Opcoach

EXPERTISE ECLIPSE, JAVA, OPEN SOURCE

OSGi, RCP, Eclipse Modeling

23 juin 2011

Eclipse Party Toulouse

OPC 11 ECP PRE RCP 01 A

Présentation

OPCoach

- SARL créée en juin 2009 *http://www.opcoach.com*
- Membre de la fondation Eclipse (Solution Member)
- Olivier Prouvost *olivier.prouvost@opcoach.com*[1]
  - Co fondateur Anyware Technologies
  - Expert XML, Java / Eclipse
- Objet
  - Formation, Expertise et Audit Eclipse

Références

- **Grands comptes**

1 - mailto:olivier.prouvost@opcoach.com

...

- **SSII** : Atos, Artal, Umanis, Aptus, Akka, Euriware, Zenika, Assystem, ...
- **PME** : Esterel, GVS, Silicom, Prometil, ...

Open Services Gateway initiative (OSGi)

II

Pourquoi OSGi ?

Java semble modulaire

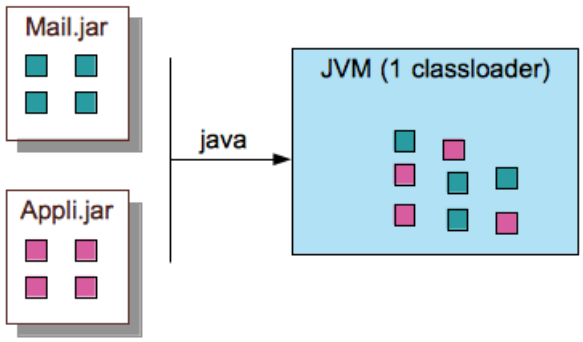
- packages
- classes
- méthodes
- différents jar

Tout est bien séparé lors du développement.

« Mais que se passe t il quand on lance un programme java ? »

Lancement Java

La jvm se lance avec un seul class loader.
La modularité s'est dissoute.



Lancement Java classique

Les inconvénients du runtime java classique

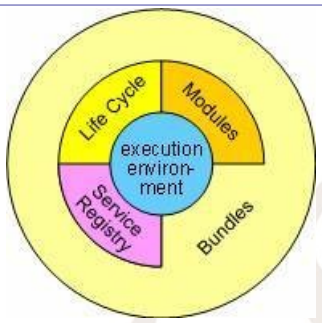
- Toute classe publique dans un jar est accessible par toute autre classe
- La notion de version dans le manifest est sans effet au runtime
- Il est impossible de changer un jar 'à chaud'
- Deux versions d'un même jar dans deux versions différentes ne peuvent

cohabiter

« OSGi va répondre à ces problèmes »

Présentation

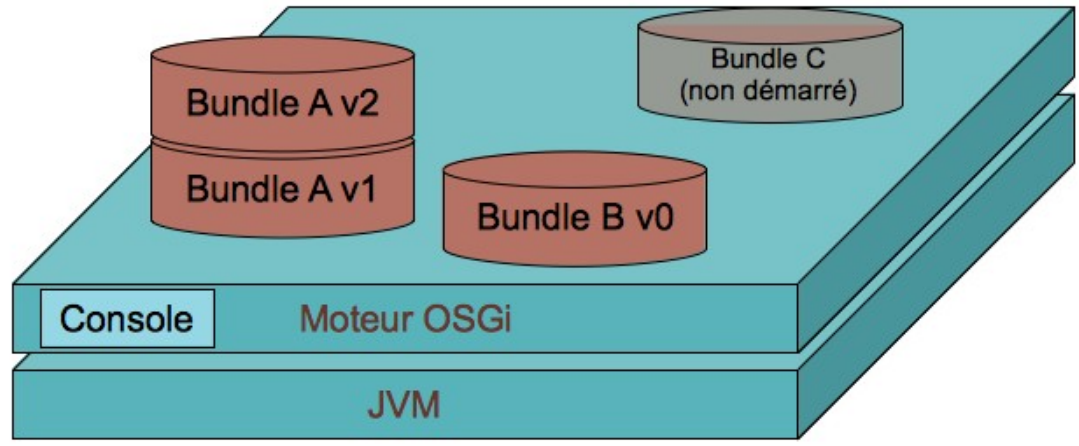
- OSGi définit la notion de 'Bundle'
- OSGi gère les versions des bundles
- OSGi gère l'activation des bundles
- OSGi permet de gérer des services (Service Registry)



OSGi Framework

- OSGI fournit un modèle de déploiement de modules logiciels java.
- OSGi est une spécification gérée par la communauté OSGi Alliance : *http://www.osgi.org*

Moteur OSGi



Moteur OSGi

Le bundle

Il est défini par un MANIFEST :

com.opcoach.message

```
Bundle-ManifestVersion: 2
Bundle-Name: Message Manager
Bundle-SymbolicName: com.opcoach.message
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: com.opcoach.message.MessageActivator
Bundle-Vendor: OPCoach
Bundle-RequiredExecutionEnvironment: J2SE-1.5
Import-Package: org.osgi.framework;version="1.3.0"
Bundle-ActivationPolicy: lazy
Export-Package: com.opcoach.message;uses:="org.osgi.framework"
```

Exemple de manifest

- Il possède un ID unique
- Il est définit pour une version
- Il peut publier ou consommer des services
- Il possède son propre class loader
- Il peut être défini pour une plateforme spécifique (windows, linux...)

