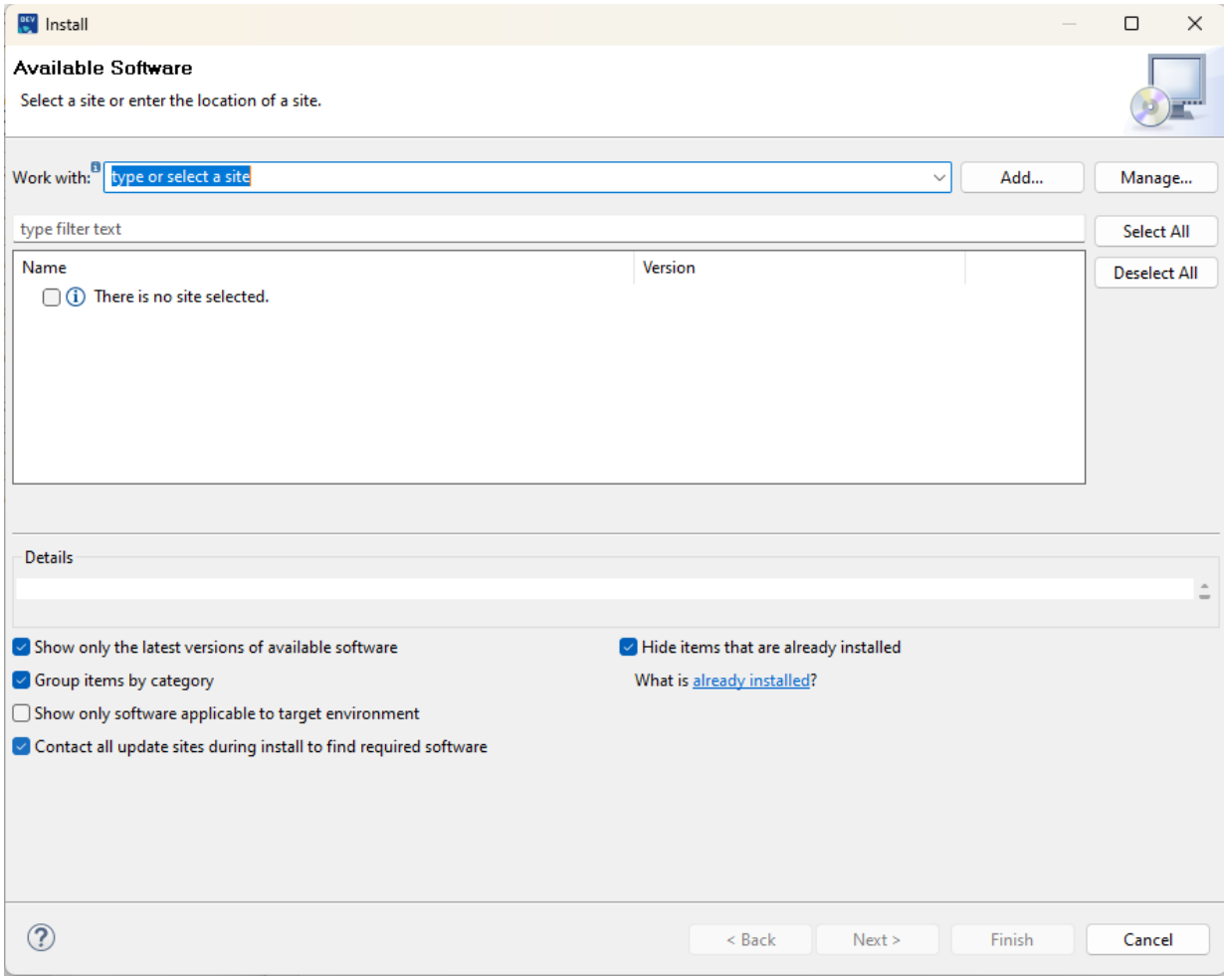
Renseignez ensuite toutes les informations demandées, puis cliquez sur le bouton `Next`.

Votre logiciel de messagerie s'ouvre et un email est prérempli avec les informations fournies. Si ce n'est pas le cas, créez un nouvel email ayant pour destinataire registration@obeo.fr et copiez la clé générée (REGISTRATION KEY) dans le corps de l'email.

Pour finir, envoyez l'email. Vous recevrez en retour une licence à installer sur votre poste.

3.1.1.2. Installation d'une licence poste

Une fois que vous disposez d'une licence poste, installez la. Pour cela, cliquez sur le menu `Help` puis sur `Install New Software...`.



Cliquez sur le bouton `Add`, puis le bouton `Local...`. Sélectionnez la licence qui vous a été envoyée, puis installez la.

3.1.2. Serveur de jetons

Si l'option retenue est un serveur de jetons, vous devez installer le serveur puis configurer les postes des utilisateurs.

3.1.2.1. Installation du serveur de jetons

Pour installer un serveur de jetons, reportez vous à la procédure d'installation (fichier DOCUMENTATION.txt) qui se trouve dans l'archive du serveur de licences.

3.1.2.2. Configuration des postes utilisateur

Afin que les SmartEA Developer utilisent le serveur de jetons, les utilisateurs doivent éditer le fichier `Obeo-SmartEA-Developer.ini` et ajouter en fin de fichier la clé fournie par Obeo.

Le paramètre permettant de préciser la clé est `OBEO_LICENSE_SERVER_CONFIGURATION`.

Ainsi votre fichier `Obeo-SmartEA-Developer.ini` doit contenir une ligne ayant la forme suivante :

```
-DOBEO_LICENSE_SERVER_CONFIGURATION=69868.......
```

3.2. Licence Serveur SmartEA

SmartEA Developer contient contient une licence Serveur SmartEA d'évaluation d'une validité de deux mois. Elle doit être remplacée.

1. Obtention d'une licence

Envoyez un email à registration@obeo.fr indiquant l'adresse MAC de votre machine ainsi que le login de l'utilisateur qui démarre le SmartEA Developer.

Obeo vous transmettra en retour un fichier nommé `smartEALicense.key.zip`.

2. Installation de la licence

Pour installer la licence :

- Arrêtez le serveur SmartEA si celui-ci est démarré.
- Décompressez le fichier `smartEALicense.key.zip` envoyé par Obeo.
- Remplacez le fichier `smartEALicense.key` obtenu dans le répertoire `etc/` qui se trouve dans le projet `fr.obeo.smartea.core.server.webapp` de votre workspace.

Pour vérifier que la licence est bien installée, vous pouvez consulter la page d'administration principale du serveur une fois celui-ci démarré.

4. Workspace

SmartEA Developer s'ouvre sur un workspace permettant de lancer un serveur et un modeleur grâce à deux launch configs.

Le workspace est organisé en Working Sets :

- Le premier Working Set, nommé `Odesigns`, contient les odesigns nécessaires (ArchiMate, BPMN et RGPD) pour lancer un serveur de démonstration par launch config.
- Le second, nommé `Utilities`, contient :
  - un plugin permettant de configurer le serveur.
  - un projet contenant les launch configs du serveur et du modeleur.

Le contenu du Working Set `Utilities` ne doit pas être supprimé. Si un des deux projet est supprimé vous ne pourrez plus lancer un serveur et un modeleur depuis SmartEA Developer.

Pour déployer votre code, inutile de builder les modules et de produire les update sites. Ajoutez simplement vos plugins dans le workspace. Une fois cela fait, ajoutez vos plugins commons et server dans la launch config du serveur. De même, ajoutez vos plugins commons et rcp dans la launch config du modeleur.

Si vous avez des plugins contenant des odesigns, éditez le fichier `fr.obeo.smartea.core.server.webapp/etc/applicationdev.conf` afin d'y ajouter le chemin vers les odesign de votre workspace (paramètre `odesign.discovery`).

Les plugins de odesign n'ont pas à se trouver dans la launch config du serveur. Ils doivent uniquement être déclarés dans ce fichier de configuration.

Une fois tout cela fait vous pouvez démarrer un serveur puis un modeleur grâce aux launch configs.

Le répertoire `fr.obeo.smartea.core.server.webapp/etc/` contient les fichiers de configuration classiques d'un serveur. C'est dans ce répertoire qu'il faut déposer votre licence serveur de tests.

Le répertoire `fr.obeo.smartea.core.server.webapp/data/run/` contient les bases H2 et les indexes Lucene. Si vous avez besoin de repartir sur un serveur sans données, effacez les répertoires :

- `data/run/content` et
- `data/run/internal`.

Si vous ne souhaitez pas utiliser des bases H2 mais une base Postgresql vous avez simplement à configurer les fichiers de configuration des projets qui se trouvent dans le répertoire `etc/projects`.

5. Créer un connecteur SmartEA

Il est possible de se connecter au référentiel SmartEA depuis un programme extérieur, en lecture ou en écriture. Pour écrire un connecteur de ce type, vous pouvez utiliser l'API SmartEA associée : un `SmartEAClient` qui permet de se connecter à un serveur SmartEA, et un `SmartEAProjectClient` qui permet de se connecter à un projet spécifique et d'y appliquer des modifications.

Une fois l'IDE SmartEA démarré, choisissez la perspective Java et créez un projet Java :

- Menu "`File > New`" puis "`Plug-in project`"
- Choisissez un nom pour votre plug-in (ex : `com.mycompany.connector`)
- Dans la zone "`Target Platform`", choisissez "`Eclipse version / 3.5 and greater`"
- Cliquez sur "`Finish`"
- Ouvrez le fichier "`META-INF/MANIFEST.MF`" en mode texte et ajoutez les dépendances suivantes :

```
org.apache.commons.codec,
org.apache.commons.logging,
org.eclipse.net4j.tcp,
org.eclipse.emf.cdo.net4j,
fr.obeo.smartea.core.basemm,
fr.obeo.smartea.core.server.client,
fr.obeo.smartea.core.prism,
fr.obeo.smartea.core.prefs,
fr.obeo.smartea.core.server.ext,
fr.obeo.smartea.core.server.cdo,
org.eclipse.jetty.websocket.api,
org.eclipse.jetty.websocket.core.client,
org.eclipse.jetty.websocket.common,
org.eclipse.jetty.websocket.core.common,
org.eclipse.jetty.io,
org.eclipse.jetty.http,
aopalliance,
fr.obeo.smartea.core.refmodel.api
```

Ce projet dispose maintenant des dépendances requises pour l'implémentation d'un connecteur. Vous pouvez maintenant l'implémenter.

Par exemple, le main Java ci-dessous permet d'ajouter des répertoires dans le référentiel de SmartEA :

```
import java.io.IOException;

import org.eclipse.emf.cdo.eresource.CDOResource;
import org.eclipse.emf.cdo.util.CommitException;

import fr.obeo.smartea.core.basemm.BaseFactory;
import fr.obeo.smartea.core.basemm.Folder;
import fr.obeo.smartea.core.common.api.client.SmartEAProtocol;
import fr.obeo.smartea.core.server.api.db.DbProjectContentCtx;
import fr.obeo.smartea.core.server.api.db.DbProjectContentTx;
import fr.obeo.smartea.core.server.client.SmartEAClient;
import fr.obeo.smartea.core.server.client.SmartEAProjectClient;

public class SampleConnector {

   public static void main(String[] args) throws IOException, CommitException {
      // Initialize SmartEA Client
      SmartEAClient smartEAClient = new SmartEAClient(SmartEAProtocol.HTTP, // Protocol
            "localhost", // HTTP host
            8080, // HTTP port
            "afontaine", // USer ID
            "123", // Password
            true // Read only mode for internal db context
      );
```

```
      SmartEAProjectClient sandboxClient = smartEAClient.newProjectClient("voyagediscount",
            // Write mode
            false);

      // Retrieve the semantic resource
      DbProjectContentCtx ctx = sandboxClient.getProjectContentCtx();
      DbProjectContentTx tx = ctx.getTx(ctx.getMasterBranch());
      CDOResource semanticResource = tx.getSemanticResource();

      // Create a new folder
      Folder newFolder = BaseFactory.eINSTANCE.createFolder();
      newFolder.setName("NewFolder_" + System.currentTimeMillis());

      // Add it to the semantic resource
      semanticResource.getContents().add(newFolder);

      // Apply the change
      sandboxClient.commit();

      // Release context
      sandboxClient.release();
      smartEAClient.release();
   }
}
```

En partant de ce principe, vous pouvez réaliser un connecteur qui ajoute des informations au référentiel depuis un fichier Excel, une source de données externe, et lancer le processus de façon régulière.

Si le connecteur change l'état du référentiel, il est conseillé de le lancer pendant une période d’inactivité des utilisateurs. Sinon, il vous faudra gérer le cas ou un ou plusieurs objets sont verrouillés.

5.1. Activer des services AQL dans un connecteur

Tout service AQL appartenant à un module SmartEA (Modules développés par Obeo inclus) doit être enregistré dans le connecteur pour pouvoir être reconnu et executé. Par exemple afin de pouvoir évaluer les valeurs de propriétés calculées.

Par exemple pour enregistrer une classe de service nommée `Services` :

```
   Services instanceDeService = new Services();
   SmartEAClientReferenceModel referenceModel = (SmartEAClientReferenceModel)
 SmartEAClientReferenceModelProvider.INSTANCE.get();
   referenceModel.injectToInstance(instanceDeService);
   referenceModel.getAQLEnvironment().registerInstanceServices(instanceDeService);
```

6. Créer un module SmartEA

Un module SmartEA est un ensemble de plug-ins permettant d'étendre Obeo SmartEA, pour ajouter par exemple le support d'un méta-modèle spécifique, ou encore des représentations, des générateurs Acceleo... ou tout autre extension supportée par l'API SmartEA.Cette section décrit comment développer et déployer un module à l'aide de l'IDE SmartEA.

6.1. Principes généraux

6.1.1. Structure d'un module

Les plug-ins développés à destination des modeleurs (méta-modèle, odesign, ...) et du serveur sont regroupés sous forme d'update-site P2. Ces update-sites sont alors déposés (préalablement dézippés) sur le serveur dans le répertoire `modules` (avec un sous-répertoire par module).

Les plug-ins à destination du modeleur seront installés par une procédure automatisée.

L'update-site d'un module devra inclure généralement 3 features :

6.1.1.1. La feature modeleur

Une feature dont le nom contient la chaîne `rcp`, destinée à être déployée sur les modeleurs. Elle référencera les plug-ins qui utilisent l'API modeleur de SmartEA.

6.1.1.2. La feature serveur

Une feature dont le nom contient la chaîne `server` qui sera déployée sur le serveur. Elle référencera les plug-ins qui utilisent l'API serveur de SmartEA.

6.1.1.3. La feature commune

une feature `common`, dont dépendront les features `server` et `rcp` et qui contiendra le code factorisable entre les plug-ins destinés au serveur et au modeleur. Elle référencera les plug-ins qui utilisent l'API commune de SmartEA.

Le nommage de la feature `common` est ici seulement une recommandation, les points d'entrées utilisés par Obeo SmartEA pour alimenter serveur & modeleur étant les features `server` & `rcp`. Si ces dernières requièrent d'autres features (présentes sur l'update-site) pour fonctionner, elles seront installées lors du lancement du serveur et des modeleurs.

6.1.2. Création d'un plug-in

Pour implémenter une extension, il vous faut au moins un plug-in :

- Menu "`File > New`" puis "`Plug-in project`"
- Choisissez un nom pour votre plug-in (ex : `com.mycompany.myextensions`)
- Dans la zone "`Target Platform`", choisissez "`Eclipse version / 3.5 and greater`"
- Cliquez sur "`Finish`"
- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`"
- Sélectionnez l'onglet "`Dependencies`"
- Dans la zone "`Required Plug-in`", ajoutez "`fr.obeo.smartea.core.server.api`" si vous voulez contribuer au serveur, "`fr.obeo.smartea.core.common.api`" si vous voulez contribuer à l'API commune ou "`fr.obeo.smartea.core.rcp.api`" si vous voulez contribuer au modeleur
- Sélectionnez l'onglet "`Overview`", puis cochez la case "`This plug-in is a singleton`"
- Sauvez

6.1.3. Exemple de module

L'IDE SmartEA fournit un exemple de module, accessible par le menu "`File > New > Example > SmartEA Examples > SmartEA Module`". Ce wizard permet d'extraire un module constitué des projets suivants :

- `fr.obeo.smartea.sample.common.extension` : un plug-in pour héberger des extensions de l'API common
- `fr.obeo.smartea.sample.server.extension` : un plug-in pour héberger des extensions de l'API server
- `fr.obeo.smartea.sample.rcp.extension` : un plug-in pour héberger des extensions de l'API rcp
- `fr.obeo.smartea.sample.common.feature` : une feature qui référence les projets d'extensions de l'API common
- `fr.obeo.smartea.sample.server.feature` : une feature qui référence les projets d'extensions de l'API server
- `fr.obeo.smartea.sample.rcp.feature` : une feature qui référence les projets d'extensions de l'API rcp
- `fr.obeo.smartea.sample.update` : un update-site qui permet de construire le module

Ce projet en l'état ne contribue à aucun point d'extension de SmartEA mais peut servir de base pour la création d'un nouveau module.

Pour construire le module, ouvrez le fichier `site.xml` du projet `fr.obeo.smartea.sample.update`, puis utilisez l'action *Build All*. Cela produira :

1. un répertoire `features`
2. un répertoire `plugins`
3. un fichier `artifacts.jar`
4. un fichier `content.jar`

Pour déployer le module dans un serveur, il suffira de copier ces éléments dans un répertoire <*nom de votre module*> dans le répertoire `modules` du serveur (préalablement éteint).

6.1.4. Injection de dépendance

Certaines extensions permettent d'accéder aux données contenues dans le référentiel SmartEA. Par exemple, la gestion d'accès aux artefacts, la politique d'écoute des artefacts, ou encore les services HTTP.

Dans Obeo SmartEA, l'injection de dépendance se fait en utilisant Google Guice. Pour recourir à l'injection de dépendance, le plug-in d'API importe pour vous la dépendance vers le plug-in `com.google.inject`.

Il est ensuite possible dans votre composant d'injecter le contexte qui vous permettra d'accéder aux données désirées, à savoir :

- `fr.obeo.smartea.core.server.api.db.DbInternalCtx` : contexte de transaction permettant d'accéder aux données du référentiel interne SmartEA (Javadoc [*api/index.html?fr/obeo/smartea/core/server/api/db/DbInternalCtx.html*])
- `fr.obeo.smartea.core.server.api.db.DbProjectInternalCtx` : contexte de transaction permettant d'accéder aux données de la zone interne du projet courant (Javadoc [*api/index.html?fr/obeo/smartea/core/server/api/db/DbProjectInternalCtx.html*])
- `fr.obeo.smartea.core.server.api.db.DbProjectContentCtx` : contexte de transaction permettant d'accéder aux données de la zone de contenu du projet courant (Javadoc [*api/index.html?fr/obeo/smartea/core/server/api/db/DbProjectContentCtx.html*])

Pour terminer, il vous faudra ajouter une annotation `com.google.inject.Inject` pour indiquer à Guice de procéder à l'injection.

Voici un exemple de lecture de données du référentiel SmartEA interne et d'écriture de données projet :

```
public class MyExtension implements [...] {
   @Inject
   private InternalCtx<CDOView> internalCtx;

   @Inject
   private ProjectContentCtx<CDOTransaction> projectContentCtx;
```

```
   public void anyMethod() {
      [...]
      // A readonly access...
      internalCtx.getTx().getObject([...])
      [...]
      // An update
      projectContentCtx.getTx().createResource([...]);
      projectContentCtx.getTx().commit();
      [...]
   }
}
```

6.1.5. Gestion des logs

Vous êtes encouragé à utiliser le mécanisme de traces proposé par Obeo SmartEA dans vos extensions (modeleur, serveur ou même en mode standalone). De cette façon, votre code commun se comportera de la même manière qu'il soit embarqué côté serveur, client ou dans un programme standalone.

L'API de logging d'Obeo SmartEA est disponible ici [*api/index.html?fr/obeo/smartea/core/common/api/log/package-frame.html*] .