Relations avec les autres bundles

- Le bundle exporte les packages qu'il veut rendre visible
- Le bundle peut dépendre d'autres bundles

com.opcoach.training.rental

```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: %pluginName
4  Bundle-SymbolicName: com.opcoach.training.rental;singleton:=true
5  Bundle-Version: 1.0.1
6  Bundle-ClassPath: .
7  Bundle-Vendor: OPCoach
8  Bundle-Localization: plugin
9  Bundle-RequiredExecutionEnvironment: J2SE-1.5
10 Export-Package: com.opcoach.training.rental,
11  com.opcoach.training.rental.impl,
12  com.opcoach.training.rental.util
13 Require-Bundle: org.eclipse.core.runtime,
14  org.eclipse.emf.ecore;visibility:=reexport,
15  org.eclipse.swt;bundle-version="3.5.1";visibility:=reexport
16 Bundle-ActivationPolicy: lazy
```

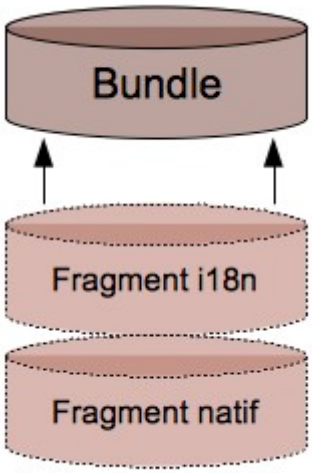
visibilité donnée aux autres bundles

Autres bundles utilisés

Relations d'un bundle

Bundle Fragment

- Le fragment est un complément d'un bundle (host bundle)
- Le fragment est utilisé pour :
  - gérer du code spécifique selon une plate forme
  - compléter un bundle avec des fichiers (i18n, ...)
  - écrire des tests (possède la visibilité sur le bundle)



Fragment

L'activator

- L'activator est le code d'entrée du Bundle.
- Il implémente l'interface BundleActivator :

```
public interface BundleActivator {
    /**
     * Called when this bundle is started so the Framework can perform the
     * bundle-specific activities necessary to start this bundle. This method
     * can be used to register services or to allocate any resources that this
     * bundle needs.
     */
    public void start(BundleContext context) throws Exception;

    /**
     * Called when this bundle is stopped so the Framework can perform the
     * bundle-specific activities necessary to stop the bundle. In general, this
     * method should undo the work that the <code>BundleActivator.start</code>
     * method started. There should be no active threads that were started by
     * this bundle when this bundle returns. A stopped bundle must not call any
     * Framework objects.
     */
    public void stop(BundleContext context) throws Exception;
}
```

Bundle Activator

- L'activator peut contenir le code d'initialisation
- L'activator publie les services

Les services

- Un bundle peut publier des services
- Un service est matérialisé par une interface qui répond à un besoin

MessageSender.java

```java
package com.opcoach.message;

import java.util.Map;

/**
 * This interface defines the message sending service
 *
 * @author olivier
 *
 */
public interface MessageSender
{

    /** send the message to the recipients defined in message.
     * @param message the message to be sent (channel independent)
     * @param params an optional map of parameters used to send the message
     */
    public void send(Message message, Map<String,Object> params) throws Exception;

}
```

Service Interface

Enregistrement d'un service

Le service s'enregistre sur le BundleContext lors du start

MailMessageActivator.java

```java
package com.opcoach.message.mail;

import org.osgi.framework.BundleActivator;

public class MailMessageActivator implements BundleActivator
{

    public void start(BundleContext context) throws Exception
    {
        context.registerService(MessageSender.class.getName(),
                new MailMessageSender(), null);
    }

    public void stop(BundleContext context) throws Exception
    {
    }

}
```

Publication du service

Utilisation d'un service

Le service s'utilise au moment voulu en interrogeant le bundleContext

MessageTestActivator.java

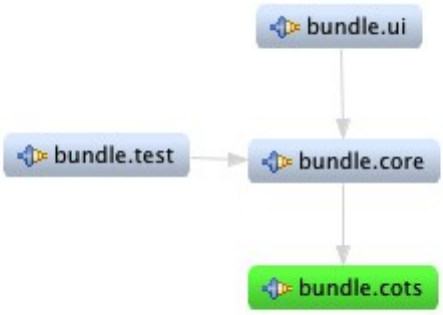
```
package com.opcoach.message.test;

import org.osgi.framework.BundleActivator;

public class MessageTestActivator implements BundleActivator
{

    public void start(BundleContext context) throws Exception
    {
        // Get a service reference
        ServiceReference ref = context.getServiceReference(MessageSender.class.getName());
        // Get the service
        MessageSender sender = (MessageSender) context.getService(ref);
        Message m = new MessageImpl("Important Message");
        m.setText("Un nouveau message");
        // Use it !
        sender.send(m, null);
    }

    public void stop(BundleContext context) throws Exception
    {
    }

}
```

Utilisation service

Utilisation d'OSGi

Il faut raisonner en 'bundles' et fragments :

- un bundle de définition du 'modèle'
- des bundles clients qui utilisent le modèle
- des bundles partagés entre projets (authentification...)
- des bundles 'wrappers' de jar qui reexportent les packages
- des fragments pour le code natif ou l'i18n
- des fragments de tests



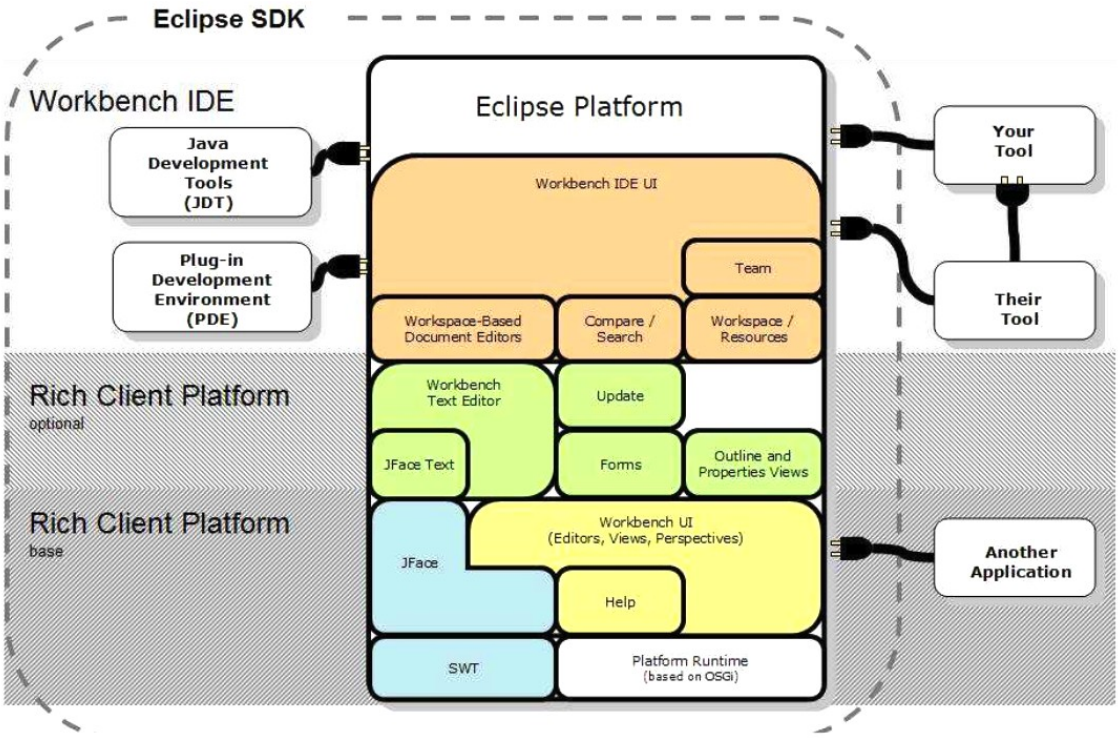
Dépendances de bundles

Architecture Eclipse

III

Principe général

- L'architecture d'Eclipse est basée sur OSGi et définit la notion de plugin.
- Un plugin est un composant logiciel élémentaire
- Les plugins sont gérés par un moteur (la Platform) qui les gère selon des règles d'optimisation.



Détail

A. Plugin

Plugin

Le plugin est une entité logicielle normalisée qui respecte la norme OSGi. Il est défini par deux fichiers :

- **META-INF/MANIFEST.mf** : fichier descriptif pour OSGi
- **plugin.xml** (optionnel) : fichier descriptif pour les extensions Eclipse

Pour simplifier la construction, un fichier est géré par Eclipse :

- **build.properties** : indique repertoire src, bin...