Par exemple, voici comment utiliser l'API depuis l'exemple précédent :

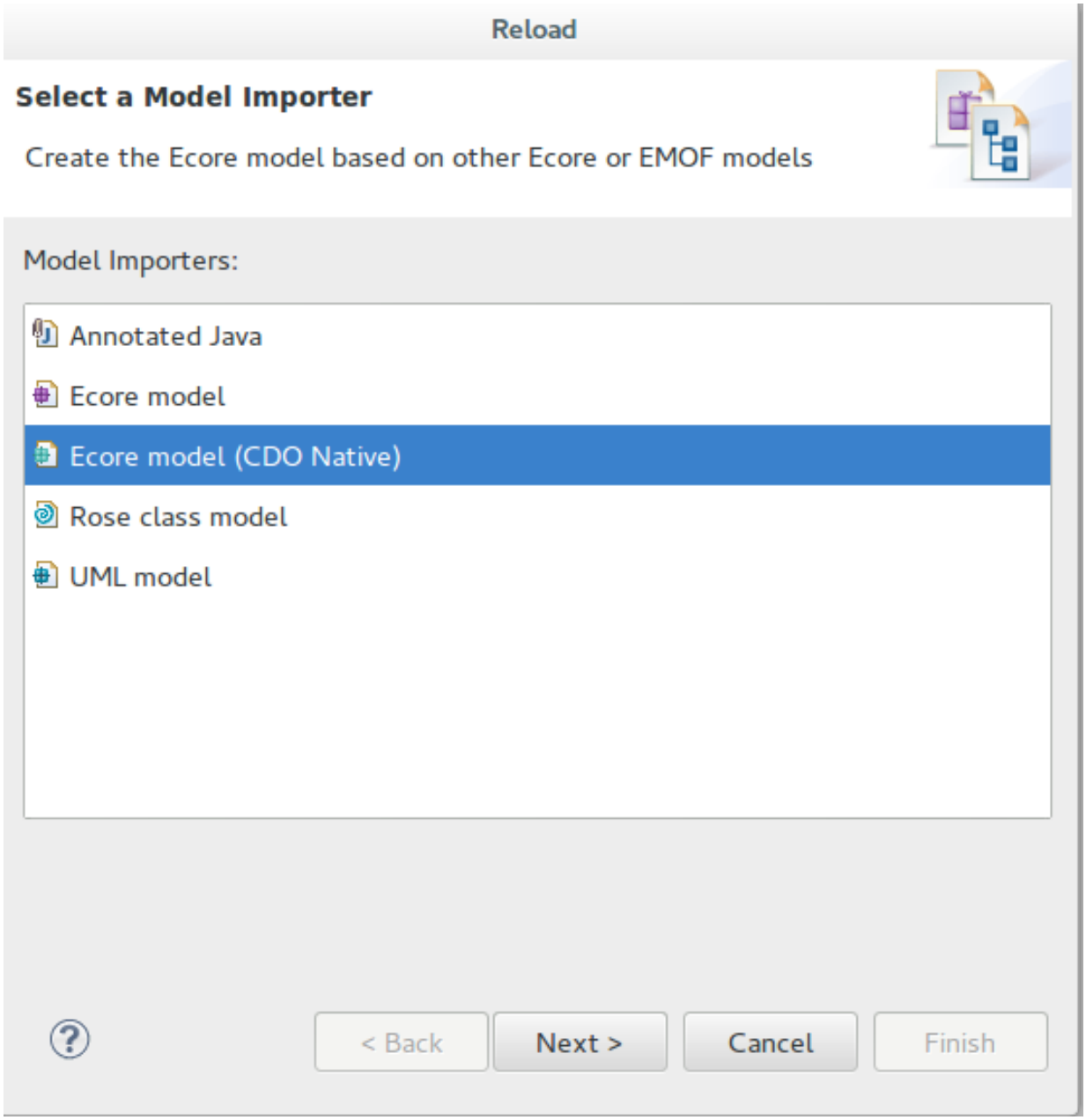
```
   Log log = LogFactory.getLog(MyExtension.class);
   log.warn([...]);
```

6.2. Personnaliser le méta-modèle

Obeo SmartEA permet d'utiliser des méta-modèles standards (Archimate, ...) ou des méta-modèles construits sur mesure.Dans les deux cas, le méta-modèle doit vérifier certains pré-requis pour pouvoir être exploité avec SmartEA.

6.2.1. Compatibilité CDO

Pour utiliser votre propre méta-modèle EMF, le seul pré-requis est de générer le code EMF en compatibilité CDO.Pour ce faire, le fichier .gen-model de votre méta-modèle doit être en mode «cdo-native».



6.2.2. Relations inverses

Attention :

Il est permis de définir des références inverses (`eOpposite`) dans votre méta-modèle EMF avec la restriction suivante : **Il est formellement interdit de définir des références inverses pour les références de contenance**.

Si vous avez le besoin de références inverses sur les relations de contenance, vous pouvez définir une référence dérivée, une `eOperation` ou une relation dérivée SmartEA.

6.2.3. Gestion des identifiants d'objets

L'ID logique est un concept propre à Obeo SmartEA et permet d'identifier un objet sur des branches différentes.Pour des problématiques de performances, il est obligatoire que toutes vos EClass disposent d'un EMF ID.

En effet, Obeo SmartEA en tire parti sous la notion d'ID logique et permet de :

1. comparer, de fusionner les objets sémantiques avec l'ID logique
2. récupérer rapidement l'ID logique
3. conserver l'historique des changements de l'ID logique
4. reconstruire de zéro l'index des ID Logiques

Pour affecter un EMF ID à une EClass de votre méta-modèle, vous devez avoir un EAttribute ayant les propriétés suivantes :

1. ID = true
2. Unique = true

Il est ensuite nécessaire d'éditer le code généré, typiquement dans le constructeur, afin de donner une valeur unique à cet ID.Le plus simple est d'utiliser l'API d'EMF avec EcoreUtil.generateUUID().

```
/**
 * @generated NOT
 */
protected ElementImpl() {
   super();
   setID(EcoreUtil.generateUUID());
}
```

Pour avoir un traitement homogène de cet EAttribute, il est conseillé de créer une EClass abstraite nommée par exemple **Element** qui définira cet EAttribute et qui servira de super-type aux autres EClasses.**EMF supporte l'héritage multiple mais pas Java.**Toutes vos EClasses avec de l'héritage multiple sont susceptibles de ne pas appeler le super() de Element. Il faudra au choix :

1. modifier l'ordre de l'héritage
2. modifier à la main le constructeur comme dans le code ci-dessus

6.2.4. Propriétés dynamiques

Les propriétés dynamiques permettent d'enrichir les objets du modèle de manière dynamique. Pour supporter les propriétés dynamiques, les types du méta-modèle doivent hériter du type fr.obeo.smartea.core.basemm.PropertiesContainer.

6.2.5. Permissions

Les permissions permettent de filtrer l'accès aux objets du modèle en fonction de droits en lecture, écriture ou administration.Pour supporter les permissions, les types du méta-modèle doivent hériter du type fr.obeo.smartea.core.basemm.Permissions.

6.2.6. Ressources externes

Les ressources externes permettent de créer des liens vers des documents externes à SmartEA via des URL. Pour supporter les ressources externes, les types du méta-modèle doivent hériter du type fr.obeo.smartea.core.basemm.ExternalResourceContainer.

6.2.7. Déploiement

Comme le méta-modèle est destiné à être utilisé par le serveur et le modeleur, il devra être inclus dans la feature commune du module.

6.2.8. Documentation intégrée

Il est également possible de personnaliser la documentation liée au métamodèle, de façon internationalisable, à l'aide d'un point d'extension décrit dans cette section.

6.2.9. Compatibilité avec les extensions de type

Afin que votre métamodèle soit compatible avec les extensions de type il est nécessaire que votre EFactory implémente l'interface *fr.obeo.smartea.core.refmodel.api.SmartEAFactory*.La plupart des méthodes sont implémentées dans *fr.obeo.smartea.core.refmodel.api.SmartEAFactoryImpl*, vous pouvez donc étendre cette implémentation dans votre EFactory.

La méthode qu'il reste à implémenter est *fr.obeo.smartea.core.refmodel.api.SmartEAFactoryImpl.setExtendedProperties(EObject)*.

Voici une proposition d'implémentation (utilisée pour le métamodèle ArchiMate) :

```
public void setExtendedProperties(EObject target) {
   if (!(target instanceof PropertiesContainer) || referenceModelProvider == null) {
      return;
   }

   IReferenceModel referenceModel = this.referenceModelProvider.get();

   if (referenceModel == null || referenceModel.getTypeExtensionLibrary() == null) {
      return;
   }

   PropertiesContainer container = (PropertiesContainer)target;
   referenceModel.getTypeExtensionLibrary().getExtensionsFor(container.eClass()).stream()
         .forEach(extension -> extension.getTypes().forEach(type -> {
            if (type instanceof PropertyType) {
               TypedPropertyServices.INSTANCE.addTypedProperty(container, (PropertyType)type);
            } else if (type instanceof ComputedPropertyType) {
               TypedPropertyServices.INSTANCE.addComputedTypedProperty(container,
                     (ComputedPropertyType)type);
            }
         }));

}
```

6.3. Ajouter un manuel d'aide spécifique

Obeo SmartEA fournit plusieurs manuels utilisateurs :

1. un guide utilisateur
2. un guide technique d'installation et d'administration
3. le présent guide de personnalisation

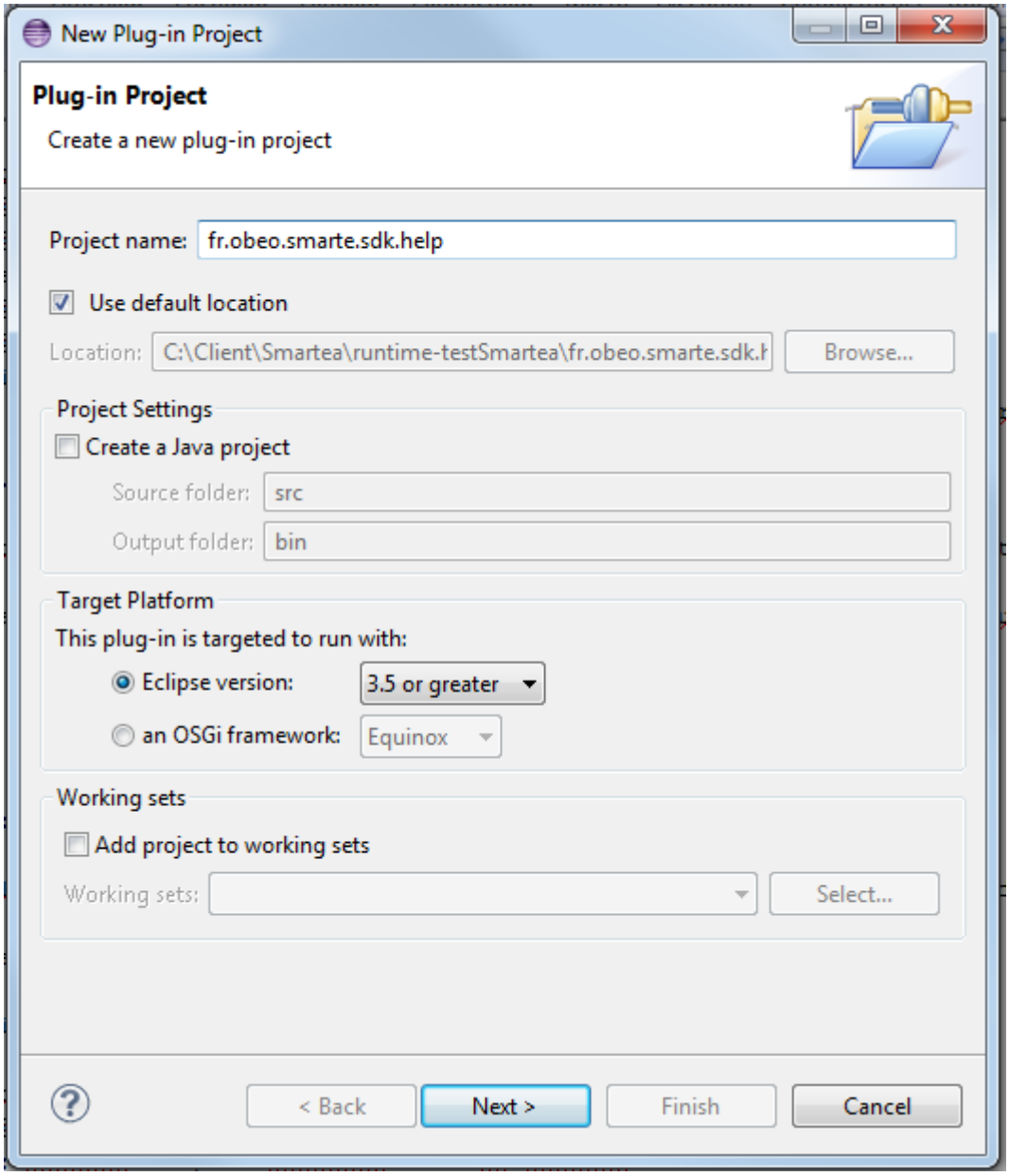
Obeo SmartEA permet de créer ses propres manuels ou documentation. L'IDE SmartEA fournit un outillage pour éditer le document (au format textile) et génère les fichiers qui permettront de rendre cette documentation disponible dans le modeleur et le serveur.

Cette documentation peut être traduite dans plusieurs langues et sera disponible ainsi dans la langue configurer dans le serveur.

6.3.1. Création du plug-in de documentation

L'utilisateur doit créer un projet de type : **Plug-in Project**.

- Sur la première page du wizard de création de plug-in :
  - Saisir le nom du projet.
  - Décocher «create a java project».



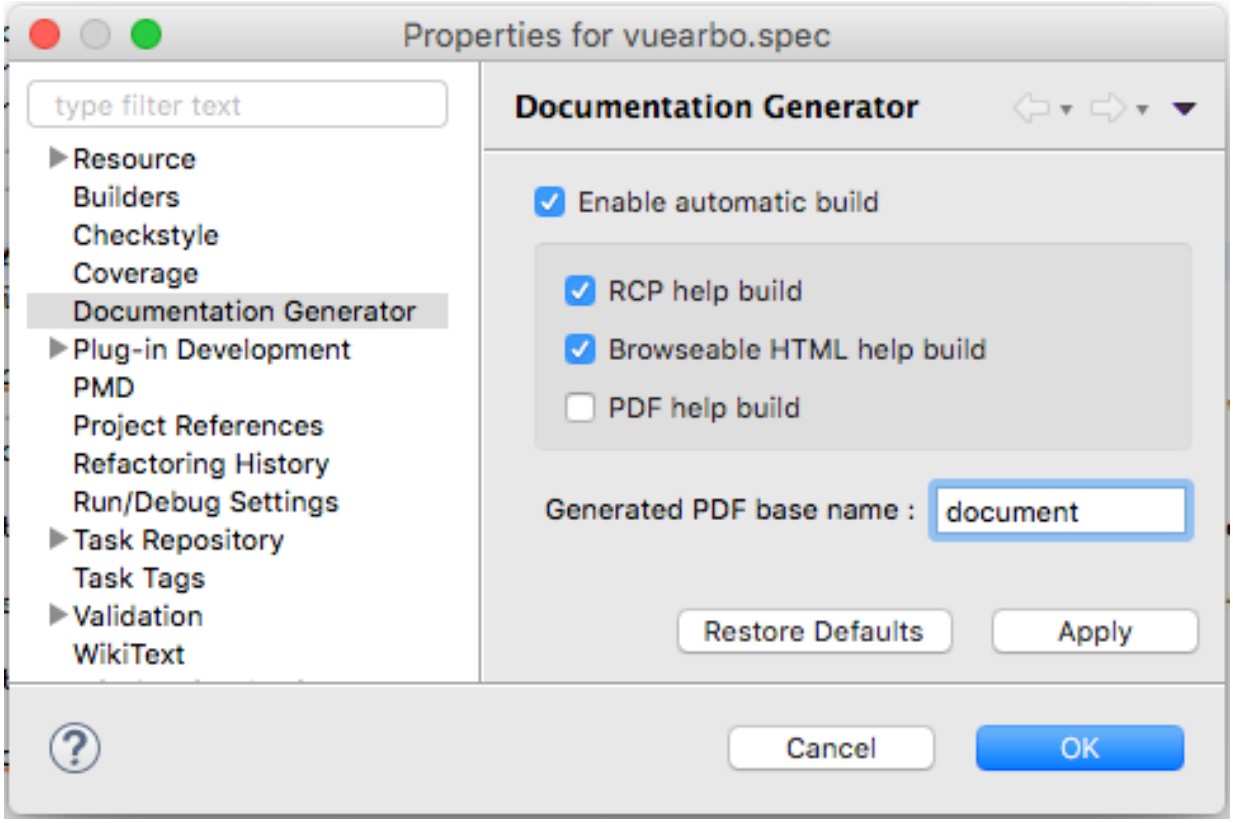
- Sur la deuxième page du wizard :
  - Saisir le nom, dans l'aide d'Eclipse la documentation apparaîtra sous ce nom.
  - Saisir le nom du «vendor», utilisé par le «copyright».
  - Décocher "Generate an activator, a Java class ..."
  - Cliquer sur le bouton «finish».

Éditer le fichier `MANIFEST.MF` du projet :

- Dans l'onglet «overview» cocher «This plug-in is a singleton».

Transformez le plug-in en plug-in de documentation :

- Afficher les propriétés du projet
- Dans la section `Documentation Generator`, activer la construction automatique (`automatic build`) :



A ce moment le projet devient un projet de documentation SmartEA et sera construit en tant que tel.

Par défaut, un fichier content.textile est généré à la racine du projet, si l'utilisateur ne l'a pas déjà créé. Ce fichier est le point de départ de la génération. Toute la documentation doit être rédigée dans ce fichier.

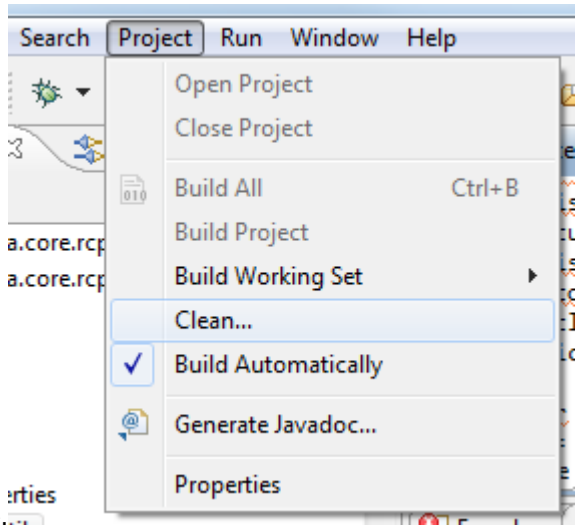
La documentation peut faire appel à un répertoire d'images qu'il est préférable de mettre à la racine du projet. Dans le cas où un dossier contenant des images est utilisé, il faut le rajouter manuellement dans le fichier `build.properties`.

6.3.2. Rédaction du manuel

Le manuel est rédigé selon la syntaxe textile (se reporter au paragraphe dédié pour syntaxe supportée) dans un fichier dont le nom est imposé : `content.textile` (celui-ci peut faire l'objet d'une traduction en plusieurs langues comme précisé dans ce paragraphe [*Internationalization)*].

Dès que le fichier est sauvegardé, la documentation au format HTML est générée :

- `common` : contient les ressources de base (javascript, CSS, ...)
- `rcp` : contient l'aide à destination du modeleur
- `web` : contient l'aide à destination du serveur.



Il est possible de forcer la génération par le menu «Project/clean...».

Dans le cas où des dossiers complémentaires sont utilisés (images, ...), ils doivent être déposés dans le plug-in et sélectionnés manuellement dans le fichier `build.properties`.

6.3.3. Internationalisation

Le fichier **content.textile** représente la langue par défaut de la documentation.

Dans le cas où la documentation doit être traduite dans plusieurs langues, un fichier textile doit être rédigé et stocké à la racine du plug-in pour chaque langue. Ce fichier doit être nommé de la façon suivante : `content_[language].textile`, `content_[language]_[country].textile` ou `content_[language]_[country]_[variant].textile` (ex: content_fr.textile, content_fr_FR_EURO.textile, ...).

Language : doit respecter la norme ISO-639

Country : doit respecter la norme ISO-3166

Dans la plupart des cas, seul le code pays est utilisé. Cela peut donner par exemple :

- `content.textile` : default language (english for example)
- `content_fr.textile` : french
- `content_de.textile` : german

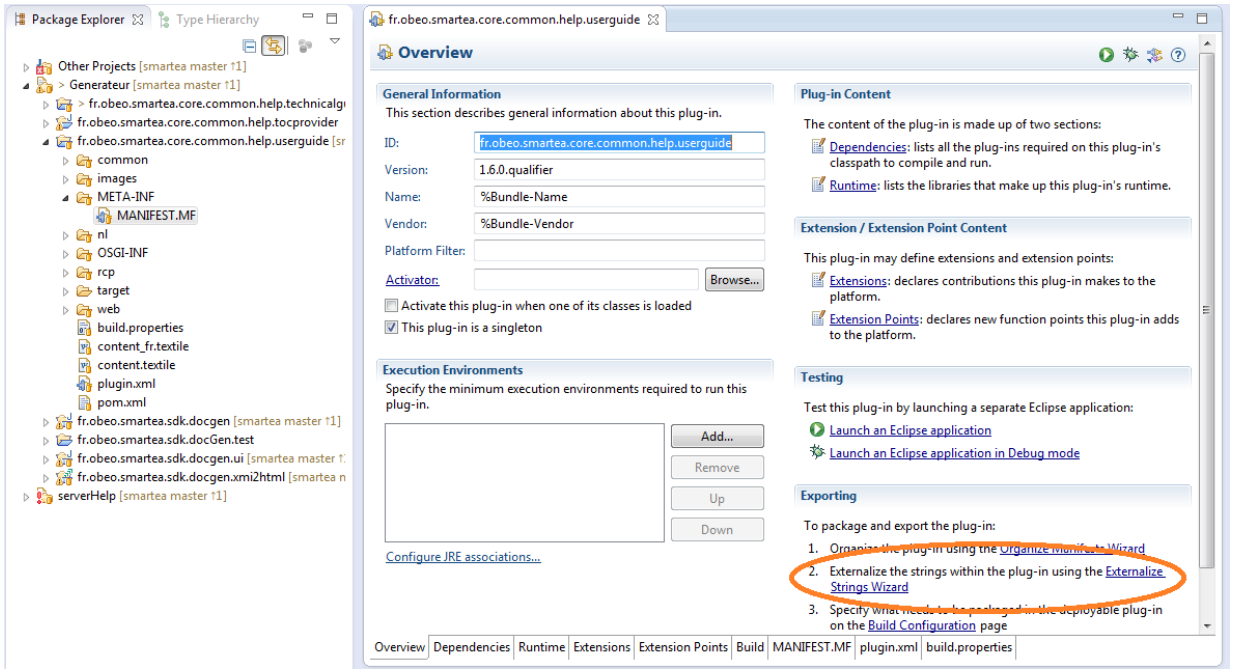
Il est possible, comme pour le fichier par défaut d'utiliser des ressources spécifiques (images, ...). Il est conseillé de stocker ces dernières dans un répertoire spécifique nommé comme précédemment (ex : images/fr).