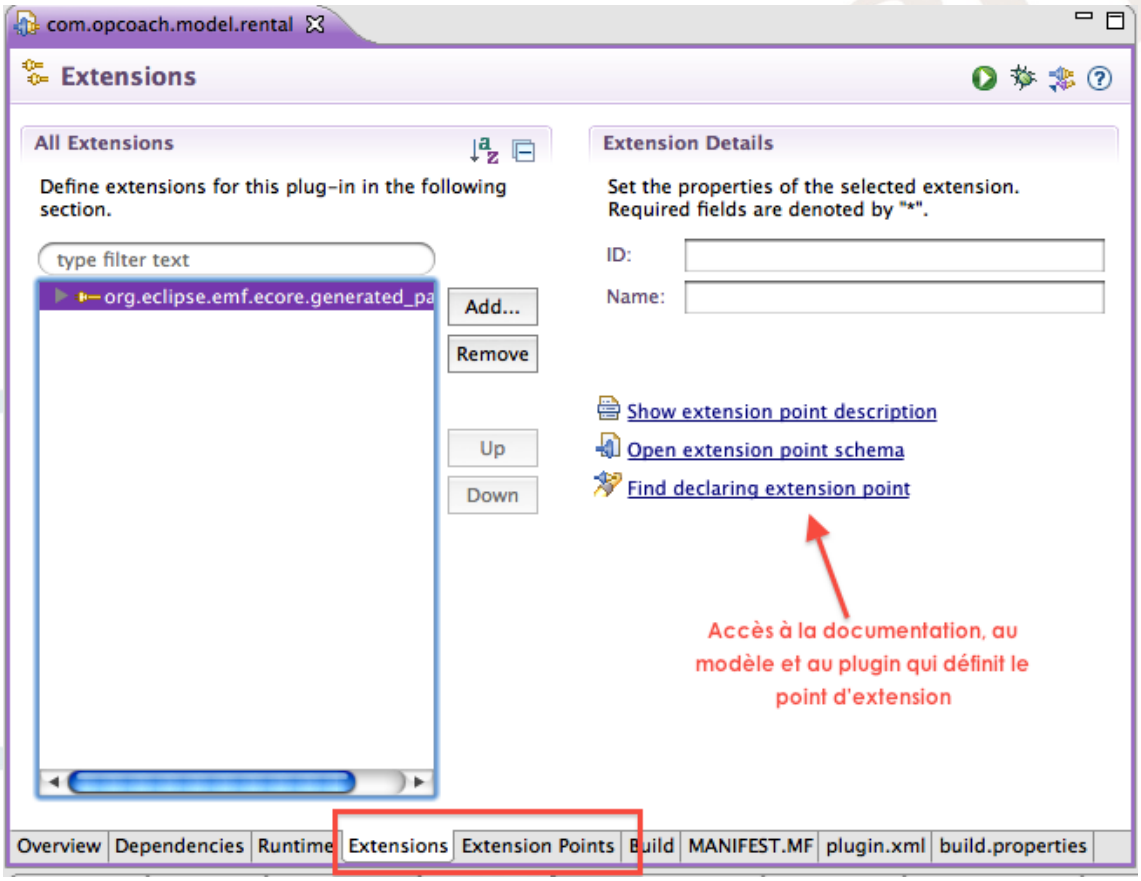
Un projet plugin est par défaut un projet java. Il contient donc automatiquement :

- Un fichier **.project**
- Un fichier **.classpath**

Plugin extension de Bundle

Le **plugin.xml** définit 2 notions supplémentaires : extensions et point d'extension :

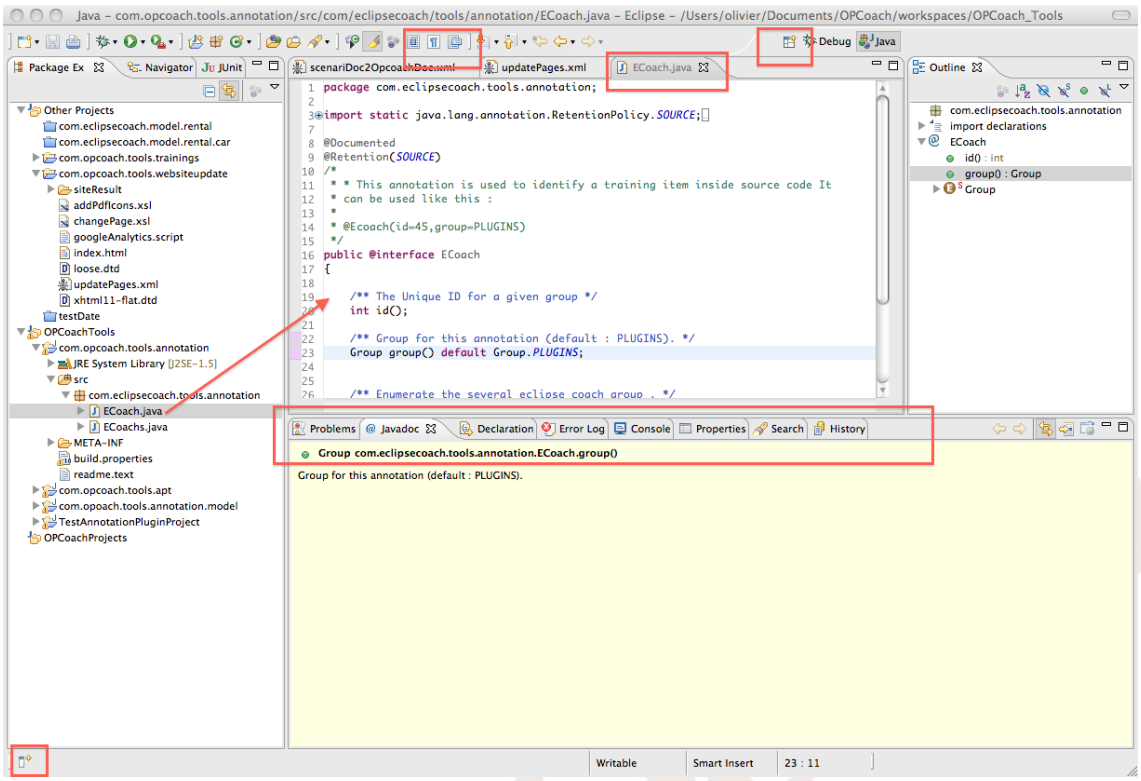
- **Point d'extension** : définit un modèle d'un concept que le plugin sait gérer (un menu, une vue, un driver...).
- **Extension** : définit une instance du modèle décrit par le point d'extension (la vue Navigator, le driver Z, ...)