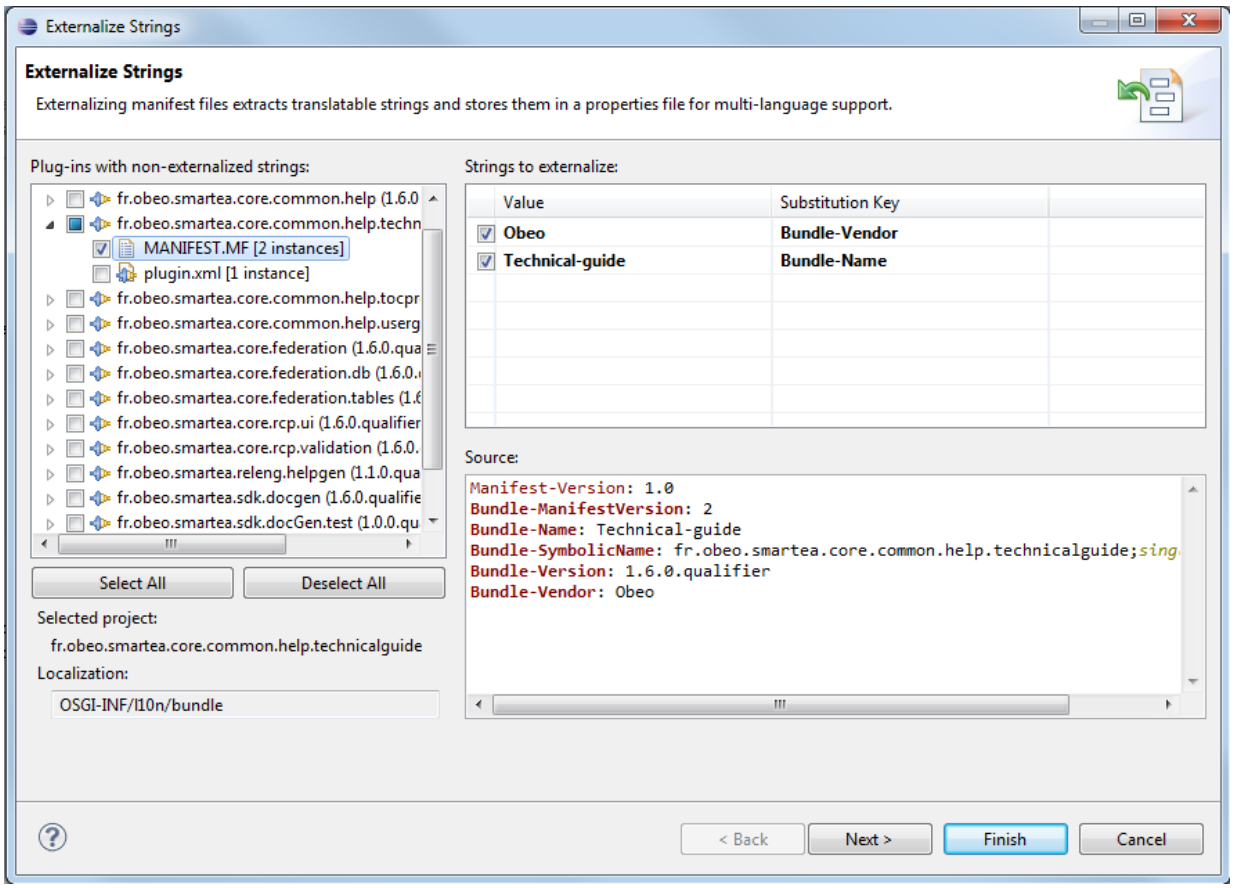
Lors du build, un dossier `nl/language[/country[/variant]]` va être généré contenant le contenu de l'aide Eclipse.

Le dossier «nl» ainsi que les dossiers images additionnels doivent être rajoutés manuellement dans le fichier build.properties.

Sous le dossier web, un dossier est créé et complété automatiquement lors du build pour chaque nouvelle langue.

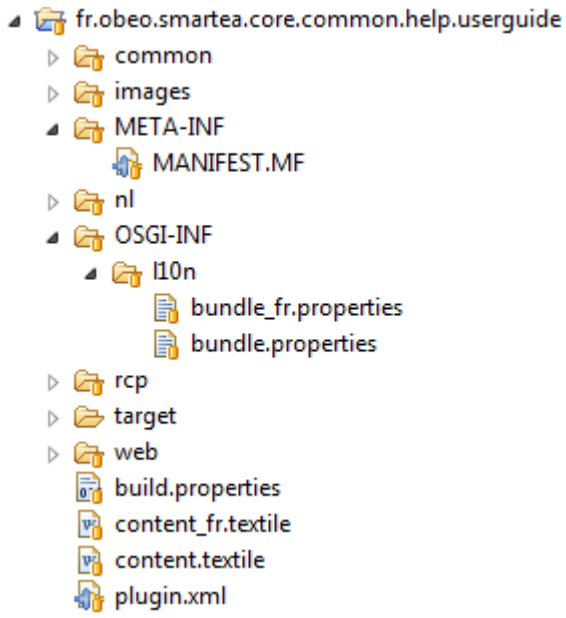
Enfin, il faut internationaliser le manifest.mf en suivant le wizard «externalize string».





Un dossier `OSGI-INF/l10n` est alors créé contenant un fichier `bundle.properties`.

Il faut ensuite manuellement créer un fichier correspondant à la langue du fichier textile nommé comme précédemment `bundle_[language].properties`, `bundle_[language]_[country].properties` ou `bundle_[language]_[country]_[variant].properties` (ex: bundle_fr_FR_EURO.properties). Dans le cas où ce fichier n'existe pas, la langue par défaut sera utilisée.



Exemple d'un fichier `bundle_fr.properties` :

```
#Properties file for fr.obeo.smartea.core.common.help.userguide
Bundle-Vendor = Obeo
Bundle-Name = Guide-Utilisateur
```

6.3.4. Déploiement

Comme la documentation sera accessible depuis le serveur et le modeleur, le plug-in réalisé devra être inclus dans la feature commune du module.

6.4. Ajouter un générateur

Il est possible de déployer dans Obeo SmartEA des générateurs Acceleo, Java ou M2Doc. Ceux-ci apparaissent alors sous forme d'artefacts.

6.4.1. Générateurs Acceleo

Pour illustrer nos propos nous prendrons l'exemple d'un générateur HTML permettant d'extraire les diagrammes d'un répertoire du référentiel.

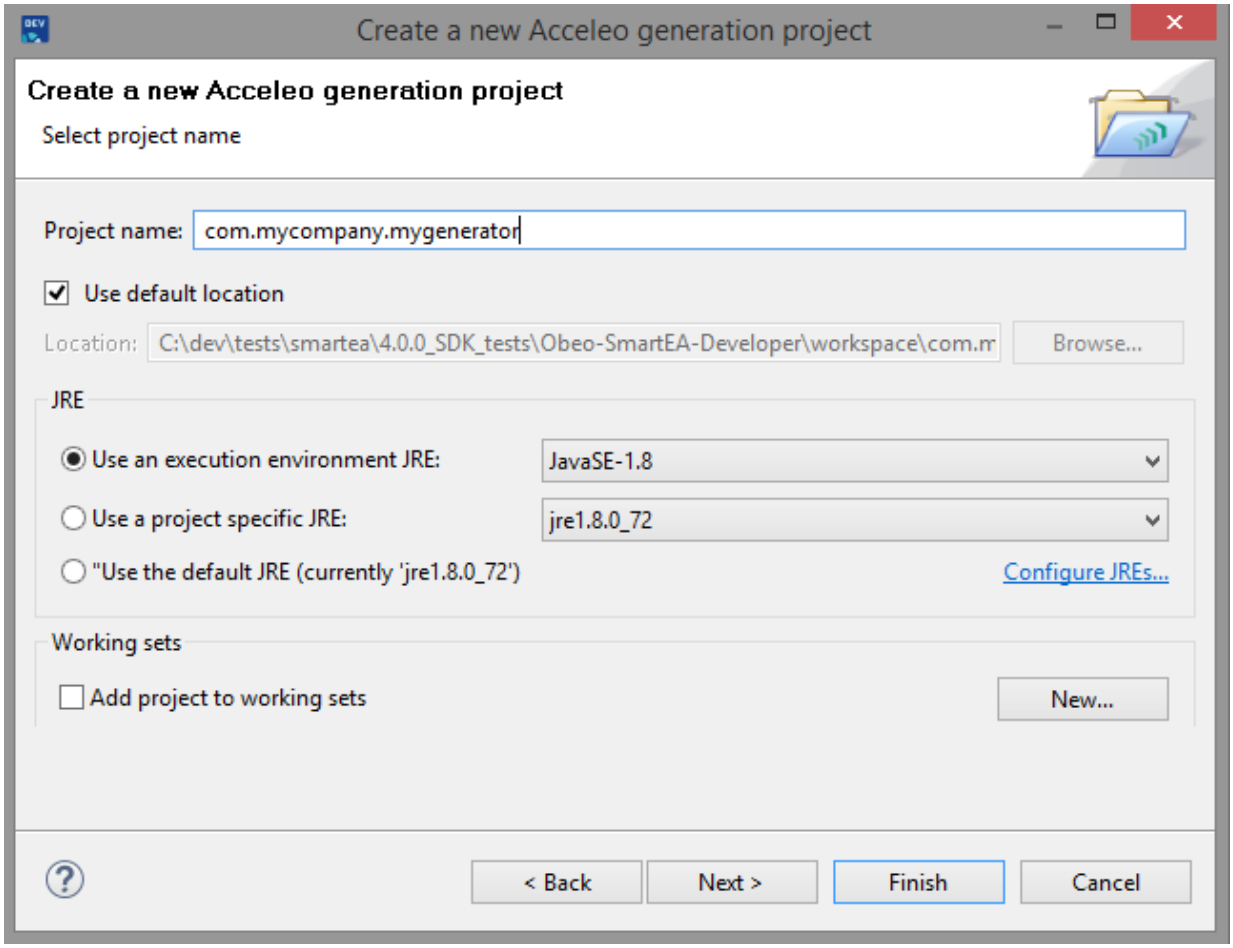
6.4.1.1. Développement du générateur

Il est tout d'abord nécessaire de créer un plug-in Acceleo qui accueillera le générateur.

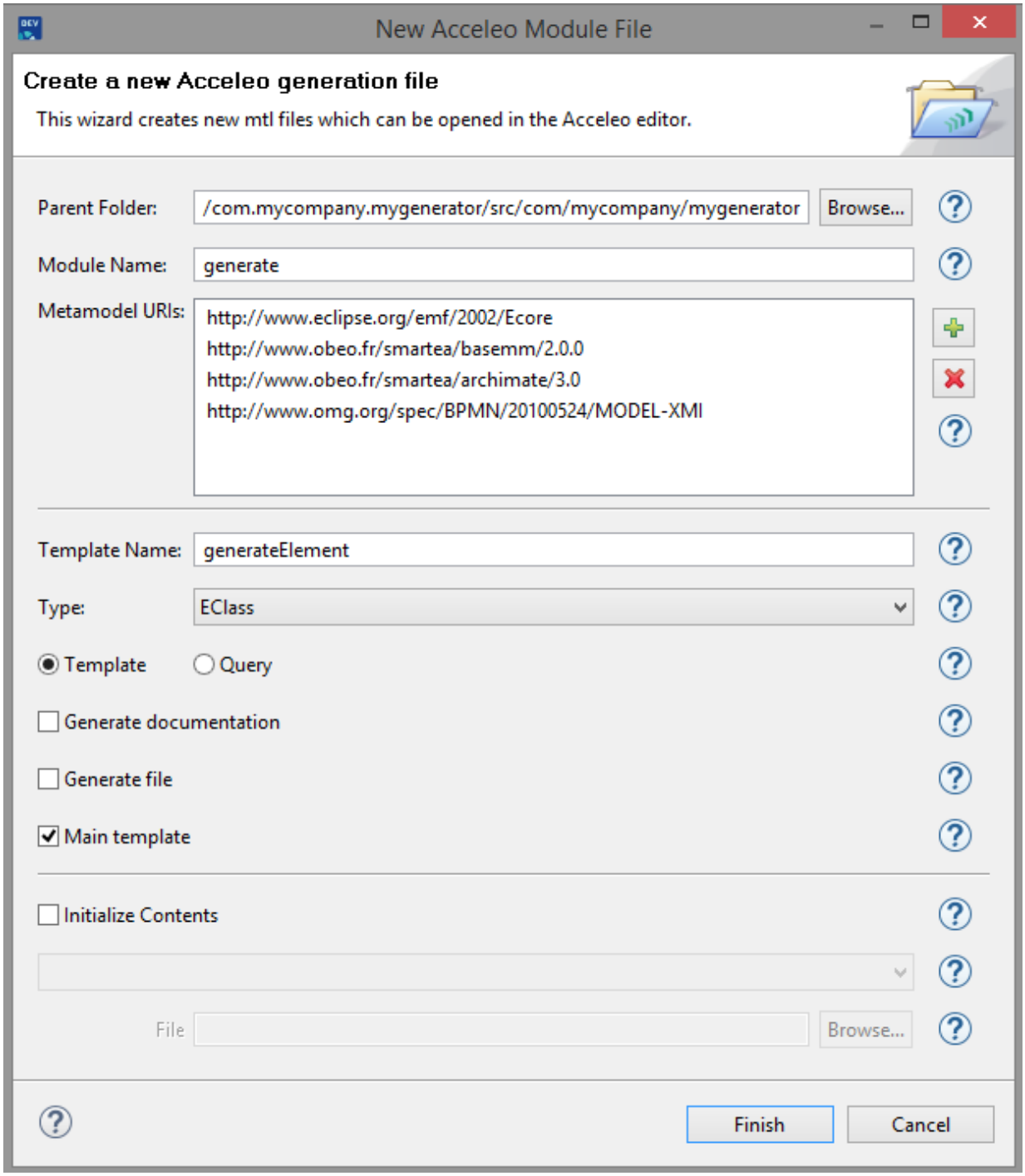
Vous trouverez ci-dessous un guide basique vous permettant d'initier un projet Acceleo. Pour de plus amples informations sur Acceleo, vous êtes invités à consulter la documentation associée.

Pour créer votre projet :

- Menu "`File > New`" puis "`Acceleo project`"
- Choisissez un nom pour votre plug-in (ex : `com.mycompany.mygenerator`)
- Cliquez sur "`Next`"



- Dans la zone "`Metamodels URIs`", choisissez les meta-modèles que vous souhaitez manipuler ; dans notre exemple nous prendrons le méta-modèle Ecore (`http://www.eclipse.org/emf/2002/Ecore`) et le métamodèle de base de SmartEA `http://www.obeo.fr/smartea/basemm/2.0.0`.
- Si vous souhaitez utiliser l'API SmartEA de génération, ajoutez également le méta-modèle suivant : `http://www.obeo.fr/smartea/core/genctx/1.0.0`
- Ajoutez tous les métamodèles que vous souhaitez utiliser dans le générateur (Archimate, BPMN...)
- Cochez l'option `Main template`
- Vous pouvez changer le nom du template, puis cliquez sur `Finish` ; dans notre exemple, nous laissons les choix par défaut



- Si vous souhaitez utiliser l'API SmartEA de génération dans votre template, ajoutez lui un paramètre d'entrée de type `GenContext`

Dans notre exemple, nous souhaitons appliquer notre template à un répertoire du référentiel et nous ajoutons le paramètre permettant d'exploiter les API SmartEA de génération :

```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/emf/2002/Ecore', 'http://www.obeo.fr/smartea/core/
genctx/1.0.0', 'http://www.obeo.fr/smartea/basemm/2.0.0', 'http://www.obeo.fr/smartea/core/
artifact/1.0.0')]

[template public generateElement(ctx : Folder, genCtx : GenContext)]
[comment @main/]
[file (ctx.name+'.html', false, 'UTF-8')]
```

```
<html>
<meta charset="UTF-8">
<title>[ctx.name/] details</title>
<style type="text/css">
body, h1, h2 {
   font-family: Lucida Sans Unicode;
   background-color: #F0F0FF;
}
</style>
<body>
<h1>[ctx.name/] details</h1>

<h2>Artifacts</h2>
<ul>
[for (element : EObject | ctx.elements)]
   [if (element.oclIsKindOf(artifact::Artifact))]
      [let artifact : artifact::Artifact = element]
      [let embeddable : Embeddable = artifact.toEmbeddable()]
<h3>[artifact.name/]</h3>
      [if (embeddable.oclIsUndefined())]
Diagram is not published.
      [elseif (embeddable.isImage())]
<img src="[genCtx.toBase64(embeddable.oclAsType(EImage))/]">
      [/if]
      [/let]
      [/let]
   [/if]
[/for]
</ul>

<h2>Sub folders</h2>
<ul>
[for (folder : Folder | ctx.folders)]
<li>[folder.name/]</li>
[/for]
</ul>

</body>
</html>
[/file]
[/template]
```

6.4.1.2. Activation du générateur

Pour mettre le générateur à disposition dans SmartEA il faut implémenter le point d'extension correspondant.

Dans notre cas, après avoir ajouté la dépendance `fr.obeo.smartea.core.common.api` à notre plug-in, la déclaration de l'extension dans le `plugin.xml` ressemblera à l'exemple suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension
         point="fr.obeo.smartea.core.common.api.generator">
      <acceleo
            class="com.mycompany.mygenerator.main.Generate"></acceleo>
   </extension>
</plugin>
```

6.4.1.3. Test du générateur (avant déploiement)

Il est possible de tester et stabiliser son générateur avant de le déployer dans un serveur, en utilisant un connecteur en lecture seule. Pour ce faire, vous devez avoir un serveur démarré.

Créez un plug-in que vous utiliserez pour vos tests (même procédure que pour le plug-in de développement des extensions), en y incluant les dépendances suivantes :

```
org.apache.commons.codec,
org.apache.commons.logging,
org.eclipse.net4j.tcp,
org.eclipse.emf.cdo.net4j,
fr.obeo.smartea.core.basemm,
fr.obeo.smartea.core.server.client,
fr.obeo.smartea.core.prism,
fr.obeo.smartea.core.prefs,
fr.obeo.smartea.core.server.ext,
fr.obeo.smartea.core.server.cdo,
com.mycompany.mygenerator
```

Vous trouverez ci-dessous un exemple de code permettant de lancer la génération d'un template Acceleo packagé au format Obeo SmartEA :

```
package fr.obeo.smartea.sample.connector.test.generator;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;
import org.eclipse.emf.cdo.CDOObject;
import org.eclipse.emf.cdo.eresource.CDOResource;

import fr.obeo.smartea.core.common.ZipUtil;
import fr.obeo.smartea.core.common.api.client.SmartEAProtocol;
import fr.obeo.smartea.core.common.api.gen.GenerationResult;
import fr.obeo.smartea.core.common.api.gen.ISmartEAGeneratorsManager;
import fr.obeo.smartea.core.server.api.db.DbProjectContentCtx;
import fr.obeo.smartea.core.server.api.db.DbProjectContentTx;
import fr.obeo.smartea.core.server.client.SmartEAClient;
import fr.obeo.smartea.core.server.client.SmartEAProjectClient;
import fr.obeo.smartea.sample.generator.acceleo.main.Generate;

public class TestAcceleo {

   public static void main(String[] args) throws IOException {
      // Initialize SmartEA Client
      SmartEAClient smartEAClient = new SmartEAClient(SmartEAProtocol.HTTP, // Protocol
            "localhost", // HTTP host
            8080, // HTTP port
            "afontaine", // User ID
            "123", // Password
            true // Read only mode for internal db context
      );

      //
      SmartEAProjectClient sandboxClient = smartEAClient.newProjectClient("voyagediscount",
            // Read only mode
            true);

      // Retrieve Generation context
      DbProjectContentCtx ctx = sandboxClient.getProjectContentCtx();
      DbProjectContentTx tx = ctx.getTx(ctx.getMasterBranch());
      CDOResource semanticResource = tx.getSemanticResource();

      // To test, we take the first folder of the repository
      CDOObject context = (CDOObject) semanticResource.getContents().get(0);
```

```
      // Generate
      GenerationResult res = sandboxClient.getInstance(ISmartEAGeneratorsManager.class)
            .getAcceleoGenerator(Generate.class).generate(context);
      if ("application/zip".equals(res.getContentType())) {
         // several files are generated
         ZipUtil.unzip(new ByteArrayInputStream(res.getContent()), new File("output"));
      } else {
         // a single file is generated
         FileUtils.writeByteArrayToFile(new File("output/generated"), res.getContent());
      }