Gestion des extensions (éditeur de plugin.xml)

Extensions dans le workbench

On retrouve les extensions dans les différentes parties de l'IDE



Impact des extensions dans les workbench

Autres points d'extension

La notion de point d'extension n'est pas réservée à l'IHM pure

Toute notion 'extensible' peut être modélisée sous cette forme

On trouve des points d'extensions sur de nombreux thèmes :

- **runtime** : adapters, produits, applications
- **workspace** : builders, natures ...
- **text** : markers, editor templates ...
- **team** : repository, synchronization ...
- **debug** : launchers, breakpoints, ...
- **help** : contextes, pages statiques

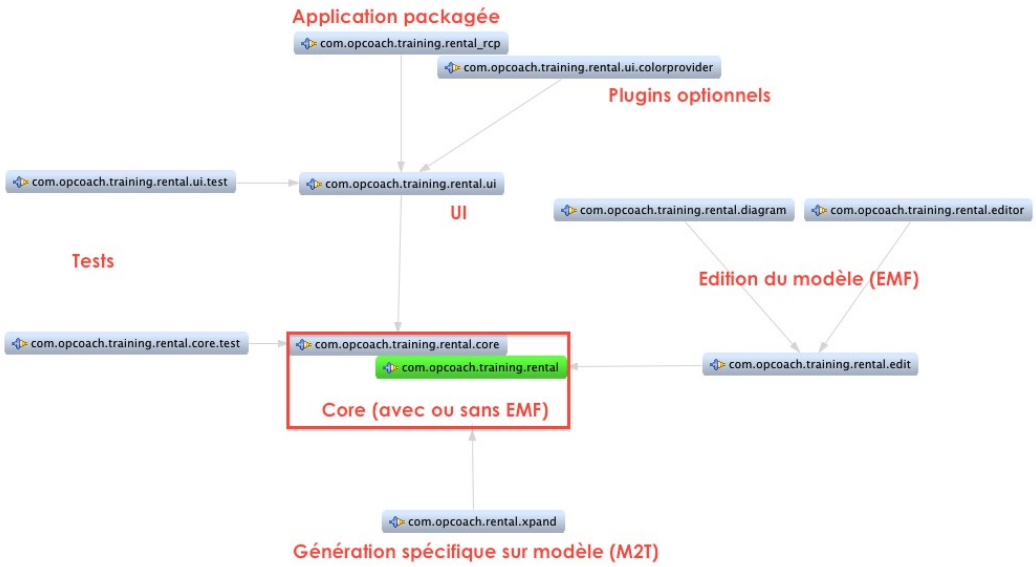
> Plusieurs centaines de points d'extensions sont disponibles dans l'architecture RCP

Plugin Fragment

- Le fragment est un complément d'un plugin (host plugin)
- Le fragment est utilisé pour :
  - gérer du code spécifique selon une plate forme
  - compléter un plugin avec des fichiers de properties (i18n)
  - écrire les tests d'un plugin
- Le fragment a toute la visibilité sur le plugin

Architecture d'une application

Comme pour les bundles OSGi on modularise une application Eclipse en différents plugins :



Architecture d'exemple

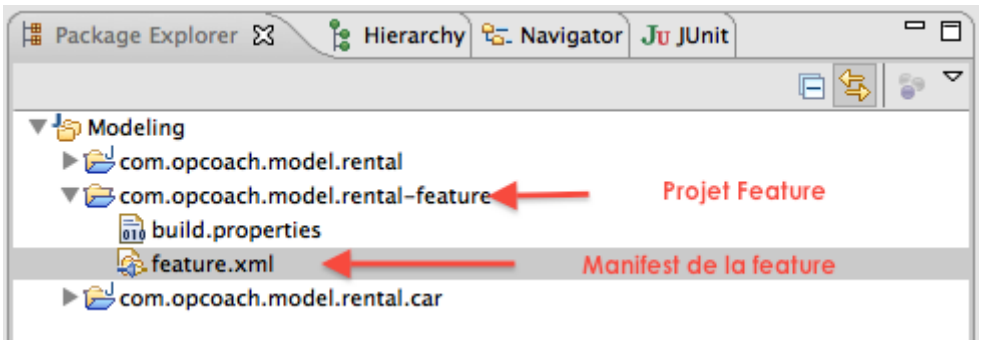
B. Feature

Feature

- La feature est un ensemble de bundles, de plugins, de fragments ou de features
- Elle possède également : une licence, et la propriété d'être installable à distance
- Elle se créé dans un projet de nature 'feature', où l'on regroupe les plugins
- Elle gère les compatiblités de version des plugins qu'elle contient ou dont elle dépend.
- Elle peut être définie pour une plateforme en particulier

Vue logique du projet Feature

La feature est représentée dans une vue logique du package explorer

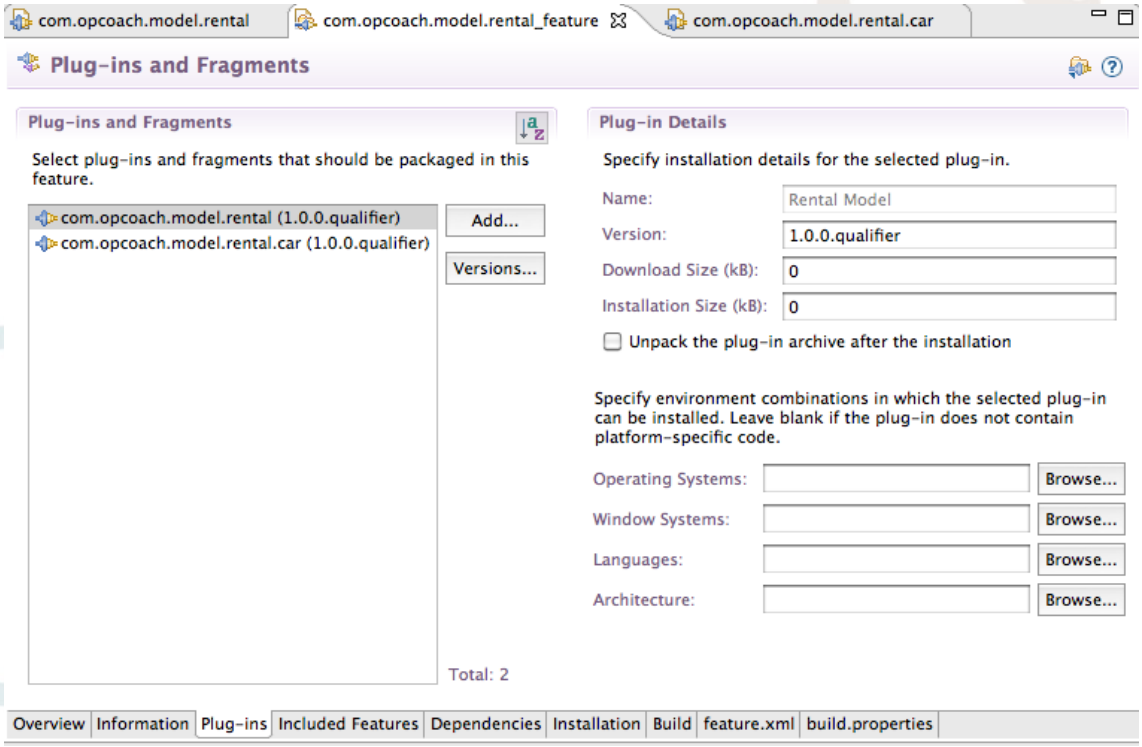


Vue logique de la feature

Edition de la feature

L'éditeur de feature permet d'indiquer :

- les plugins contenus dans la feature
- les données complémentaires : ID, description, texte de licence, etc...