      // Release context
      sandboxClient.release();
      smartEAClient.release();
   }
}
```

6.4.1.4. Paramétrage avancé

Il est possible d'annoter son générateur afin de modifier le comportement lors de la génération de l'artefact.

Exemple :

```
/**
 * Entry point of the 'DescProcessusODT' generation module.
 *
 * @generated
 */
@GenerationBase("odt.zip")
@FileExtension("odt")
@ContentType("application/vnd.oasis.opendocument.text")
@GeneratorIcon("libreoffice-writer.png")
public class DescProcessusODT extends AbstractAcceleoGenerator {
```

Les annotations disponibles sont les suivantes :

- `GenerationBase` :
  - Cette annotation permet de définir un ensemble de fichiers servant de base pour la génération (ex : un ensemble de ressources pour un site web ; images, css, js, ...)
  - Ces fichiers doivent être rassemblés dans une archive (format zip) et l'archive déposée dans le même package Java que le générateur Acceleo.
  - Cette annotation prend un paramètre qui correspond au nom de l'archive zip contenant les fichiers.
- `FileExtension` : Cette annotation permet de définir l'extension de fichier qui va être utilisé pour construire le nom du fichier retourné au navigateur web (par concaténation du nom de l'artefact et de l'extension dans le cas ou plusieurs fichiers sont générés par le générateur Acceleo, ou du nom du seul fichier généré si un seul fichier est généré)
- `ContentType` : Cette annotation permet de définir le type de contenu retourné au navigateur web (cf. liste d'exemples sur Wikipedia)
- `GeneratorIcon` :
  - Cette annotation permet de changer l'icône utilisée pour les instances et types d'artefact associés au générateur.
  - Cette annotation prend un paramètre qui correspond au nom du fichier de l'icône, qui doit être déposé dans le même package Java que le générateur Acceleo.

6.4.1.5. Déploiement

Pour construire la version déployable de votre générateur Acceleo, il est nécessaire que celle-ci embarque la version compilée du template (soit le fichier .emtl) dans le jar.Pour automatiser cette étape, deux solutions sont possibles selon la façon dont vous construisez l'update-site P2. Les deux solutions ne sont pas incompatibles.

6.4.1.5.1. Construction de l'update-site avec Eclipse PDE

Si vous construisez votre module en utilisant l'action «export» fournie par Eclipse sur votre plug-in :

- Ajoutez à la racine du plug-in un fichier nommé `pde-deploy-emtl.xml` contenant le code (ant) suivant :

```
<!-- ============================================================================ -->
<!-- This script is intended to deploy built emtl from within a PDE build  -->
<!-- just before the jar is created.                                        -->
<!-- Don't forget to set customBuildCallbacks in your build.properties :   -->
<!-- customBuildCallbacks = pde-deploy-emtl.xml                             -->
<!-- ============================================================================ -->
<project name="Deploy EMTL" default="noDefault">
    <target name="noDefault">
        <echo message="This file must be called with explicit targets" />
    </target>

    <!-- ============================================================================ -->
    <!-- This will be called automatically after the compilation of each   -->
    <!-- Bundle... in dependency order.                                     -->
    <!-- ============================================================================ -->
    <target name="post.compile.@dot">
        <copy todir="${target.folder}">
            <fileset dir="${source.folder1}/../bin">
               <include name="**/*.emtl"/>
            </fileset>
        </copy>
    </target>

</project>
```

- Puis ajoutez la ligne suivante au fichier `build.properties` du plug-in :

```
customBuildCallbacks = pde-deploy-emtl.xml
```

6.4.1.5.2. Construction de l'update-site avec Tycho

Si vous construisez le module avec un build Tycho, ajoutez simplement le code suivant à la fin du `pom.xml` de votre plug-in :

```
[...]
  <packaging>eclipse-plugin</packaging>

  <build>
      <plugins>
         <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>exec-maven-plugin</artifactId>
            <version>1.2.1</version>
            <extensions>true</extensions>
            <executions>
               <execution>
                  <phase>process-classes</phase>
                  <goals>
                     <goal>java</goal>
                  </goals>
                  <configuration>
                     <mainClass>fr.obeo.smartea.core.common.gen.util.SmartEAAcceleoCompiler</
mainClass>
                     <arguments>
                        <argument>${basedir}/src</argument>
                        <argument>${project.build.directory}/classes</argument>
                     </arguments>
                     <classpathScope>compile</classpathScope>
                  </configuration>
```

```
               </execution>
            </executions>
         </plugin>
      </plugins>
   </build>

</project>
```

6.4.2. Générateurs Java

L'approche est exactement la même que pour les générateurs Acceleo (voir le paragraphe précédent), à l'exception du fait qu'il vous faudra implémenter un générateur en Java plutôt que sous la forme d'un template Acceleo (basé sur l'interface `fr.obeo.smartea.core.server.api.artifacts.generator.IGeneratorExtension`).

Le point d'extension à utiliser est le même que pour les générateurs Acceleo, mais au lieu de l'entrée **acceleo**, c'est une entrée **generic** qu'il vous faudra ajouter.

Vous pouvez tester le générateur avant déploiement en utilisant un connecteur de la même façon que celle décrite pour Acceleo mais en utilisant cette fois la méthode `fr.obeo.smartea.core.common.api.gen.ISmartEAGeneratorsManager.getJavaBasedGeneration(Class<? extends IGeneratorExtension<C>>)` pour initialiser le template.

6.4.3. Générateurs M2Doc

Pour ajouter un template M2Doc à SmartEA il vous faudra implémenter une classe Java référençant le template, qui implémente `fr.obeo.smartea.core.common.api.gen.IM2docGeneratorExtension`, comme dans l'exemple suivant :

```
public class M2docEmbeddedGenerator implements IM2docGeneratorExtension<Folder> {

   @Override
   public String getTemplatePath() {
      return M2docEmbeddedGenerator.class.getPackage().getName().replace('.', '/') + "/
template.docx";
   }

   @Override
   public void configure(Folder context, IQueryEnvironment queryEnvironment, Map<String,
 Object> variables)
         throws IOException {
      // Nothing special to do for this template
   }

}
```

Dans cet exemple, le template est placé dans le package Java de la classe `M2docEmbeddedGenerator`.

Puis comme pour les générateurs Acceleo et Java il vous faudra implémenter le point d'extension en y ajoutant une entrée **m2doc**.Vous pouvez tester le générateur avant déploiement en utilisant un connecteur de la même façon que celle décrite pour Acceleo mais en utilisant cette fois la méthode `fr.obeo.smartea.core.common.api.gen.ISmartEAGeneratorsManager.getConfigurerBasedM2docGenerator(Class<? extends IM2docGeneratorExtension<C>>)` pour initialiser le template.

6.4.4. Définition du libellé d'un générateur

Dans le plug-in qui contient votre générateur, créez un répertoire à la racine nommé `i18n`. Ouvrez le fichier `build.properties` et sélectionnez le répertoire que vous venez de créer.

Le fichier `build.properties` devrait ressembler à ceci (en particulier le dernier item du paramètre `bin.includes`) :

```
source.. = src/
output.. = bin/
bin.includes = META-INF/,\
               .,\
               plugin.xml,\
               i18n/
```

Dans le répertoire `i18n`, nous allons créer autant de fichiers que nous souhaitons supporter de langages :

- `i18n/messages` : libellés par défaut (en général on y consigne les libellés en anglais)
- `i18n/messages.fr` : libellés français
- `i18n/messages.it` : libellés italiens
- ...

Si vous souhaitez ne supporter qu'un langage vous pouvez vous contenter de fournir simplement le fichier `i18n/messages`.

Ces fichiers doivent contenir une liste de libellés ou messages indexés par des clés.

La clé qui doit être utilisée doit être construite en concaténant `artifact.generator.` au nom qualifié de la classe Java du générateur. Par exemple :

```
artifact.generator.com.mycompany.mygenerator.main.Generate=Mon Générateur
```

- Pour Acceleo on référencera la classe qui hérite de `org.eclipse.acceleo.engine.service.AbstractAcceleoGenerator`
- Pour un générateur Java on référencera la classe qui implémente `fr.obeo.smartea.core.common.api.gen.IGeneratorExtension`
- Pour un générateur M2Doc on référencera la classe qui implémente `fr.obeo.smartea.core.common.api.gen.IM2docGeneratorExtension`

6.4.5. Déploiement d'un générateur dans SmartEA

Le générateur devra être inclus dans la feature commune du module.

Le serveur doit être redémarré pour prendre en compte les modifications. Une fois déployé, le générateur est rendu disponible sous forme de type d'artefact. Ce type d'artefact peut ainsi être instancié sur tous les objets héritant du type du premier paramètre du template Acceleo.

6.5. Ajouter une vue Sirius

Dans cette section nous décrivons la procédure nécessaire à la création d'une nouvelle vue Sirius et à son déploiement dans SmartEA.

6.5.1. Création de la vue

Vous pouvez créer une vue Sirius depuis L'IDE, en utilisant le menu "`File > New`" puis "`Viewpoint Specification Project`".

Pour plus de détails sur la conception d'une vue Sirius, veuillez vous reporter à la documentation Sirius.

Dans la suite de cette section nous utiliserons un exemple simple : une table listant le contenu direct d'un répertoire SmartEA. Voici le contenu du fichier .odesign :

```
<?xml version="1.0" encoding="UTF-8"?>
<description:Group xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns:description="http://www.eclipse.org/sirius/
description/1.1.0" xmlns:description_1="http://www.eclipse.org/sirius/table/description/1.1.0"
 name="project" version="12.0.0.2017041100">
```

```
  <ownedViewpoints name="MyViewpoint">
    <ownedRepresentations xsi:type="description_1:EditionTableDescription" name="contentTable"
 domainClass="basemm.Folder">
      <metamodel href="http://www.obeo.fr/smartea/basemm/2.0.0#/"/>
      <ownedLineMappings name="folders" domainClass="basemm.Folder"
 semanticCandidatesExpression="feature:folders"/>
      <ownedLineMappings name="element" domainClass="EObject"
 semanticCandidatesExpression="feature:elements"/>
    </ownedRepresentations>
  </ownedViewpoints>
</description:Group>
```

Cette vue exploite le métamodèle de base de SmartEA, vous devrez donc ajouter la dépendance `fr.obeo.smartea.core.basemm` aux dépendances du plugin contenant le .odesign.

Attention : Une description d'un diagramme Sirius ne doit pas avoir la valeur `true` pour les 2 attributs suivants (noeud Diagram Description) :

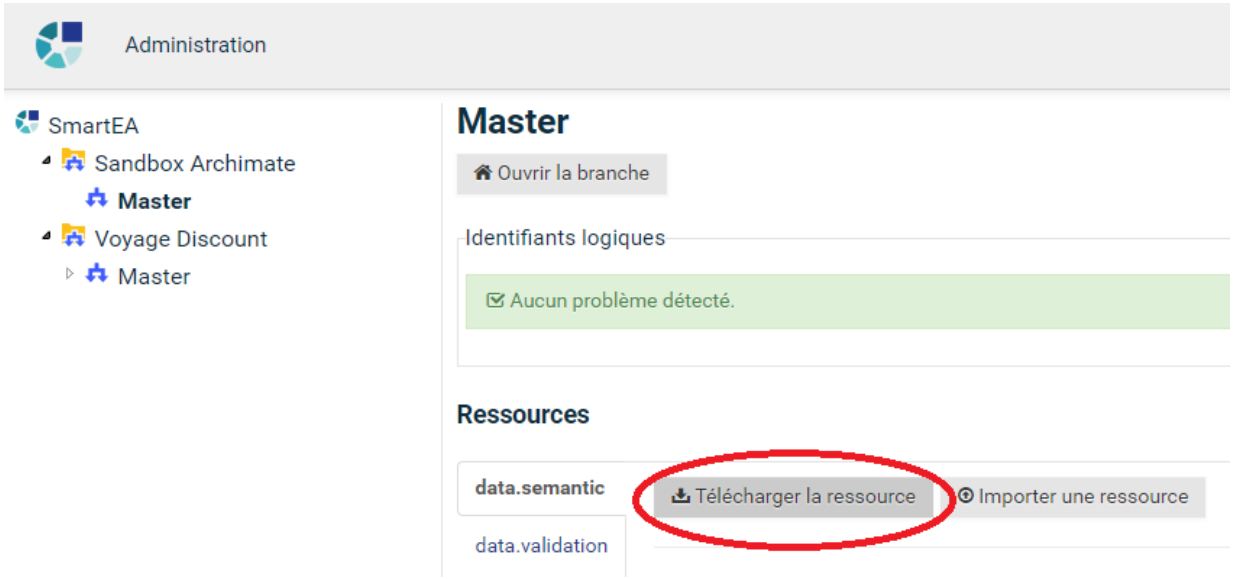
- Initialization
- Show on Startup

Ces fonctionnalités apportées par Sirius ne sont pas supportées par SmartEA.

6.5.2. Test de la vue (avant déploiement)

Il est possible de tester la nouvelle vue sur des données réelles sans avoir à la déployer dans SmartEA.

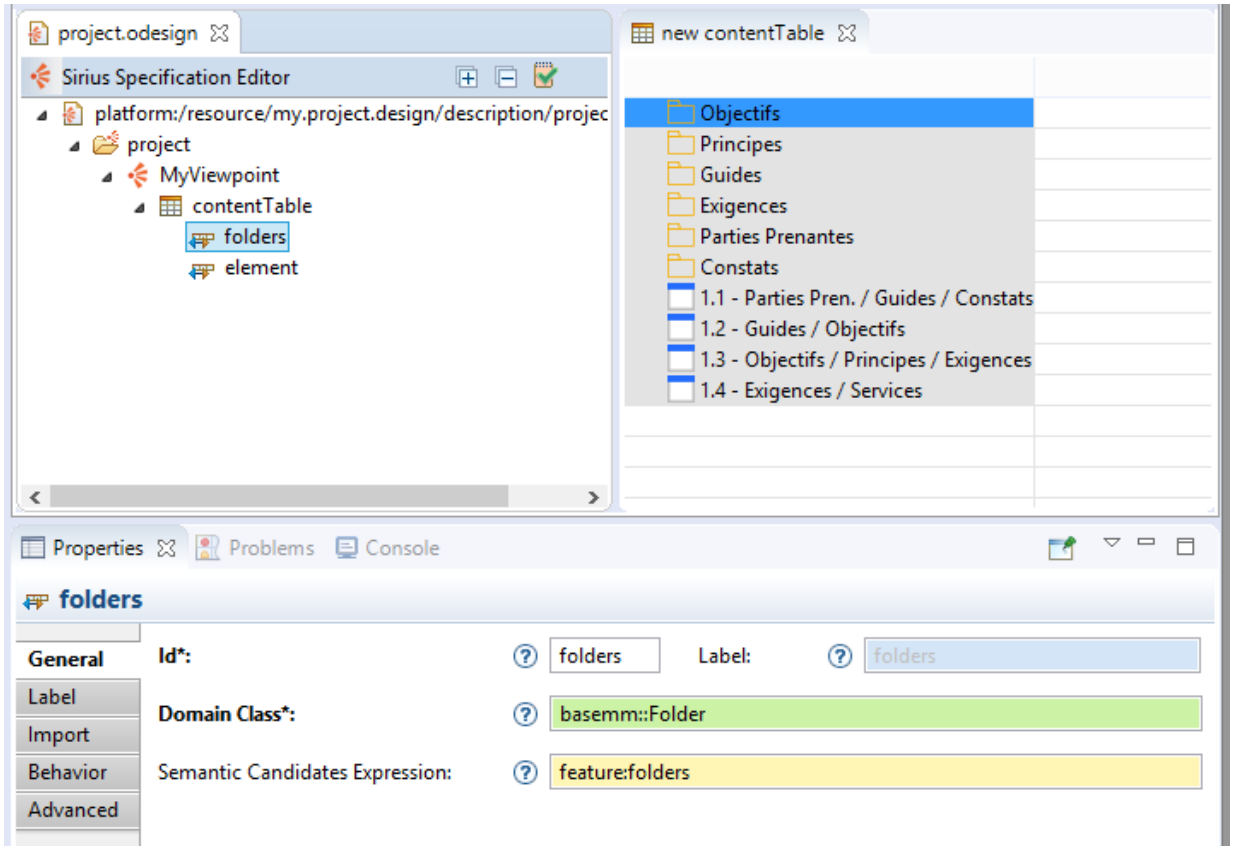
Pour ce faire, la première étape est de récupérer un modèle de test. La solution la plus simple est de faire un export du modèle sémantique de SmartEA, en utilisant l'interface d'administration du serveur comme montré ci-dessous :



Vous obtiendrez ainsi un fichier `data.semantic`.

Créez ensuite un nouveau projet de test à l'aide du menu "`File > New`" puis "`Modeling Project`".Dans ce projet, importez le fichier `data.semantic` précédemment obtenu.

Sélectionnez le nouveau point de vue sur le projet de test puis, par clic-droit sur un répertoire dans le modèle sémantique, créez une nouvelle représentation.

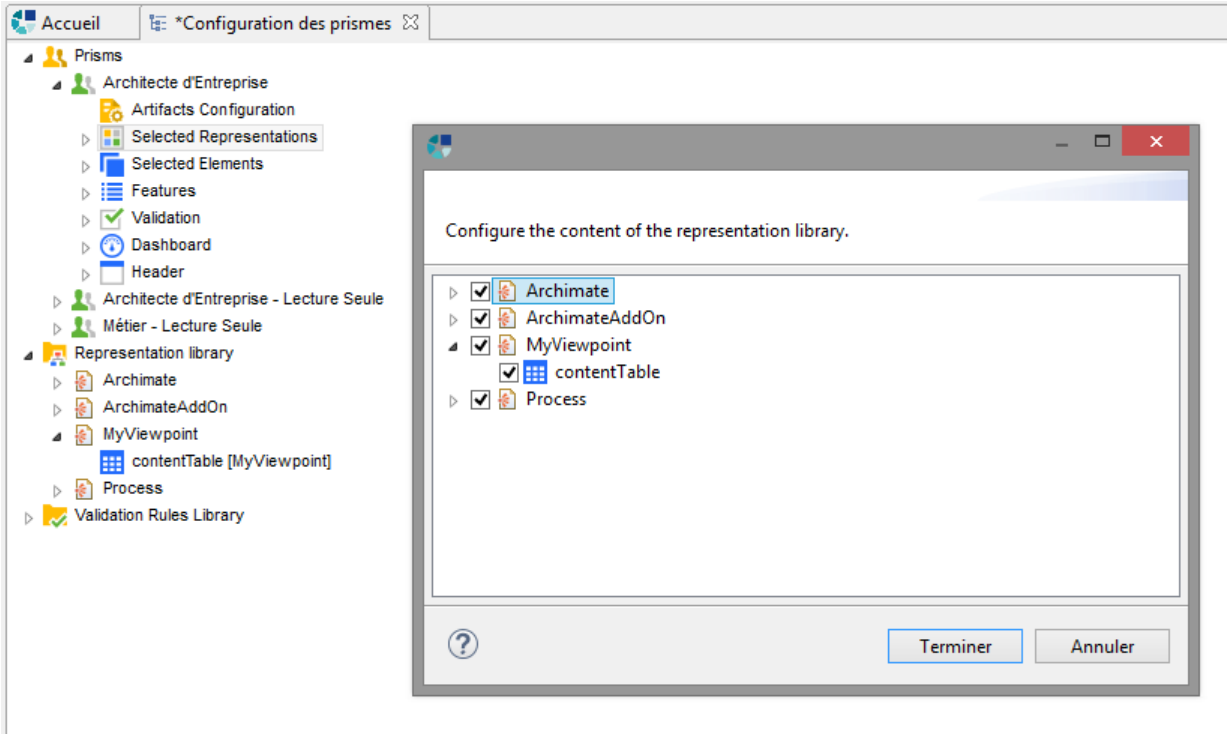


Attention : si vous souhaitez utiliser des services Java dans votre vue Sirius, vous devrez lancer un runtime eclipse depuis l'IDE puis créer un projet de test (comme décrit précédemment) dans ce nouveau workspace.

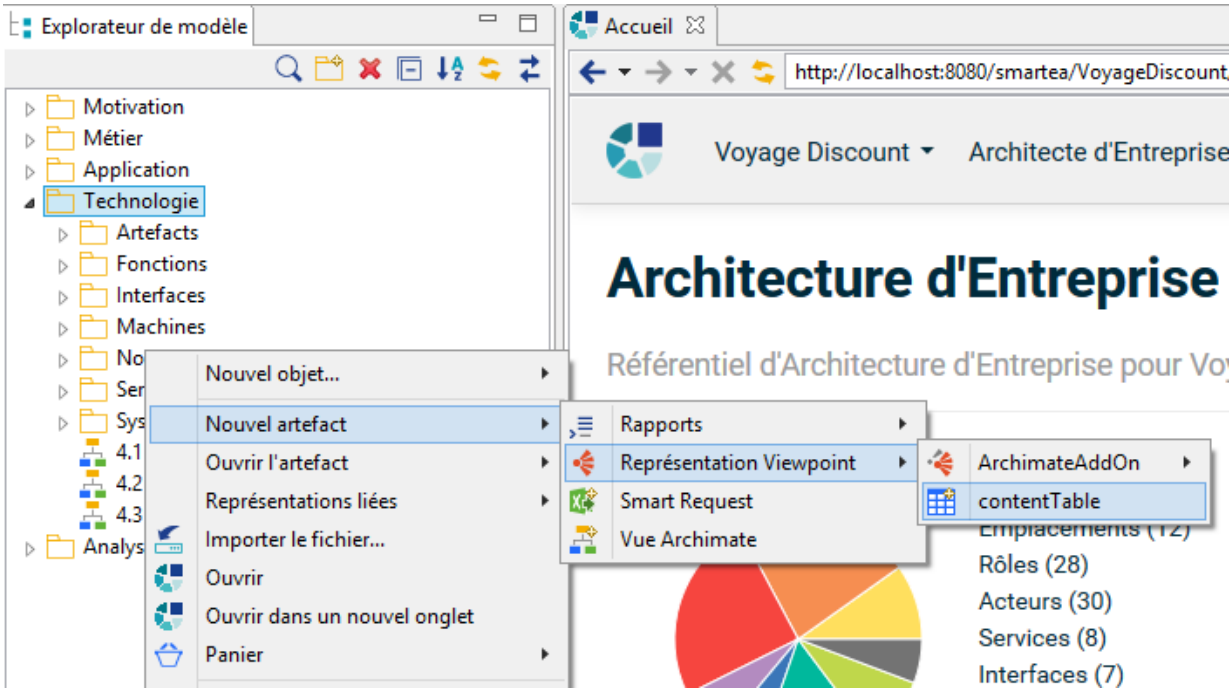
6.5.3. Déploiement

Le plugin de spécification Sirius (celui contenant le .odesign) devra être inclus dans la feature modeleur du module.

Le serveur doit être redémarré pour prendre en compte les modifications. Une fois déployé, la nouvelle vue pourra être déclarée dans la «Representation Library» du prisme, puis être sélectionnée dans les prismes.



Enfin, vous pourrez la créer en tant que nouvel artefact depuis un objet correspondant au contexte de la vue (soit un répertoire dans notre exemple).



6.6. Activation d'une fonctionnalité spécifique dans le prisme

Il est possible de conditionner l'activation de votre code spécifique par une fonctionnalité du prisme. Pour cela vous devrez convenir d'un identifiant décrivant votre fonctionnalité, déclarer cet identifiant dans les prismes ou vous souhaitez rendre la fonctionnalité disponible via un clic-droit sur l'élément **Features** > «Create a custom feature».Enfin au sein de votre code vous devrez gérer l'activation ou non de votre fonctionnalité en vérifiant la présence de cette fonctionnalité dans le prisme courant, comme dans l'exemple ci-dessous :

```
String myFeatureId = "myFeatureId";
boolean isMyFeatureActivated = false;

final String currentPrismID = RCPContext.INSTANCE.getCurrentPrismID();
if (RCPContext.INSTANCE.getProjectInternalSession() != null &&
 RCPContext.INSTANCE.getProjectInternalSession().isOpen() && currentPrismID != null) {

     final Prism activePrism =
 RCPContext.INSTANCE.getProjectInternalSession().findById(currentPrismID);
   if (activePrism != null) {
      for (final GenericFeature feature : Iterables.filter(activePrism.getFeatures(),
 GenericFeature.class)) {
         if (myFeatureId.equals(feature.getId())) {
            isMyFeatureActivated = true;
         }
      }
   }
}

if (isMyFeatureActivated) {
   // specific feature code is allowed from there
   // ...
}
```

6.7. Autres personnalisations courantes

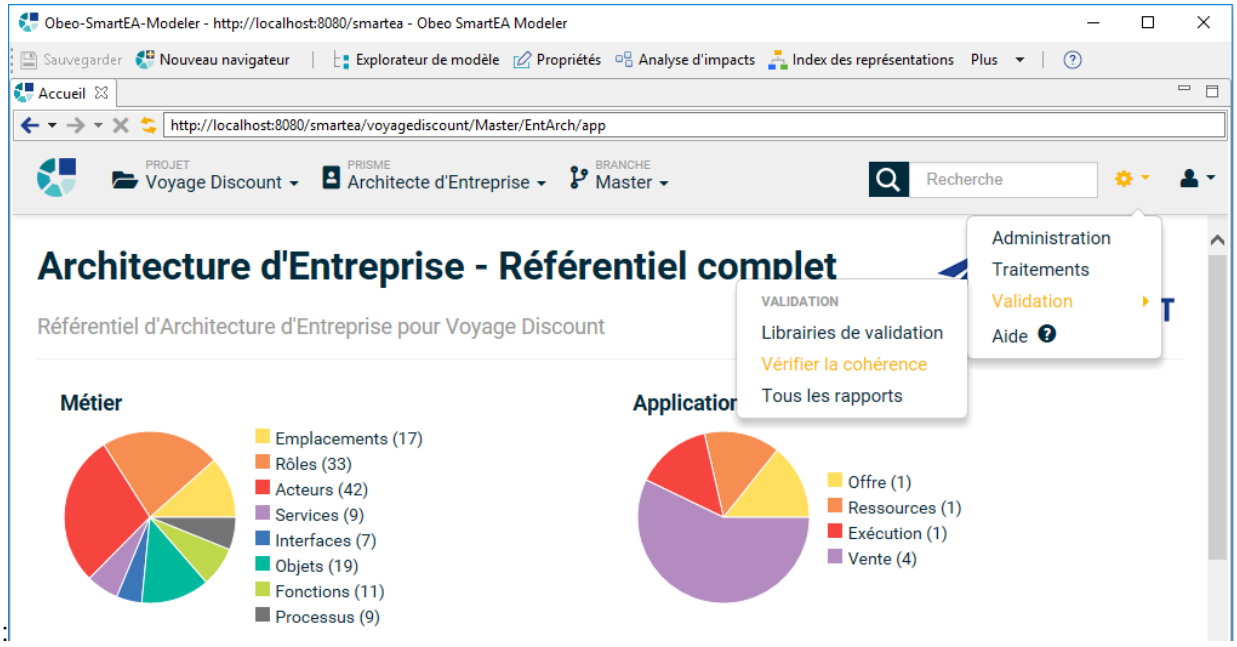
Les APIs SmartEA permettent bon nombre de personnalisations. Vous trouverez des descriptions détaillées permettant de les implémenter dans le glossaire, les plus courantes lors de la création d'un module étant :

- Contribuer au menu de création d'éléments
- Contribuer des références dérivées
- Contribuer un service AQL
- Etendre une représentation Sirius existante : Il est possible d'utiliser le mécanisme d'extension de représentation proposé par Sirius (voir documentation Sirius). Une fois le plugin contenant le odesign étendant une représentation terminé, vous devez l'ajouter dans votre update site déployé coté serveur afin qu'il s'installe au prochain démarrage des modeleurs.

7. Valider le référentiel avec Java

La validation du référentiel SmartEA peut se faire à l'aide de règles AQL ou de librairies Java.

Dans tous les cas, pour lancer une validation sur un prisme, vous devez être dans le modeleur et utiliser le menu suivant :



Dans certains cas, il peut être plus facile d'implémenter une règle de validation Java qu'en AQL.

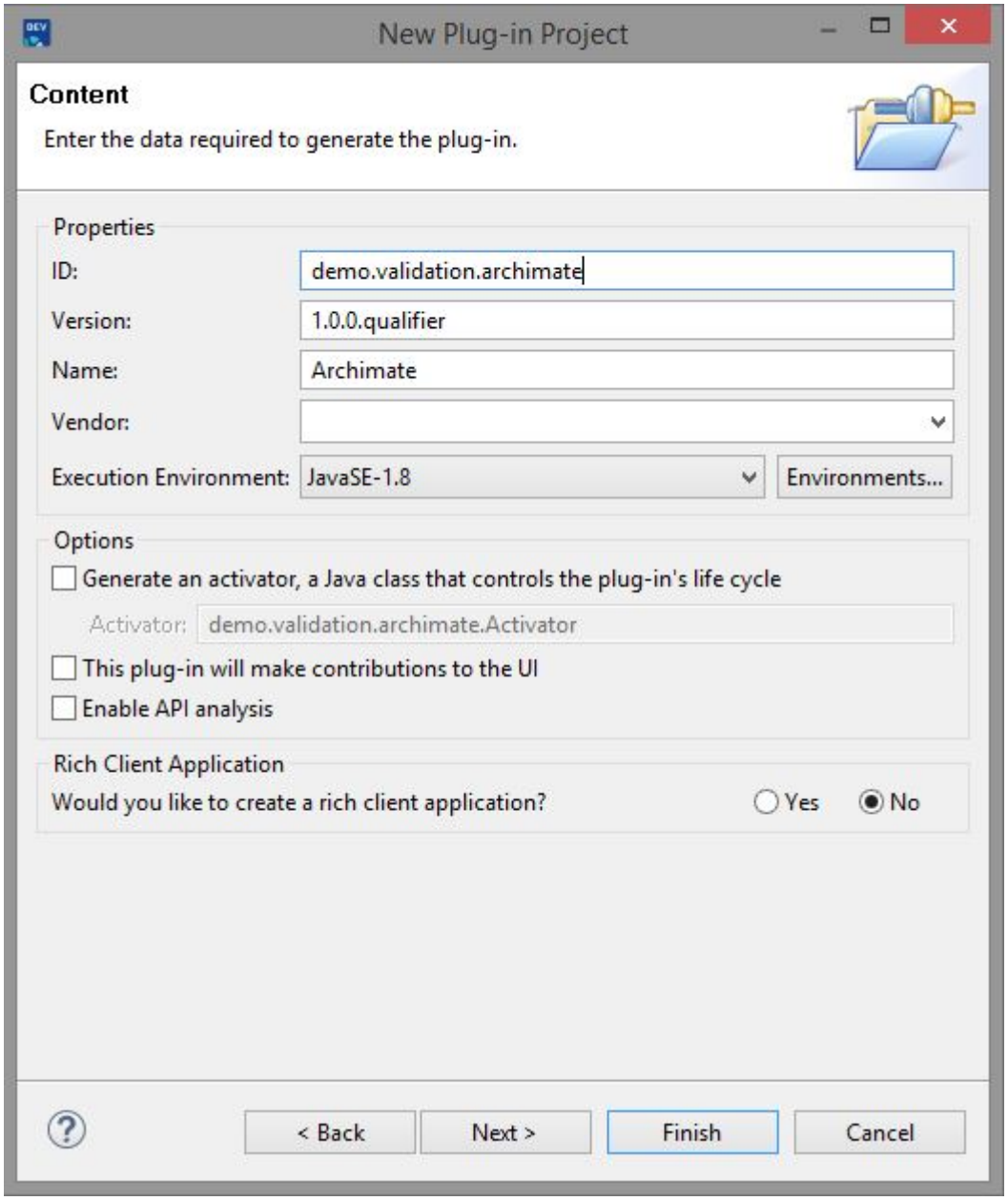
Une librairie de validation est un plug-in Eclipse simple exporté sous la forme d'un fichier jar.Une règle de validation est une méthode Java retournant un booléen et ayant en argument un sous-type de CDOObject.

Pour valider le modèle avec du code Java, nous allons créer une librairie qui dépend uniquement du métamodèle Archimate (fr.obeo.smartea.archimate) et optionnellement d'une API d'annotation (fr.obeo.smartea.core.common.api).

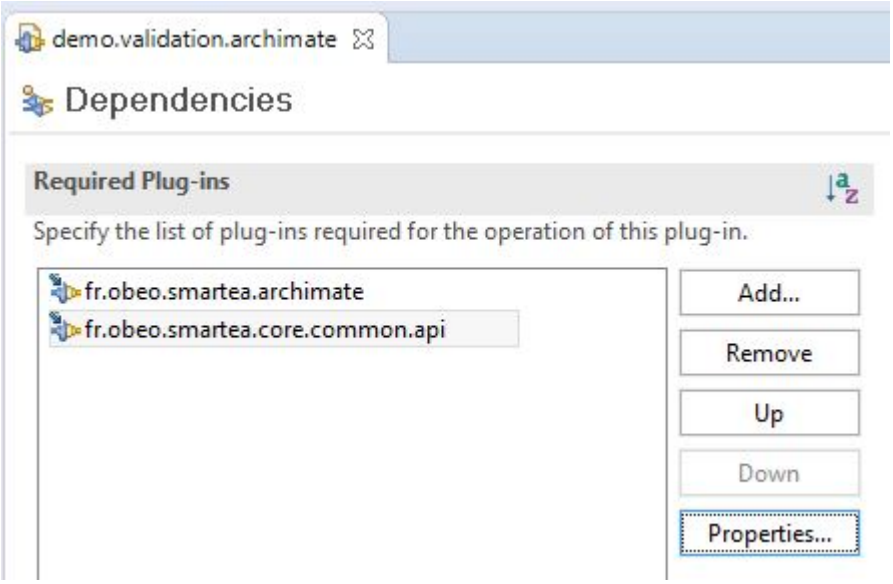
7.1. Créer une librairie de validation

Depuis l'IDE SmartEA, créer un plug-in Eclipse :

- New > Plug-in project > (Ne pas utiliser de template)



- Ajouter dans le MANIFEST.MF les **Required plug-ins** suivants :



- Créer un package **demo.validation.archimate**
- Créer une classe **ApplicationValidation.java** (l'ensemble des classes seront utilisées sur un ou plusieurs prismes)
- Créer plusieurs méthodes respectant le format **public boolean nomDeMaMethode(EObject param) { ... }**
  - Une méthode de ce format représente une règle de validation.
    - Son argument est le contexte
    - Son résultat booléen est le statut de validité
  - Une méthode n'est pas une règle si :
    - protected ou private
    - plus d'un argument
    - son argument est incompatible avec un EObject
- Les valeurs par défaut et les annotations
  - Par défaut, le **label** et la **description** de la règle de validation correspondront au nom de la méthode et le statut d'un échec sera "**information**".
  - Des annotations Java SmartEA sont disponibles pour préciser d'autres valeurs afin de personnaliser :
    - le label avec **@Label(«Mon label»)** ;
    - la description avec **@Description(«Ma description»)** ;
    - le statut d'un l'échec avec **@Error**, **@Warning** ou **@Information**.

Exemple d'une règle :

```
package demo.validation.archimate;

import fr.obeo.smartea.archimate.ApplicationComponent;
import fr.obeo.smartea.archimate.ArchimateElement;
import fr.obeo.smartea.archimate.DataObject;
import fr.obeo.smartea.core.common.api.annotation.Error;
import fr.obeo.smartea.core.common.api.annotation.Label;

public class ApplicationValidation {

   @Error
   @Label("A data object must be accessed by at least one application component")
   public boolean mustBeAtLeastInOneLocationWithOneActor(DataObject object) {
      for (ArchimateElement element : object.accessedBy()) {
         if (element instanceof ApplicationComponent) {
            return true;
         }
      }
      return false;
```

```
    }
}
```

Exporter le plug-in **demo.validation.archimate**

- Clic gauche sur le plug-in
- Export...
- Deployable plug-ins and fragments
- Spécifier un dossier pour retrouver l'export
- Finish

7.2. Importer une librairie de validation

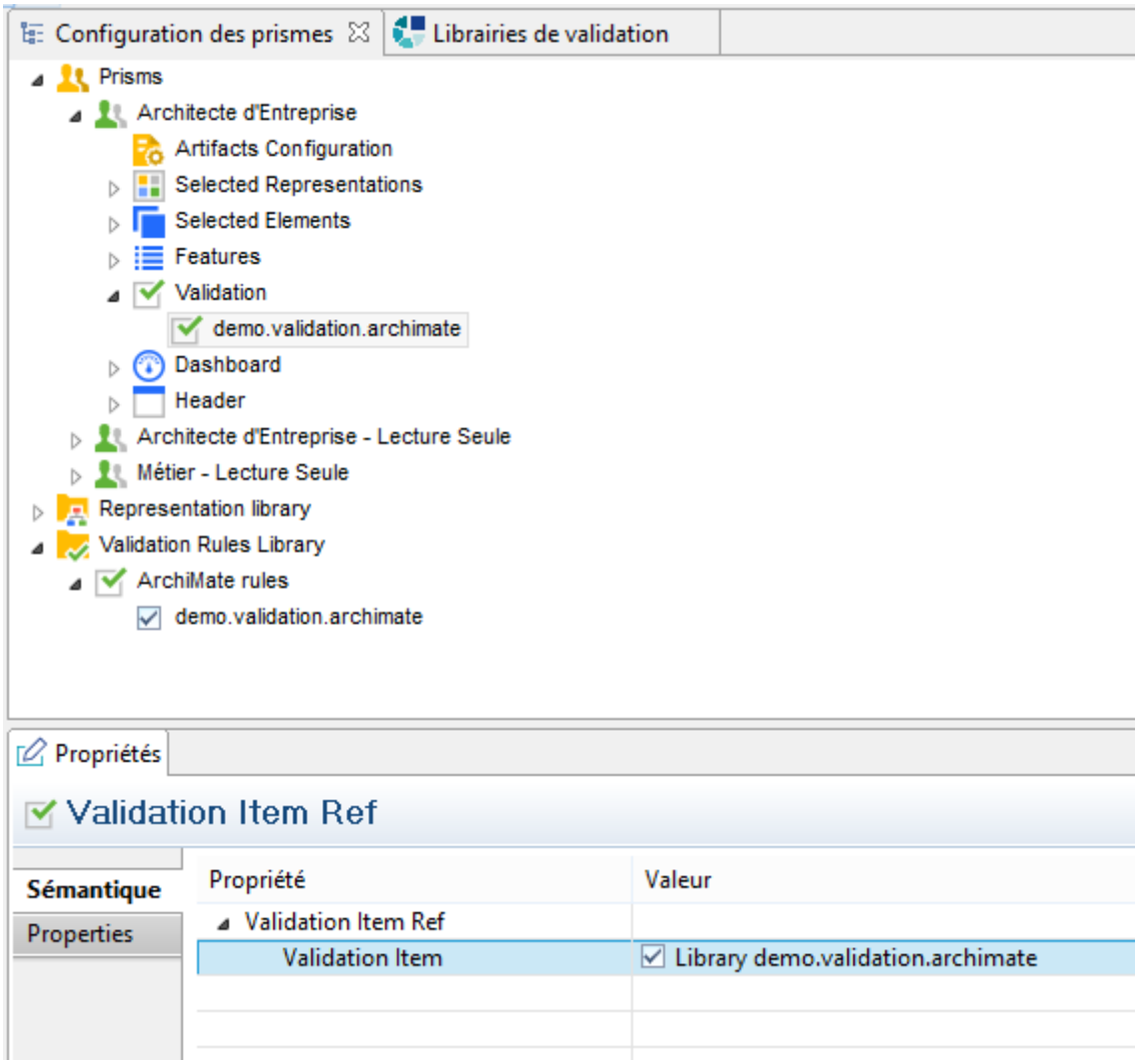
- Dans Validation > Librairies de validation

- Importer la librairie de validation

7.3. Référencer la nouvelle librairie dans l'éditeur de prisme

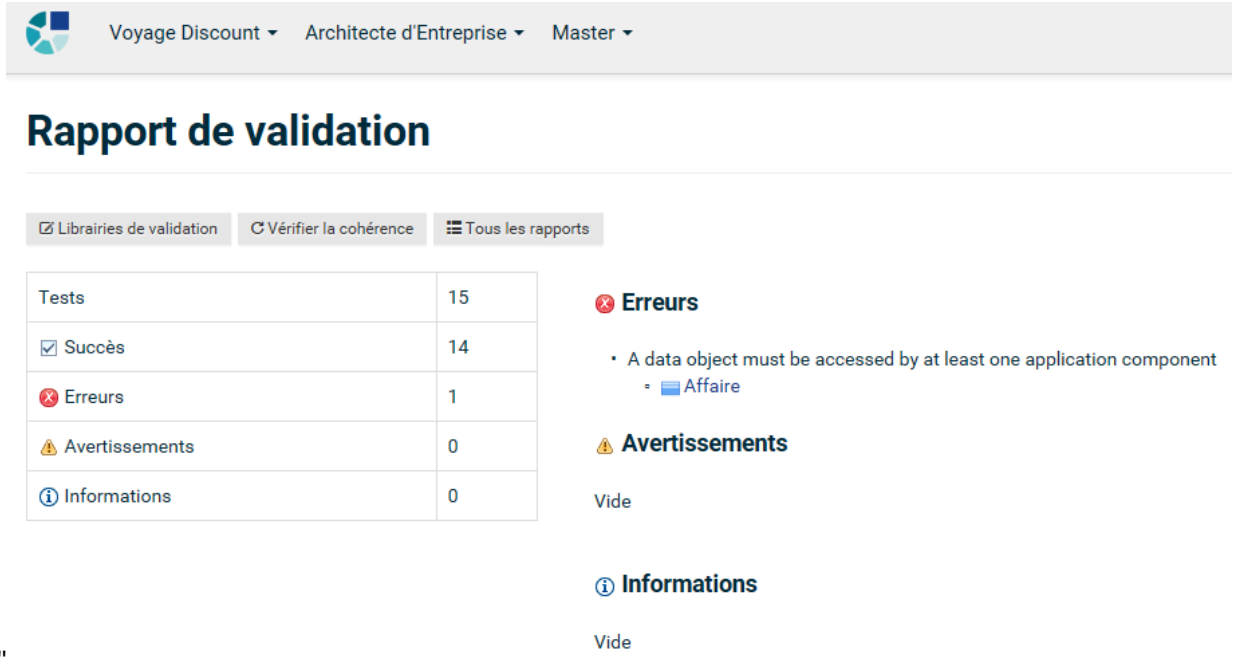
- Ouvrir l'éditeur de prisme
- Dans la partie "Validation Rules Library"
- Créer ou utiliser un "Validation Rules Set"
- Créer une "Validation Library"

- Dans la vue propriétés, sélectionner le nom du plug-in (demo.validation.archimate)
- Choisir le prisme «Enterprise Architect» par exemple
- rajouter une référence vers la librairie de validation



7.4. Lancer une validation

Lancer une validation sur le prisme "Enterprise Architect"



8. Glossaire des points d'extensions

Cette section recense tous les points d'extensions fournis par Obeo SmartEA.

8.1. API Commune

Pour pouvoir utiliser les points d'extension de l'API commune d'Obeo SmartEA, vous devez créer un plug-in conformément au paragraphe dédié en ajoutant dans la dépendance "`fr.obeo.smartea.core.common.api`".

8.1.1. Ajouter un générateur

La création de générateurs, notamment Acceleo, est détaillée dans cette section.

Un point d'extension unique permet ensuite de contribuer ces générateurs dans SmartEA de façon à pouvoir les matérialiser sous forme d'artefacts du référentiel.

Ce point d'extension permet à SmartEA de découvrir un générateur.

- Ouvrez le fichier `MANIFEST.MF` de votre projet de génération (Acceleo, Java ou M2Doc)
- Ajoutez `fr.obeo.smartea.core.common.api` dans l'onglet des dépendances
- Dans l'onglet des extensions, ajoutez une extension du type `fr.obeo.smartea.core.common.api.generator`
- Ajoutez une entrée de type :
  - `acceleo` pour déclarer un générateur Acceleo. Vous y référencerez dans l'attribut `class` la classe qui hérite de `org.eclipse.acceleo.engine.service.AbstractAcceleoGenerator`
  - `generic` pour déclarer un générateur Java. Vous y référencerez dans l'attribut `class` la classe qui implémente `fr.obeo.smartea.core.common.api.gen.IGeneratorExtension`
  - `m2doc` pour déclarer un générateur M2Doc. Vous y référencerez dans l'attribut `class` la classe qui implémente `fr.obeo.smartea.core.common.api.gen.IM2docGeneratorExtension`

8.1.2. Redéfinir des fournisseurs de labels et d'images

Il est possible de redéfinir l'`ItemProviderAdapterFactory` d'un EPackage donné, afin de modifier par exemple la façon dont sont calculés les labels ou les images pour une instance.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.common.api.itemProviderAdapterFactoryOverride`" avant de cliquer sur OK
- Dans la zone de droite choisissez :
  - le EPackage pour lequel s'appliquera votre personnalisation, par exemple `fr.obeo.smartea.core.basemm.BasePackage`
  - le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.common.MyAdapterFactory`"
- Implémentez votre factory. L'exemple suivant montre comment redéfinir les labels des répertoires dans SmartEA :