Editeur de feature

C. Distribution de l'application

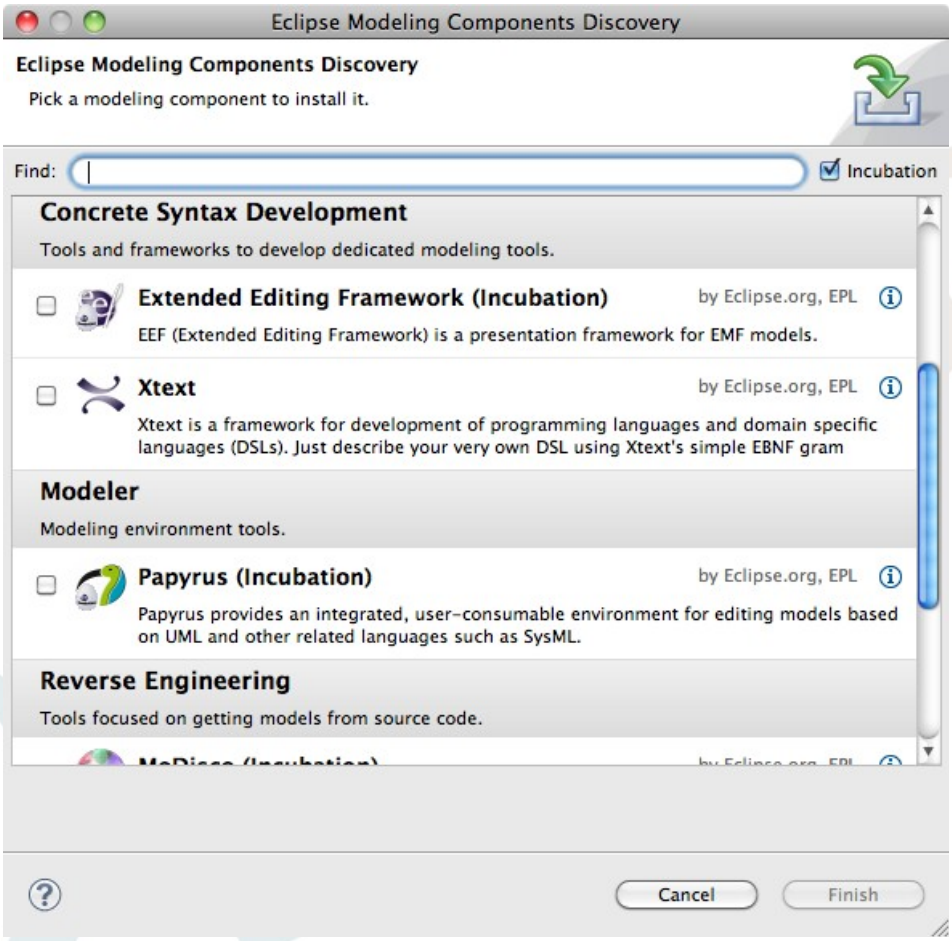
Repository p2

- p2 est le nouveau gestionnaire de mise à jour
- p2 gère des 'installable units' (IU) ayant des caractéristiques d'installation
- p2 est fourni avec un installeur indépendant (p2 agent)

- les sites p2 sont générables via Eclipse, Buckminster, Maven Tycho

Eclipse Market Place

- Pour simplifier l'installation de composants dans Eclipse
- Permet d'accéder en un clic au choix et à l'installation
- S'utilise dans une IHM conviviale :



Modeling MarketPlace

La target platform

C'est l'ensemble des plugins à utiliser pour exécuter une application

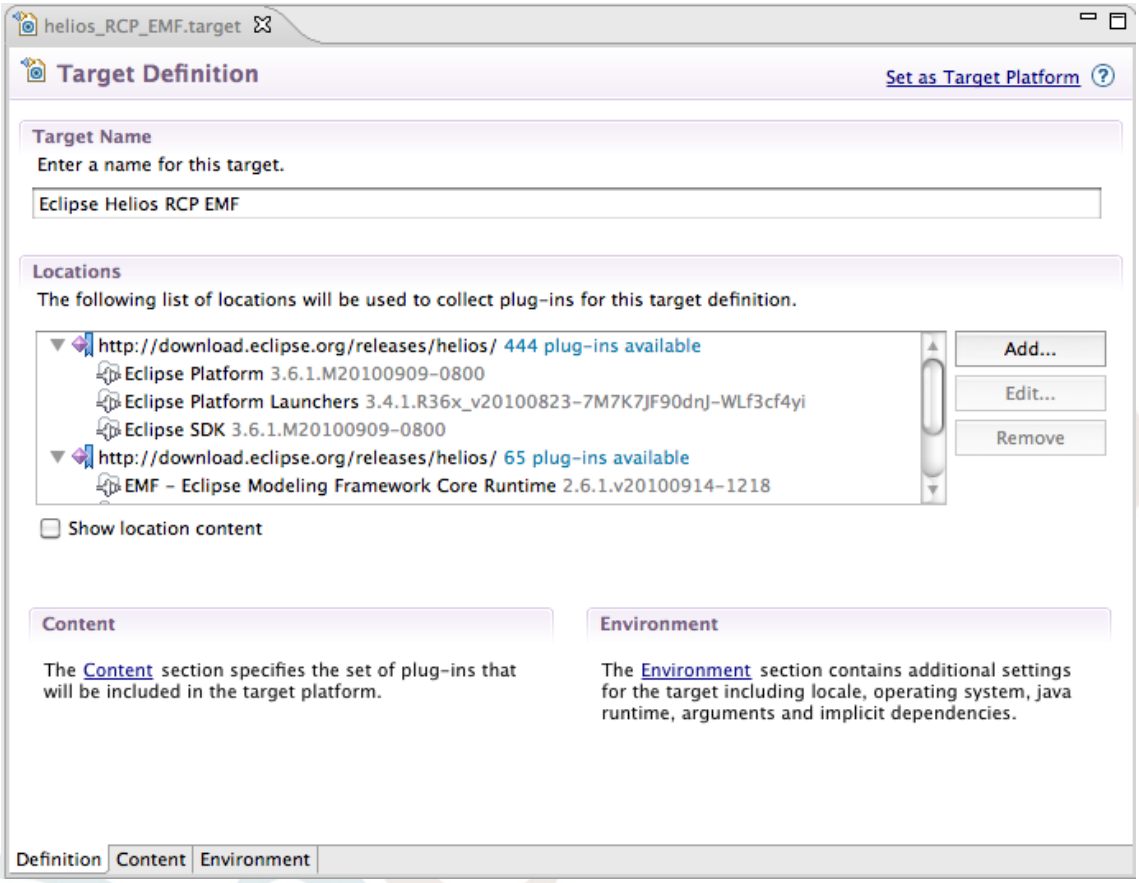
Elle peut être créée à partir :