```
import org.eclipse.emf.common.notify.Adapter;

import fr.obeo.smartea.core.basemm.provider.BaseItemProviderAdapterFactory;
import fr.obeo.smartea.core.basemm.provider.FolderItemProvider;

public class CustomAdapterFactory extends BaseItemProviderAdapterFactory {
   @Override
   public Adapter createFolderAdapter() {
      if (folderItemProvider == null) {
         folderItemProvider = new FolderItemProvider(this) {
            @Override
```

```
            public String getText(Object object) {
               return "Custom_" + super.getText(object);
            }
         };
      }
      return folderItemProvider;
   }
}
```

8.1.3. Contribuer des services AQL

Le langage AQL, permettant la navigation dans un modèle, est utilisable à plusieurs endroits dans SmartEA :

- pour écrire des Smart Requests
- dans la barre de recherche
- pour définir des générateurs M2Doc
- dans l'interpréteur de requêtes du modeleur

Attention, le service AQL que nous déclarons ici est différent du service java Sirius. Un service Java Sirius est uniquement disponible dans une session Sirius.

Pour contribuer des services Java spécifiques accessibles depuis n'importe quelle expression AQL dans SmartEA, vous pouvez implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.common.api.aql`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.common.MyServices`"
- Implémentez votre classe de services. Par exemple :

```
public class MyServices {

   public Collection<Folder> allSubFolders(Folder folder) {
      List<Folder> res = new ArrayList<>();
      res.addAll(folder.getFolders());
      for (Folder subFolder : folder.getFolders()) {
         res.addAll(allSubFolders(subFolder));
      }
      return res;
   }
}
```

Pour tester ce service une fois déployé, vous pouvez par exemple l'invoquer sur le premier répertoire de la ressource principale en tapant dans la barre de recherche l'expression `aql:self.getContents()->first().allSubFolders()`.

Attention, si votre service traite des données ne provenant pas du modèle sémantique vous devez inclure l'instruction suivante afin que son résultat ne soit pas mis en cache par le serveur :

Les services qui étendent la classe `fr.obeo.smartea.core.common.util.AbstractAQLCommonService` auront accès à des instances de :

- `fr.obeo.smartea.core.common.api.IdManager`
- `fr.obeo.smartea.core.artifact.ArtifactRepository`
- `com.google.inject.Provider<CDOView>`

```
AQLUtils.NO_CACHE_EXECUTION_FLAG.set(true);
```

Par exemple, si le résultat de votre service dépend de la date actuelle alors le résultat du service change même si le modèle sémantique ne change pas. Le serveur ne doit donc pas mettre en cache le résultat de votre service. Il est donc nécessaire de rajouter l'instruction `AQLUtils.NO_CACHE_EXECUTION_FLAG.set(true)` dans le code de votre service.

8.2. API du serveur

Pour pouvoir utiliser les points d'extension serveur Obeo SmartEA, vous devez créer un plug-in conformément au paragraphe dédié en ajoutant dans la dépendance "`fr.obeo.smartea.core.server.api`".

8.2.1. Modifier la gestion des utilisateurs

Par défaut, Obeo SmartEA utilise un fichier de configuration (`users.yml`) ou un connecteur LDAP pour authentifier les utilisateurs.

Il est possible de remplacer ce mécanisme en fournissant sa propre implémentation du point d'extension `fr.obeo.smartea.core.server.api.userProvider`.

Une fois contribué, celui-ci est utilisé en priorité, même si le fichier de configuration définit autre chose (en particulier si le connecteur LDAP y est activé).

Pour définir votre propre gestionnaire d'utilisateurs :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.server.api.userProvider`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.MyUserProvider`"
- Implémentez votre gestionnaire d'utilisateur

Vous pouvez vous reporter à l'API [*api/index.html?fr/obeo/smartea/core/server/api/IUserProviderExtension.html*] pour plus de détails.

8.2.2. Modifier la gestion des accès

Il est possible de modifier la gestion de l'accès aux projets et aux prismes.

Pour définir votre propre gestionnaire d'accès :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.server.api.userAccessManager`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.MyUserAccessManager`"
- Implémentez votre gestionnaire d'accès

Vous pouvez vous reporter à l'API [*api/index.html?fr/obeo/smartea/core/server/api/IUserAccessManagerExtension.html*] pour plus de détails.

8.2.3. Configurer les objets de contexte des artefacts de la page d'accueil

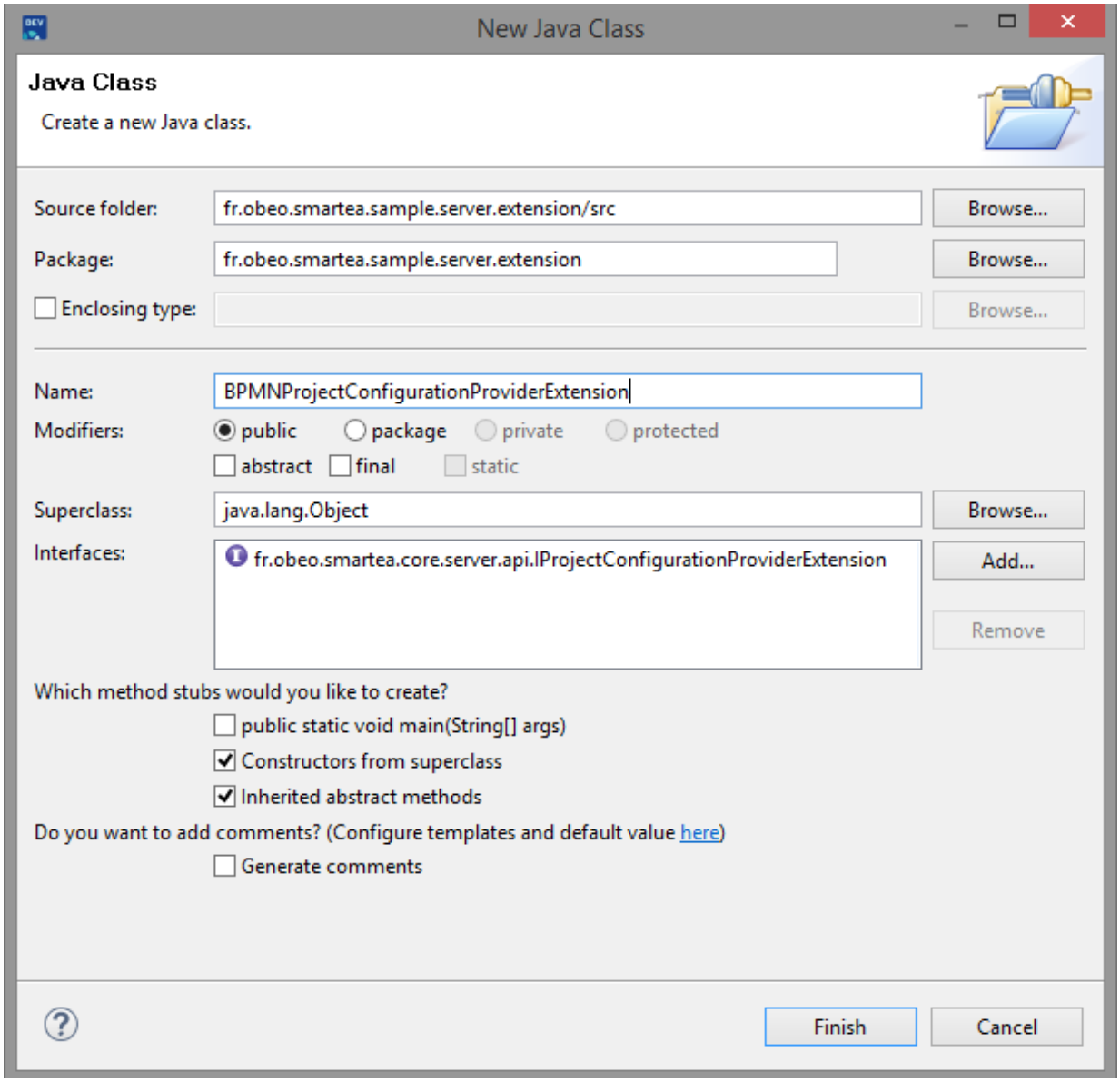
Sur la page d'accueil d'Obeo SmartEA, il est possible de créer des artefacts portant sur certains objets de contextes qui ont été définis ou non au préalable.Lorsqu'aucun objet de contexte n'a été défini explicitement pour la page d'accueil, Obeo SmartEA retient la racine de la ressource sémantique.

Si l'on souhaite prendre la main pour indiquer à Obeo SmartEA quel(s) objet(s) doit(vent) être considéré(s) pour la home page, il faut contribuer au point d'extension `fr.obeo.smartea.core.server.api.projectConfigurationProvider`.

Pour illustrer nos propos nous prendrons l'exemple du méta-modèle BPMN. Nous imaginerons que nous souhaitons que les artefacts portant sur l'élément racine `Definitions` de BPMN soient directement accessibles depuis la page d'accueil.

Pour y parvenir :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.server.api.projectConfigurationProvider`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, par exemple "`com.mycompany.myextensions.BPMNProjectConfigurationProvider`"



- Implémentez votre composant :

```
public class BpmnProjectConfigurationProvider implements
 IProjectConfigurationProviderExtension {
```

```
   @Override
   public Set<CDOObject> getRootElements(String projectId, String prismId, CDOResource
 semanticResource) {
      Set<CDOObject> roots = new HashSet<CDOObject>();
      for (final EObject object : semanticResource.getContents()) {
         if (object instanceof Definitions) {
            roots.add((CDOObject)object);
         }
      }
      return roots;
   }

   @Override
   public boolean appliesOn(String projectId, CDOResource semanticResource) {
      for (EObject object : semanticResource.getContents()) {
         if (object instanceof Definitions) {
            return true;
         }
      }
      return false;
   }

   @Override
   public Priority getPriority() {
      return Priority.LOWEST;
   }

}
```

Vous pouvez vous reporter à l'API [*api/index.html?fr/obeo/smartea/core/server/api/IProjectConfigurationProviderExtension.html*] pour plus de détails.

8.2.4. Ajouter des services HTTP

Obeo SmartEA permet d'embarquer dans le serveur des services capables d'interagir avec le référentiel SmartEA.

Pour ajouter un service, utilisez le point d'extension `fr.obeo.smartea.core.server.api.httpservice` et créez une classe Java qui implémente l'interface `fr.obeo.smartea.core.server.api.services.IHttpServiceExtension`.

Vous pouvez vous reporter à l'API [*api/index.html?fr/obeo/smartea/core/server/api/services/IHttpServiceExtension.html*] pour plus de détails.

8.2.5. Ecouter les commits

Ce point d'extension permet de définir un composant apte à réagir à l'ensemble des modifications apportées sur le référentiel SmartEA d'un projet.

Pour définir votre propre composant d'écoute des commits :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.server.api.db.commitInfoHandler`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.MyCommitInfoHandler`"
- Implémentez votre composant

Vous pouvez vous reporter à l'API [*api/index.html?fr/obeo/smartea/core/server/api/db/DbCommitInfoHandlerExtension.html*] pour plus de détails.

8.2.6. Paramétrer l'import/export Excel en ajoutant un résolveur d'identifiants dérivés

L'import/export Excel repose sur un résolveur d'identifiants dérivés qui est pris en compte de deux façons :

- lors d'un export, si l'identifiant d'un objet est dérivé et qu'il est supporté par un résolveur, la feuille Excel n'aura pas de colonne pour l'identifiant
- lors d'un import, si une feuille a dans ses métadonnées l'option «derivedId=true», un résolveur sera utilisé pour calculer l'identifiant d'un objet à partir des valeurs d'une ligne de la feuille

Pour définir une stratégie spécifique pour résoudre les identifiants, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.server.api.xls.derivedIdResolver`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.server.DemoDerivedIdResolver`"
- Implémentez votre résolveur

L'exemple ci-dessous montre un exemple d'implémentation de l'interface `IXLSDerivedIdResolver`.Ici l'identifiant dérivé des éléments de type `MyTypeWithDerivedId` est calculé à partir du champ `location` de celui-ci, préfixé par l'identifiant du conteneur.

```
public class DemoDerivedIdResolver implements IXLSDerivedIdResolver {

   public boolean canResolve(EClass eClass) {
      return eClass.equals(DemoPackage.eINSTANCE.getMyTypeWithDerivedId());
   }

   public String resolve(EClass eClass, EObject container, Map<String, String> rowValues) {
      return EcoreUtil.getID(container) + '/' + rowValues.get("location");
   }

}
```

8.2.7. Paramétrer l'export Excel en spécifiant le contenu à exporter

Par défaut, l'export Excel exporte les objets contenus par le contexte d'export. Ce point d'extension vous permet d'exporter également des objets contenus par les objets exportés, en spécifiant quelles références de contenance doivent être explorées.Les sous-objets exportés le seront dans des feuilles de type «content», qui ont comme première colonne l'identifiant du conteneur associé à l'objet défini par une ligne.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.server.api.xls.exportContentFilter`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.server.DemoXLSExportFilter`"
- Implémentez votre filtre

L'exemple ci-dessous montre un exemple d'implémentation de l'interface `IXLSExportContentFilter`, qui permettra d'exporter les éléments contenus par la référence «myUsefulContent» des objets de type `MyTypeWithUsefulContent`.

```
public class DemoXLSExportFilter implements IXLSExportContentFilter {

   public boolean accept(EClass eClass, EStructuralFeature feature) {
```

```
      return eClass.equals(DemoPackage.eINSTANCE.getMyTypeWithUsefulContent())
         && "myUsefulContent".equals(feature.getName());
   }

}
```

8.2.8. Se brancher sur l'initialisation du serveur

Il est possible de contribuer du code qui sera exécuté au lancement du serveur.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.server.api.initializer`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.server.MyInitializer`"
- Implémentez le code d'initialisation spécifique :
  - dans la méthode `fr.obeo.smartea.core.server.api.IInitializerExtension.initialize()`, il sera alors exécuté au lancement du serveur
  - dans la méthode `fr.obeo.smartea.core.server.api.IInitializerExtension.initializeProject(String)`, il sera alors exécuté à l'initialisation d'un projet donné

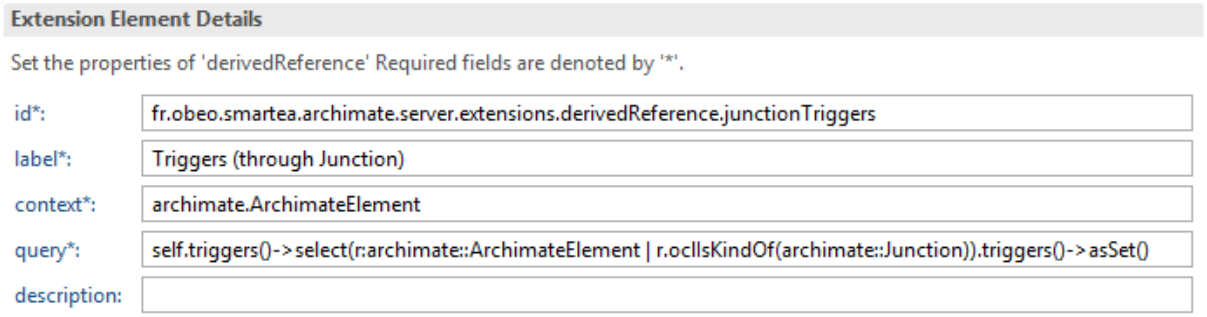
8.2.9. Contribuer des références dérivées

Il est possible d'ajouter des références dérivées spécifiques qui seront ensuite activables dans le prisme pour un type d'élément donné.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.server.api.derivedReference`" avant de cliquer sur OK
- Dans la zone de droite choisissez :
  - un identifiant unique pour cette référence dérivée, par exemple "`com.mycompany.myextensions.server.myderivedreference`"
  - un label "`My Derived Reference`"
  - un contexte qui définira sur quel type d'éléments cette référence est applicable. La valeur à renseigner doit être constituée du nom du EPackage contenant la EClass contexte, suivi de "`.`" puis du nom de la EClass. Par exemple : "`mypackage.MyEClass`"
  - la requête à évaluer pour obtenir le ou les éléments cibles de la référence dérivée
  - une description optionnelle de la référence dérivée

Voici un exemple de référence dérivée déclarée pour le métamodèle Archimate :



Une fois l'extension de la référence dérivée déployée sur un serveur SmartEA, la référence peut être appliquée sur un élément du prisme.Il est possible de définir, en plus du contexte, une contrainte sur l'applicabilité de la référence sur un type donné ou d'internationaliser son label en utilisant une extension du modeleur.

8.2.10. Implémenter un «fournisseur» d'artefact (Non supporté)

Le concept d'artefact qui peut être produit est générique et extensible.

Ce mécanisme repose sur l'organisation suivante :

- Des «fournisseurs» d'artefacts peuvent être configurés au sein d'Obeo SmartEA, chacun possédant un identifiant qui lui est propre. Aujourd'hui, Obeo SmartEA embarque par défaut plusieurs fournisseurs pré-définis, notamment : `generator`, `sirius` et `smartrequest`. Le composant technique correspondant est le suivant : `fr.obeo.smartea.core.server.api.artifacts.IArtifactTypeProviderExtension`
- Chaque «fournisseur» rassemble un ensemble de «types d'artefact» (`fr.obeo.smartea.core.artifact.ArtifactType`), chaque type se caractérisant principalement par :
  - un identifiant (unique pour le «fournisseur» auquel il appartient)
  - un contexte (`org.eclipse.emf.ecore.EClass`) qui conditionne les instances sur lesquelles ce type d'artefact pourra être instancié
- Chaque «type d'artefact» rassemble un ensemble d'instances d'artefact (`fr.obeo.smartea.core.server.api.artifacts.IArtifactInstance`), chacune se caractérisant principalement par :
  - un identifiant (unique au sein du type auquel elle se rapporte)
  - un contexte (`org.eclipse.emf.cdo.CDOObject`) qui correspond à l'instance à laquelle se rapporte l'artefact

Aujourd'hui, ce mécanisme et le point d'extension correspondant (`fr.obeo.smartea.core.server.api.artifacts.provider`) n'est pas encore ouvert à la contribution.

8.2.11. Redéfinir le contenu de la page d'accueil multi projets

Le contenu de la page d'accueil multi-projets peut être redéfini par contribution au point d'extension `fr.obeo.smartea.core.server.api.multiProjectHomepage`.

Pour ce faire :

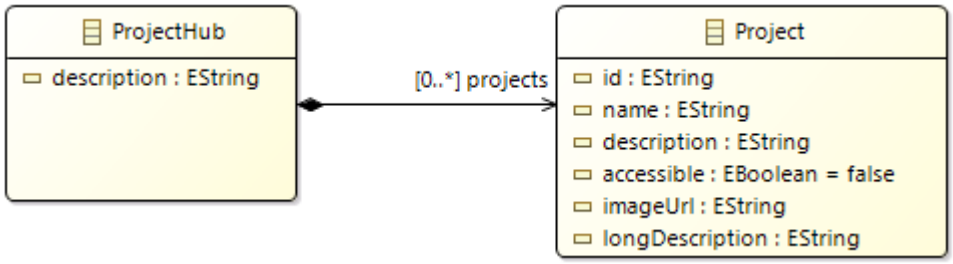
- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`"
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.server.api.multiProjectHomepage`" et cliquez sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, par exemple "`com.mycompany.myextensions.server.MultiProjectHomePageService`"

La classe d'implémentation doit implémenter l'interface "`fr.obeo.smartea.core.server.api.IMultiProjectHomePageService`". La méthode `generateHomePageContents` de cette interface doit être implémentée de manière à générer le contenu de la page d'accueil sous forme de fichiers html et images dans le dossier "`outputFolder`". `projectHub` est la structure de données qui porte les informations données à disposition pour cette génération.

La méthode `generateHomePageContents` prend deux paramètres en entrée :

- `projectHub` : Modèle de description du contexte de la page d'accueil multi projets.
- `outputFolder` : Dossier dans lequel le contenu de la page d'accueil doit être généré.

Le modèle de description est conforme à ce méta-modèle :



L'instance de `ProjectHub` donnée porte la description commune à tous les projets telle que définie par la propriété `cross.project.home.description` du ficher «etc/application.conf» ainsi que la liste de projets accessibles à l'utilisateur connecté.

L'exhaustivité de cette liste est soumise à la valeur de la propriété `cross.project.home.showInaccessibleProject` du ficher «etc/application.conf».

Chaque projet porte les attributs suivants :

- `id` : Identifiant du projet tel que défini par le nom des fichiers «etc/projects/< project-id >.conf».
- `name` : Nom du projet tel que défini par la propriété `label` du fichier «etc/projects/< project-id >.conf» correspondant.
- `description` : Description du projet telle que définie par la propriété `description` du fichier «etc/projects/< project-id >.conf» correspondant.
- `accessible` : Valeur booléenne positionnée à «true» si le projet est accessible à l'utilisateur courant, à «false» sinon.
- `imageUrl` : URL de l'image du projet pouvant être utilisée dans une balis HTML «img».
- `longDescription` : Description du projet telle que définie par la propriété `longDescription` du fichier «etc/projects/< project-id >.conf» correspondant.

Le dossier `outputFolder` est destiné à accueillir les fichiers HTML et images consituant le contenu de la page d'accueil. Ce dossier doit être peuplé avec à minima un fichier nommé «index.html» constituant le point d'entrée de la page d'accueil. Ce document html peut contenir des hyperliens vers d'autres fichiers générés dans le même dossier `outputFolder` ou l'un de ses sous dossiers.

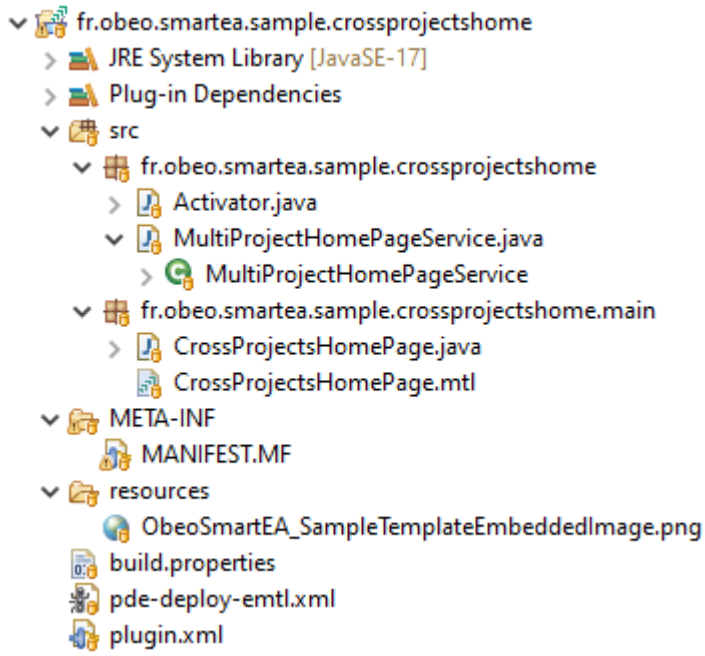
Seules les extensions de fichier suivantes sont acceptées :

- «htm» ou «html» pour les fichiers html,
- «gif» pour les images GIF,
- «jpg» ou «jpeg» pour les images Jpeg,
- «png» pour les images PNG.

L'implémentation conseillée de la méthode `generateHomePageContents` fait appel à un générateur Acceleo qui génère les fichiers html à partir du modèle de description `ProjectHub`.

Reportez vous à la section Générateurs Acceleo pour la création d'un projet Acceleo, en utilisant l'URI de méta-modèle "`http://www.obeo.fr/smartea/projecthub/1.0.0`".

La structure du projet doit être conforme à cet exemple :



Où :

- `MultiProjectHomePageService` est le nom de la classe qui implémente l'interface `IMultiProjectHomePageService`,
- `CrossProjectHomePage` est le nom du générateur Acceleo (saisi dans l'assistant de création de projet Acceleo).

L'implémentation de la méthode `generateHomePageContents` peut alors simplement appeler le générateur Acceleo :

```
  @Override
  public void generateHomePageContents(ProjectHub projectHub, File outputFolder) throws
IOException {

     // Generate html contents
     List<String> arguments = new ArrayList<String>();
     CrossProjectsHomePage generator = new CrossProjectsHomePage(projectHub, outputFolder,
arguments);
     generator.doGenerate(new BasicMonitor());

  }
```

Des ressources statiques additionnelles (comme des images) peuvent être embarquées dans le plug-in. Dans l'exemple ci-dessus, un dossier «resources» a été créé pour accueillir une image. Dans ce cas, le dossier «resources» doit être ajouté à la liste des éléments faisant partie du «Binary Build» (dans l'onglet «Build» de l'éditeur du fichier «build.properties»).

Les ressources statiques peuvent être copiées dans le dossier `outputFolder` de la manière suivante :

```
     // Add other contents
     Bundle bundle = Activator.getDefault().getBundle();
     URL sampleEmbeddedImageUrl = bundle.getEntry("/resources/
ObeoSmartEA_SampleTemplateEmbeddedImage.png");
     InputStream in = sampleEmbeddedImageUrl.openStream();
     Files.copy(in,
outputFolder.toPath().resolve("ObeoSmartEA_SampleTemplateEmbeddedImage.png"));
     in.close();
```

Voici un exemple de générateur (fichier «CrossProjectsHomePage.mtl») :

```
[comment encoding = UTF-8 /]
[module CrossProjectsHomePage('http://www.obeo.fr/smartea/projecthub/1.0.0')]

[template public generateCrossProjectsHomePage(projectHub : ProjectHub)]
[comment @main/]
[file ('index.html', false, 'UTF-8')]
<html>
   <head>
      <style>
         table {
           font-family: arial, sans-serif;
           border-collapse: collapse;
           width: 100%;
         }

         td, th {
           border: 1px solid #dddddd;
           text-align: left;
           padding: 8px;
         }

         tr:nth-child(even) {
           background-color: #dddddd;
         }
      </style>
   </head>
   <body>
      <div style="overflow: hidden;">
         <img src="welcome/ObeoSmartEA_SampleTemplateEmbeddedImage.png" alt="Sample template
 embedded image." style="float: left;">
         <a href="logout" style="float: right;">Logout</a>
      </div>
      <h2>Cross project home page</h2>
      <table>
         [generateProjectsTableHeader()/]
         [for (project : Project | projectHub.projects)]
         [generateProjectsTableRow()/]
         [/for]
      </table>
   </body>
</html>
[/file]
[for (project : Project | projectHub.projects)]
[generateProjectDetailsPage()/]
[/for]
[/template]

[template private generateProjectsTableHeader(project : ProjectHub)]
   <tr>
      <th>Project</th>
      <th>ID</th>
      <th>Description</th>
      <th>Accessible</th>
      <th>Link</th>
      <th>Details</th>
   </tr>
[/template]

[template private generateProjectsTableRow(project : Project)]
   <tr>
      <td>[project.name/]</td>
      <td>[project.id/]</td>
      <td>[project.description/]</td>
      <td>[(if project.accessible then 'Yes' else 'No' endif)/]</td>
```

```
      <td><a href="[project.id/]"><img src="[project.imageUrl/]" alt="[project.name/] image."
 width="80"></a></td>
      <td><a href="welcome/[project.id/]-details.html">[project.name/] Details</a></td>
   </tr>
[/template]

[template private generateProjectDetailsPage(project : Project)]
[file (project.id+'-details.html', false, 'UTF-8')]
<html>
   <body>
      <h2>[project.name/] details page</h2>
      <img src="[project.imageUrl/]" alt="[project.name/] image.">
      <ul>
         <li>Name: [project.name/]</li>
         <li>Id: [project.id/]</li>
         <li>Description: [project.description/]</li>
         <li>Accessible: [(if project.accessible then 'Yes' else 'No' endif)/]</li>
         <li><a href="../[project.id/]">Navigate to project</a></li>
      </ul>
      <a href="../welcome">Back</a>
   </body>
</html>
[/file]
[/template]
```

Cet exemple illustre l'utilisation des différentes propriétés du modèle `ProjectHub`, l'utilisation des images de projets, d'images statiques embarquées dans le générateur, de styles conditionnels, ainsi que de pages annexes.

Pour plus d'information sur l'écriture de templates Acceleo, reportez vous à la documentation en ligne : Acceleo.

8.2.11.1. Gestion du cache

Le contenu généré dans le `outputFolder` est automatiquement mis en cache par SmartEA de manière à limiter l'utilisation CPU du serveur. Ce cache est géré selon les droits d'accès de l'utilisateur (les utilisateurs ayants les mêmes droits auront accès au même contenu généré).

Pour la mise au point du template Acceleo dans un environnement de test, il peut être utile d'invalider le cache. Le paramètre "`clearcache`" peut alors être ajouté à l'URL du service REST de la manière suivante :

```
http://localhost:8080/smartea/welcome?clearcache=current
```

Ce paramètre peut prendre les valeurs suivantes :

- "`current`" ou pas de valeur : Invalide le cache pour l'utilisateur connecté
- "`all`" : Invalide le cache pour tous les utilisateurs

8.3. API du modeleur

Pour pouvoir utiliser les points d'extension du modeleur, vous devez créer un plug-in conformément au paragraphe dédié en ajoutant dans la dépendance "`fr.obeo.smartea.core.rcp.api`".

8.3.1. Réagir au changements de projet, branche et/ou prisme

Il est possible de déclencher des traitements lors de chaque chargement de page coté modeleur. En pratique cela permet de réagir à tout changement de projet, branche et/ou prisme par l'utilisateur.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"

- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.pagetrigger`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MyPageTrigger`"
- Implémentez votre trigger

8.3.2. Ajouter un feedback utilisateur listant les objets avant suppression

Par défaut quand un ou plusieurs objets sont supprimés depuis le modeleur, l'utilisateur n'a pas de feedback particulier sur la liste des éléments supprimés (la suppression d'un objet pouvant entrainer la suppression d'autres objets en cascade). En contribuant à ce point d'extension, il est possible de notifier à l'utilisateur la liste des éléments qui sont sur le point d'être supprimés.

Cela offre à l'utilisateur la possibilité de voire cette liste mais aussi une chance d'annuler la suppression.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.removedObjectsUIFeedbackFilter`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MyFilter`"
- Implémentez votre filtre

Au lieu d'implémenter un filtre complet, vous pouvez repartir d'une implémentation de base que vous pouvez sous-classer (voir Javadoc [*api/index.html?fr/obeo/smartea/core/rcp/api/ui/IRemovedObjectsUIFeedbackFilterExtension.WhiteListBasedImpl.html*]).

Exemple :

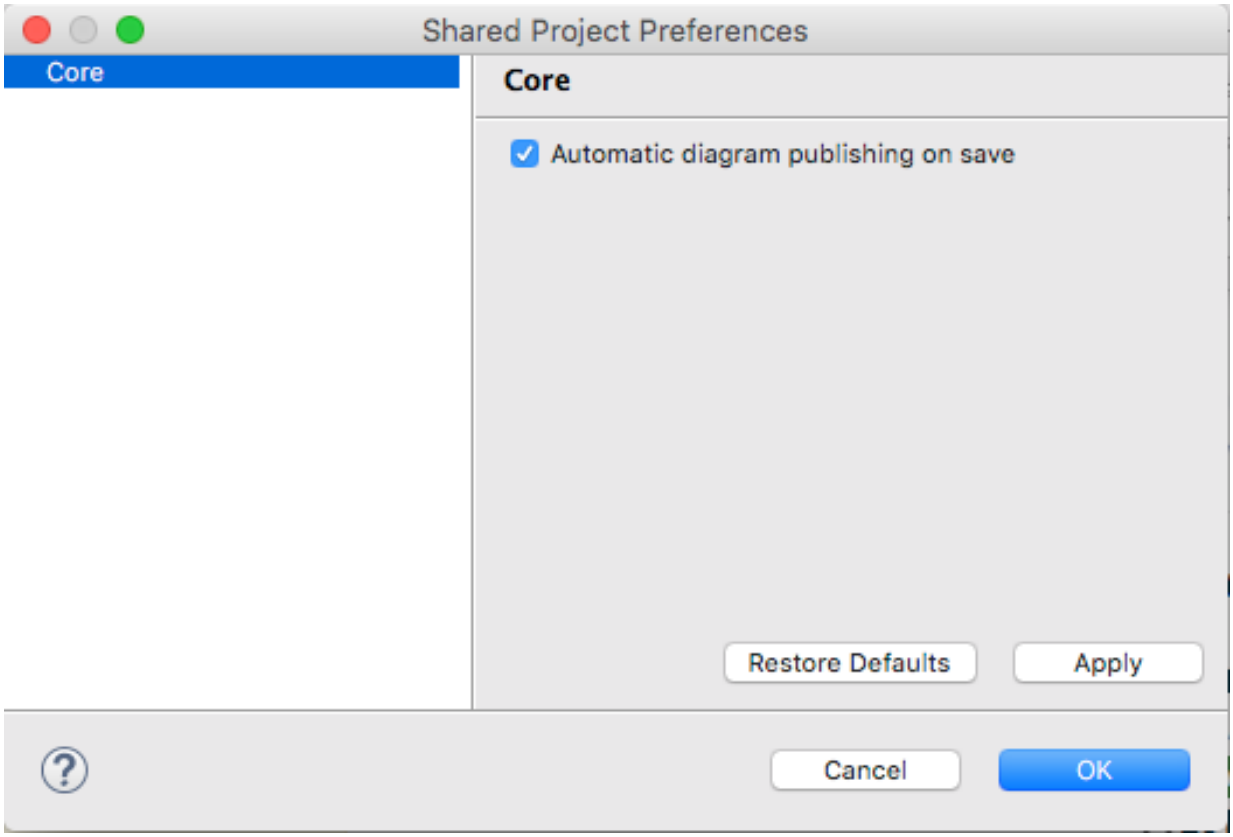
```
public class MyFilter extends IRemovedObjectsUIFeedbackFilter.WhiteListBasedImpl {

   public MyFilter() {
      super(new HashSet<EPackage>(
         // Keep all Archimate EClasses
         Collections.singleton(ArchimatePackage.eINSTANCE)),
         // No need to add anymore EClasses
         Collections.<EClass> emptySet());
   }

}
```

8.3.3. Contribuer des préférences projet

Il est possible d'ajouter des pages de préférences qui vont permettre de gérer des préférences spécifiques à un projet donné.



Une fois les pages de préférence créées, les préférences qu'elle permet de modifier peuvent être utilisées par exemple depuis des représentations Sirius (dans des services Java) au travers de l' API de gestion des préférences [*api/index.html?fr/obeo/smartea/core/common/api/IProjectPreferences.html*] .

Pour ajouter une page de préférences :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.projectPreferencePage`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MyProjectPreferencePagePage`"
- Implémentez votre page de préférences

Exemple :

```
import org.eclipse.jface.preference.BooleanFieldEditor;
import org.eclipse.jface.preference.FieldEditorPreferencePage;

public class MyProjectPreferencePage extends FieldEditorPreferencePage {

   public SmartEAImagePreferencePage() {
      super(GRID);
   }

   public void createFieldEditors() {
      addField(new BooleanFieldEditor("automaticPublish",
            "Automatic publishing", getFieldEditorParent()));
   }

}
```

Note : les pages de préférences peuvent être imbriquées.

Il est également possible de définir des valeurs par défaut pour ces préférences (côté serveur). C'est décrit dans le chapitre «Gestion des préférences projet partagées» du guide technique.

8.3.4. Contribuer des préférences utilisateur

Les préférences projet sont partagées par tous les utilisateurs. Les préférences utilisateur elles, sont propres à un modeleur donné.

Pour rajouter des préférences utilisateur, vous devez simplement les développer comme des préférences Eclipse classiques puis les contribuer à Obeo SmartEA en implémentant le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.contributedPreferencePage`" avant de cliquer sur OK
- Dans la zone de droite renseignez l'identifiant de votre page de préférence

8.3.5. Restreindre le glisser-déposer dans l'explorateur de modèle

Vous pouvez restreindre les capacités du drag & drop dans l'explorateur de modèle. On peut ainsi empêcher un déplacement d'objet sémantiquement incorrect.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.explorerDropFilter`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MyDropFilter`"
- Implémentez votre filtre

Dans l'exemple suivant on interdit le déplacement de toute relation Archimate :

```
import org.eclipse.core.runtime.IStatus;
import org.eclipse.core.runtime.Status;
import fr.obeo.smartea.archimate.Relationship;
import fr.obeo.smartea.core.rcp.api.ui.IExplorerDropFilter;

public class MyDropFilter implements IExplorerDropFilter {
   public IStatus canDrop(EObject source, EObject target, String projectId, String branchId,
 String prismId,
         String userId) {
      IStatus status = new Status(IStatus.OK, "my.plugin", null);
      if (source instanceof Relationship) {
         status = new Status(IStatus.WARNING, "my.plugin", "Cannot move relationships");
      }
      return status;
   }
}
```

8.3.6. Ajouter des filtres spécifiques à la barre de recherche de l'explorateur de modèle

L'explorateur de modèles propose des filtres prédéfinis permettant de focaliser son contenu sur un ou plusieurs types donnés, par exemple. Vous pouvez ajouter des filtres spécifiques à la barre de recherche de l'explorateur de modèle.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension

- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.explorerSearchFilterExtension`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MySearchFilter`"
- Implémentez votre filtre

Dans l'exemple suivant on filtre les éléments en fonction d'un attribut spécifique.

```
public class MySearchFilter implements IExplorerSearchFilter {

   @Override
   public boolean appliesOn(String projectId, String branchId, String prismId, String userId)
 {
      // you can restrict the availability of the filter here
      return true;
   }

   @Override
   public String getName() {
      // the name that will be displayed in the search bar filter menu
      return "My search filter";
   }

   @Override
   public boolean apply(Object element) {
      // Here we restrict the search to elements which holds a specific flag, "visible"
      return element instanceof MyType && ((MyType)element).isVisible();
   }

}
```

8.3.7. Personnaliser les menus de création et les filtres de recherche de l'explorateur de modèles

Vous pouvez contribuer des menus de création spécifiques à l'explorateur de modèle, qui sont aussi exploités par la barre de recherche pour fournir les filtres correspondants.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.explorerMenuConfigurerExtension`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MyMenuConfigurer`"
- Implémentez votre «configurateur» de menus selon l'interface `fr.obeo.smartea.core.rcp.api.ui.IExplorerMenuConfigurerExtension`

Dans l'exemple suivant on peuple le menu en fonction du projet et du prisme.

```
public void populateRootMenuLevel(IMenuLevel menu, CDOObject container, String projectId,
 String prismId, String branchId) {
    if ("MyProjectId".equals(projectId) && "MyPrismId".equals(prismId)) {
       final IMenuLevel cat1 = menu.getOrCreateChildLevel("Cat1");
       cat1.addConstructibleTypes(MyPackage.eINSTANCE.getMyType11(),
 MyPackage.eINSTANCE.getMyType12());

       final IMenuLevel cat2 = menu.getOrCreateChildLevel("Cat2");
       cat2.addConstructibleTypes(MyPackage.eINSTANCE.getMyType12());
```

```
       menu.addConstructibleTypes(MyPackage.eINSTANCE.getMyType3(),
 MyPackage.eINSTANCE.getMyType4());
    } else {
       // no constructible types.
    }
}
```

L'interface `fr.obeo.smartea.core.rcp.api.ui.IExplorerMenuConfigurerExtension` permet également d'implémenter deux méthodes :

- `boolean appliesOn(CDOObject container, String projectId, String branchId, String prismId, String userId)` : détermine si le `container` (sélection courante) déclenche ou pas la création du menu spécifique. Vous pouvez par exemple tester le type de l'objet courant, ou simplement renvoyer `true`.
- `void onCreate(CDOObject cdoObject)` : cette méthode est lancée lors de la création d'un objet à l'aide du menu spécifique. Vous pouvez laisser l'implémentation de cette méthode vide.

8.3.8. Modifier les éditeurs Sirius

Il est possible de se brancher sur l'ouverture des éditeurs Sirius pour modifier l'instance de org.eclipse.sirius.ui.business.api.dialect.DialectEditor.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.dialectEditorConfigurer`" avant de cliquer sur OK
- Spécifiez le «viewpointName» et le «representationName» qui permettront d'activer le configurateur pour une représentation donnée
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MyDialectEditorConfigurer`"
- Implémentez votre configurateur, en écrivant le code de la méthode `postOpen(DialectEditor dialectEditor)`

8.3.9. Modifier le comportement de l'action de tri de l'explorateur de modèle

Le comparateur utilisé par l'action de tri de l'explorateur de modèle peut être redéfini.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.explorerSorterExtension`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MyComparator`"
- Implémentez votre filtre

Le comparateur suivant trie les éléments selon leur nom par ordre alphabétique :

```
public class MyComparator extends ViewerComparator {

   public int compare(Viewer viewer, Object e1, Object e2) {
      if (e1 instanceof Nameable && e2 instanceof Nameable) {
         return ((Nameable)e1).getName().compareTo(((Nameable)e2).getName());
      }
      return super.compare(viewer, e1, e2);
   }
}
```

8.3.10. Ajouter un fournisseur de contenu spécifique à la vue Navigateur

Le fournisseur de contenu de la vue navigateur, qui détermine quels éléments sont associés à la sélection courante, peut être redéfini.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.navigatorContentProviderExtension`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.DemoNavigatorContentProviderExtension`"
- Implémentez votre filtre

L'exemple ci-dessous montre un exemple d'implémentation de l'interface `INavigatorContentProviderExtension`.

```
public class DemoNavigatorContentProviderExtension implements
 INavigatorContentProviderExtension {

   public boolean apply(CDOObject object) {
      return MyMetaModelPackage.eNS_URI.equals(object.eClass().getEPackage().getNsURI());
   }

   //The parameter "reverse" is true if outgoing children are searched, false if incoming
 children are searched
   public Collection<Object> getChildren(CDOObject parentElement, boolean reverse) {
      //Return the child elements of the given parent element.
   }

}
```

8.3.11. Personnaliser la documentation d'un métamodèle

Dans le prisme on peut modifier, pour chaque type d'éléments, sa description courte et sa description étendue. Ces descriptions sont affichées dans la vue «Astuces» du modeleur, ainsi que dans la palette Sirius de création d'éléments.

Lors de la définition d'un métamodèle spécifique, on peut initialiser les documentations des éléments du métamodèle en utilisant le mécanisme suivant. Ce mécanisme supporte l'internationalisation, contrairement à la solution alternative consistant à ajouter les descriptions une par une dans le prisme.

Pour ajouter la documentation spécifique à un métamodèle, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.metamodelDocumentation`" avant de cliquer sur OK
- Dans la zone de droite choisissez :
  - l'uri du métamodèle concerné (par exemple `http://www.obeo.fr/smartea/archimate/3.2`)
  - dans `htmlDescriptions`, un chemin relatif au plug-in vers un répertoire contenant les descriptions étendues, par exemple `doc/`. Ce répertoire devra contenir un fichier .html par EClass du métamodèle, avec un suffixe indiquant la locale si besoin. Par exemple : `ApplicationComponent.html`, `ApplicationComponent_fr.html` etc.
  - dans `shortDescriptions`, un chemin relatif au plug-in vers un fichier de properties contenant les descriptions courtes, par exemple `plugin.properties`. Ce fichier de properties devra contenir des couples clé/valeur correspondant au nom d'une EClass suffixé par `_ShortDescription` en clé, la valeur étant la description courte. Par exemple : `BusinessActor_ShortDescription=A business actor is a business entity that is capable of performing behavior.`

Pour permettre à Obeo SmartEA de retrouver la documentation associée à un outil Sirius lors du survol de la palette, il vous faudra implémenter un résolveur qui traduit un identifiant Sirius vers la EClass qui y correspond.

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.siriusToolResolver`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.ToolResolver`"
- Implémentez votre résolveur

Voici par exemple un extrait de la classe permettant de résoudre les outils de création pour les diagrammes Archimate dans Obeo SmartEA :

```
public class ToolResolver implements ISiriusToolResolver {

   private static final String TYPE_CREATION_TOOL_REGEX = "platform:/plugin/
fr.obeo.smartea.archimate.design/description/archimate.odesign#//@ownedViewpoints\
\[name='Archimate'\\]/@ownedRepresentations\\[name='archimate.ArchiMateView'\\]/@defaultLayer/
@toolSections.[0-9]+/@ownedTools\\[name='Create_([a-zA-Z]+)'\\]"; //$NON-NLS-1$

   @Override
   public Object resolveToolDocumentation(String siriusToolId) {
      Object resolved = null;
      if (siriusToolId.matches(TYPE_CREATION_TOOL_REGEX)) {
         Pattern pattern = Pattern.compile(TYPE_CREATION_TOOL_REGEX);
         Matcher matcher = pattern.matcher(siriusToolId);
         if (matcher.matches()) {
            resolved = ArchimatePackage.eINSTANCE.getEClassifier(matcher.group(1));
         }
      }
      return resolved;
   }

}
```

8.3.12. Filtrer/Internationaliser les références dérivées

Pour internationaliser le label d'une référence dérivée ou pour permettre ou non son activation sur un type donné dans l'éditeur de prisme, vous pouvez utiliser le point d'extension `fr.obeo.smartea.core.rcp.api.derivedReferenceFilter` :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.derivedReferenceFilter`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MyDerivedReferenceFilter`"
- Implémentez votre filtre :
  - la méthode `boolean apply(EClass eClass, String id)` déterminera si la référence dérivée avec l'identifiant donné est valide pour la EClass donnée.
  - la méthode `String getLabel(String id)` renverra le label internationalisé pour la référence ayant l'identifiant donné.

8.3.13. Ajouter un fournisseur de labels à la vue Analyse d'impacts

Le fournisseur de labels de la vue Analyse d'impacts peut être redéfini en vue de modifier le style des noeuds et connections affichés.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.impactsLabelProviderExtension`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.DemoImpactsLabelProviderExtension`"
- Implémentez votre extension

L'exemple ci-dessous montre un exemple d'implémentation de l'interface `IImpactsLabelProviderExtension`.

```
@Override
public boolean apply(Object object) {
   // return true if your label provider must apply
}

@Override
public void styleConnection(Object element, Object connection) {
   PolylineConnection zestFigure = (PolylineConnection)((GraphConnection)connection)
         .getConnectionFigure();
   // You can modify the connection style here
}

public void styleNode(Object element, Object node) {
   // You can modify the node style here
}
```

8.3.14. Ajouter un fournisseur de contenus à la vue Analyse d'impacts

Le fournisseur de contenus de la vue Analyse d'impacts peut être redéfini en vue de modifier les éléments et connections affichés.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.impactsContentProviderExtension`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.DemoImpactsContentProviderExtension`"
- Implémentez votre extension

L'exemple ci-dessous montre un exemple d'implémentation de l'interface `IImpactsContentProviderExtension` qui permet de renvoyer les connections spécifiques aux éléments Archimate.

```
public boolean apply(Object object) {
   return object instanceof ArchimateComponent;
}

public Collection<?> getElements(Object input, int direction) {
   Collection<Object> res = new ArrayList<>();
   ArchimateComponent element = (ArchimateComponent)input;
   // Check if it was deleted
   if (element.eContainer() != null) {
      if (element instanceof Relationship) {
         res.add(element);
      } else {
         switch (direction) {
            case DIR_IN:
               res.addAll(element.getIncomingRelationships());
               break;
            case DIR_OUT:
```

```
                res.addAll(element.getOutgoingRelationships());
                break;
            case DIR_BOTH:
                res.addAll(element.getIncomingRelationships());
                res.addAll(element.getOutgoingRelationships());
                break;
            default:
                break;
        }
      }
   }
   return res;
}
```

8.3.15. Ajouter un type de connection à la vue Analyse d'impacts

Par défaut la vue Analyse d'impacts considère les EReferences EMF comme des connections entre éléments. Il est possible d'afficher d'autres connections ayant du sens dans un métamodèle spécifique (classes-associations), voire des connections calculées automatiquement.

Pour définir un nouveau type de connection vous pouvez utiliser le point d'extension `fr.obeo.smartea.core.rcp.api.ui.impactsConnectionExtension` :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.impactsConnectionExtension`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MyConnectionExtension`"
- Implémentez votre extension. Voici par exemple l'implémentation des connections Archimate :

```
public boolean apply(Object element) {
   return element instanceof Relationship;
}

public Object getSource(Object rel) {
   return ((Relationship)rel).getSourceElement();
}

public Object getDestination(Object rel) {
   return ((Relationship)rel).getTargetElement();
}
```

8.3.16. Personnaliser les liens des représentations publiées

Lorsqu'une représentation est publiée, les éléments de l'image obtenue sont cliquables dans la vue Web. Des métadonnées définissant un mapping entre les coordonnées d'un élément et l'identifiant logique d'un élément permettent cette fonctionnalité. Par défaut, cliquer sur un élément d'une représentation publiée mène à la page de détails de cet élément.

Il est possible de définir une redirection afin de modifier ce comportement. On peut ainsi faire en sorte que cliquer sur un élément :

- mène à la page de détails d'un autre objet calculé à partir de l'élément sélectionné.
- mène à la page de détails du premier artefact d'un type donné dont l'élément sélectionné est le contexte.
- cumuler ces deux aspects: cliquer sur un élément mènera sur le premier artefact défini sur un élément calculé à partir du premier.

Pour définir ce changement de comportement, vous pouvez utiliser le point d'extension `fr.obeo.smartea.core.rcp.api.ui.publicationRedirection` :

- Ouvrez l'éditeur du fichier "`META-INF/MANIFEST.MF`" de votre plug-in d'extension
- Sélectionnez l'onglet "`Extensions`"
- Cliquez sur "`Add`" puis sélectionnez "`fr.obeo.smartea.core.rcp.api.ui.publicationRedirection`" avant de cliquer sur OK
- Dans la zone de droite choisissez le nom de la classe d'implémentation, exemple "`com.mycompany.myextensions.rcp.ui.MyPublicationRedirectionResolver`"
- Implémentez votre extension. Voici par exemple l'implémentation des redirections BPMN :

```
   private static final String PROCESS_DIAGRAM_TYPE_URI = "sirius://bpmn2.Process/Process/
ProcessDiagram";
   private static final String SUB_PROCESS_DIAGRAM_TYPE_URI = "sirius://bpmn2.SubProcess/
Process/SubProcessDiagram";

   public boolean appliesOn(EObject element) {
      return element instanceof Process || element instanceof CallActivity || element
 instanceof SubProcess;
   }

   public String getRedirectId(EObject element) {
      if (element instanceof CallActivity) {
         return getRedirectId(((CallActivity)element).getCalledElementRef());
      }
      return EcoreUtil.getID(element);
   }

   public String getRedirectArtifactTypeURI(EObject element) {
      String res = null;
      switch (element.eClass().getClassifierID()) {
         case Bpmn2Package.CALL_ACTIVITY:
            res = PROCESS_DIAGRAM_TYPE_URI;
            break;
         case Bpmn2Package.PROCESS:
            res = PROCESS_DIAGRAM_TYPE_URI;
            break;
         case Bpmn2Package.SUB_PROCESS:
            res = SUB_PROCESS_DIAGRAM_TYPE_URI;
            break;
         default:
            break;
      }
      return res;
   }
```

Cette extension fait en sorte que :

- cliquer sur un Process mène au premier ProcessDiagram (si présent) défini sur cet élément, sinon mène à la page de détails du Process
- cliquer sur un SubProcess mène au premier SubProcessDiagram (si présent) défini sur cet élément, sinon mène à la page de détails du SubProcess
- cliquer sur un CallActivity mène au premier ProcessDiagram du Process appelé par le CallActivity si le diagramme est présent, sinon mène à la page de détails du Process appelé par le CallActivity

8.3.17. Ajouter un module fonctionnel dans l'éditeur de prisme

Il est possible d'ajouter un module fonctionnel dans l'éditeur de prismes. Les modules fonctionnels apparaissent dans le menu «Module fonctionnel» de l'éditeur de prismes. Un module fonctionnel permet d'ajouter un ensemble d'éléments prédéfinis dans l'éditeur de prismes. L'entrée de menu et les éléments à ajouter dans l'éditeur de prismes sont à définir via le point d'extension `fr.obeo.smartea.core.rcp.api.ui.functionalModuleProvider`.

Pour ce faire, il vous est possible d'implémenter le point d'extension prévu à cet effet :

- Ouvrez l'éditeur du fichier `META-INF/MANIFEST.MF` de votre plug-in d'extension

- Sélectionnez l'onglet "Extensions"
- Cliquez sur «Add» puis sélectionnez `fr.obeo.smartea.core.rcp.api.ui.functionalModuleProvider` avant de cliquer sur Finish
- Dans la zone de droite, définissez :
  - id: l'identifiant unique de l'entrée de menu du module fonctionnel.
  - name: le nom du module fonctionnel qui sera affiché dans l'entrée de menu.
  - class: la classe qui fournit les éléments à ajouter à l'éditeur de prismes.
- Implémentez la classes qui fournit les éléments à ajouter à l'éditeur de prismes (Reference Model uniquement actuellement). Cette classe doit définir la méthode `DocumentRoot getProvidedPrismModel()` de l'interface `fr.obeo.smartea.core.rcp.api.ui.IFunctionalModuleProvider`. Voici un exemple :

```
public class EntityReferenceModelProvider implements IFunctionalModuleProvider {

   @Override
   public DocumentRoot getProvidedPrismModel() {
      URI uri = URI.createURI("my_custom_prism.prism");
      EcoreResourceFactoryImpl factory = new EcoreResourceFactoryImpl();
      Resource res = factory.createResource(uri);
      try {
         res.load(null);
         List<EObject> content = res.getContents();
         DocumentRoot prism = (DocumentRoot)content.get(0);
         return prism;
      } catch (IOException e) {
         e.printStackTrace();
         return null;
      }
   }

}
```

8.4. Syntaxe textile

Cette section décrit la syntaxe textile supportée pour la définition de manuels d'aide spécifiques.

Les références ci-dessous ont été le point d'entrée pour la définition de la syntaxe:
- Wikitext user guide
- Wikipedia
- TxStyle

La conversion du fichier textile passe par trois étapes:
- conversion de textile vers un fichier dita basée sur wikitext
- conversion vers un modèle dita
- génération des fichiers à partir du modèle.

Ces trois étapes ont induit des limitations :
- syntaxe non prise en compte par wikitext
- perte d'information liée à la conversion vers dita

8.4.1. Les Block modifiers

Type	Syntaxe	Supporté	Non implémenté pour le moment
Block code	bc., bc.. p.	Oui	

Type	Syntaxe	Supporté	Non implémenté pour le moment
No textile formating	notextile.	Non	Non supporté (limitation liée à la conversion Dita)
Heading (1->6)	hn.	Oui	
Paragraph, optional	p.	Oui	indentation de paragraphe, alignement de paragraphe, retour, non supportés (limitation liée à la conversion Dita)
Block quote	bq., bq.. p.	Oui	
Textile comment	###.	Non	Non supporté (limitation wikitext)
CSS Styler	{}	Non	Non supporté (limitation liée à la conversion Dita)
Pre-formatted	pre., pre.. p.	Oui	
HTML		Non	L'utilisation d'HTML n'est pas supporté par le générateur

8.4.2. Listes et Notes

Type	Syntaxe	Supporté	Non implémenté pour le moment
Definition list	- … := …	Non	Non supporté (limitation wikitext)
Auto numbered note		Non	Non supporté (limitation wikitext)
Numeric list	#, ##, …	Oui	Non supporté (limitation wikitext) : #n Numeric list starting from a defined number, #_ Continue a numeric list
Footnote	@ @fnn.	Non	La définition associée à la note doit être dans le même paragraphe que l'appel à la note. Le document étant découpé en pages, le même numéro peut être utilisé une fois par chapitre h1.

Type	Syntaxe	Supporté	Non implémenté pour le moment
Bulleted list	*	Oui	

8.4.3. Liens

Type	Syntaxe	Supporté	Non implémenté pour le moment
Image link	!image!:url	Oui	
E-mail link	"title":mailto:someone@example.org	Oui	
Formatted title link	"*bold title example*":url	Oui	
External Links	"title":url	Oui	
Internal link to an anchor	"title":#Anchor	Oui	
Local links	"title":/path	Oui	
URL title link	"$":url	Oui	
Using alias as link	"title":alias [alias]url	Non	Non supporté (limitation wikitext)

8.4.4. Images

Type	Syntaxe	Supporté	Non implémenté pour le moment
CSS style		Non	Non (limitation liée à la conversion Dita)
Image aligned right	!>/path/file.png!	Oui	
Scaling	!{width:80%}images/foo.png!	Non	Non (limitation liée à la conversion Dita)
Image from a remote service	!http://www.codecogs.com/eq.latex?\int_{-\infty}^{\infty}e^{-x^{2}}dx=\sqrt{\pi}!	Oui	exemple d'image de formule latex fournit par un générateur
Relative path	!/path/file.png!	Oui	
CSS style	!{border:5px solid red;}images/foo.png!	Non	Non (limitation liée à la conversion Dita)
Sizizing	!{width=32px;height=64px}images/foo.png!	Non	Non (limitation liée à la conversion Dita)
Image centered	!=/path/file.png!	Oui	
image with title	!/path/file.png (title)!	Non	Non supporté (limitation wikitext)
image with title	!/path/file.png (title)!	Non	
Other domain	!http://example/file.png!	Oui	

8.4.5. Tables

Type	Syntaxe	Supporté	Informations complémentaires
Cell horizontal left aligned	<.	Oui	
Table attributs	table(tableclass).	Non	
Standard table		Oui	
html5 style		Non	
Cell vertical middle aligned		Oui	
Cell vertical top aligned	^.	Non	Non supporté (limitation liée à la conversion Dita)
Row and colums Spanning	\n. text /n. text	Non	
Cell horizontal right aligned	>.	Non	Non supporté (limitation liée à la conversion Dita)
Row attributes	.	Non	
Cell vertical bottom aligned		Oui	
Cell horizontal center aligned		Oui	
Cell CSS styled	{}	Non	Non supporté (limitation liée à la conversion Dita)
Header	_.	Non	Non supporté (limitation liée à la conversion Dita), mais du texte formaté peut être utilisé **bold example**

note : Afin d'apparaitre dans le document final, une case vide doit contenir un caractére blanc (limitation liée à la conversion dita).

8.4.6. Les phrases modifiers

Type	Syntaxe	Supporté	Informations complémentaires
Superscript	^..^	Oui	
Subscript	.~	Oui	
Citation	?..?	Oui	
Deleted text	.-	Oui	
Italic	__..__	Oui	

Type	Syntaxe	Supporté	Informations complémentaires
Span	%{CSS Style}..%	Oui	Span est supporté mais le typage CSS n'est pas supporté (limitation liée à la conversion Dita)
Bold	**..**	Oui	
Code	..	Oui	
Emphasis	_.._	Oui	
Strong	*..*	Oui	converti en bold
Inserted text	_.._	Oui	

8.4.7. Alignement et indentation

Type	Syntaxe	Supporté	Informations complémentaires
Right aligned paragraph	p>.	Non	Non supporté (limitation liée à la conversion Dita)
Paragraph indentation	p)., p))., …	Non	Non supporté (limitation liée à la conversion Dita)
Left aligned paragraph	p<.	Non	Non supporté (limitation liée à la conversion Dita)
Justified paragraph	p<>.	Non	Non supporté (limitation liée à la conversion Dita)
Paragraph indentation	hn).	Non	Non supporté (limitation liée à la conversion Dita)
Centered paragraph	p	Non	Non supporté (limitation liée à la conversion Dita)

8.4.8. Auto-conversion de caractères

Type	Syntaxe	Supporté	Informations complémentaires
“ignored textile” conversion	=.==	Non	Non supporté (limitation liée à la conversion Dita)
Degree ° symbol	° -> °	Oui	
Double hyphen becomes a long dash	— -> —	Oui	

Type	Syntaxe	Supporté	Informations complémentaires
Three dots become an ellipsis character	… -> …	Oui	
Copyright ©	(C) -> ©	Oui	
x between numbers becomes a dimension sign	1x3 -> 1×3	Oui	
One half ½ symbol	½ -> ½	Oui	
Ampersand escaped	& ->&amp	Oui	
Trademark™	(tm) -> ™	Oui	
Registered®	(R) -> ®	Oui	
Straight apostrophe convert to curly	" ->‘ ’	Oui	
Three quarters ¾ symbol	¾ -> ¾	Oui	
One quarter ¼ symbol	¼ -> ¼	Oui	
Angle brackets	<> -> &lg &gt	Oui	
Plus/minus ± symbol	± -> &#177	Oui	
A hyphen between whitespaces becomes a unicode dash	text–text -> –	Oui	