- Directory : répertoire sur le disque
- Installation : installation d'un Eclipse
- Features : ensemble de features défini
- Software Site : à partir d'un repository p2

Définition d'une target platform

- Créer avec le New Wizard une target platform

- La remplir avec les plugins nécessaires



Target Platform Definition

Produit

- Un produit est un ensemble de features organisées sur une version RCP
- Le produit se livre pour une plateforme donnée
- On le relie à une feature pour le définir
- Il se livre sous la forme d'un exécutable RCP
- Il peut utiliser un repository p2 pour se mettre à jour (si cablé)
- Il peut définir des informations de branding (icone de lancement, splash screen..)
- Il se créé à l'aide de wizards dans eclipse (fichier .product)

D. Eclipse e4

Présentation

E4 est une évolution d'Eclipse pour simplifier le développement.

Au niveau IHM plusieurs possibilités :

- description d'une IHM SWT sans code
- utilisation des CSS pour gérer le rendu

- déportation d'une application SWT dans un browser
- extension et interrogation de l'IHM facilités

Au niveau architecture noyau :

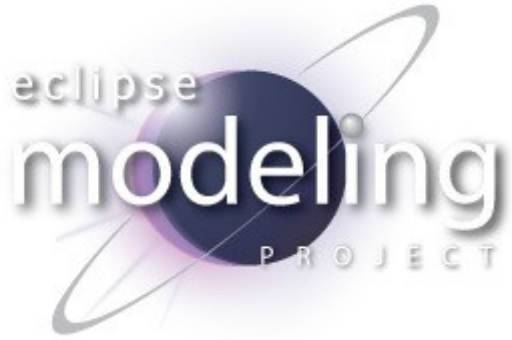
- utilisation intensive des modèles (ihm, appli, ...)
- utilisation intensive des services OSGi
- utilisation intensive de l'injection

> Le white paper e4 en décrit les principales fonctionnalités : *http://www.eclipse.org/e4/resources/e4-whitepaper.php*

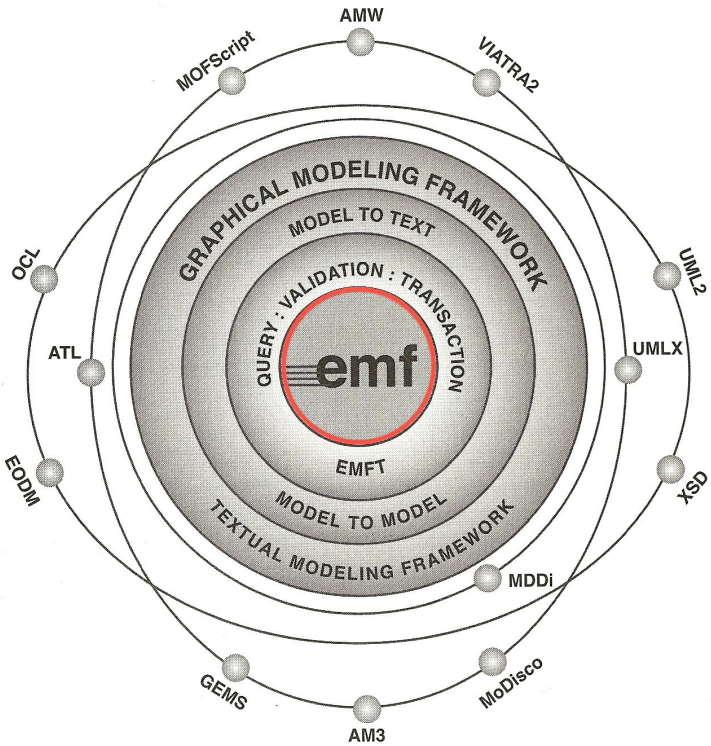
Le projet Modeling Eclipse

IV

Présentation

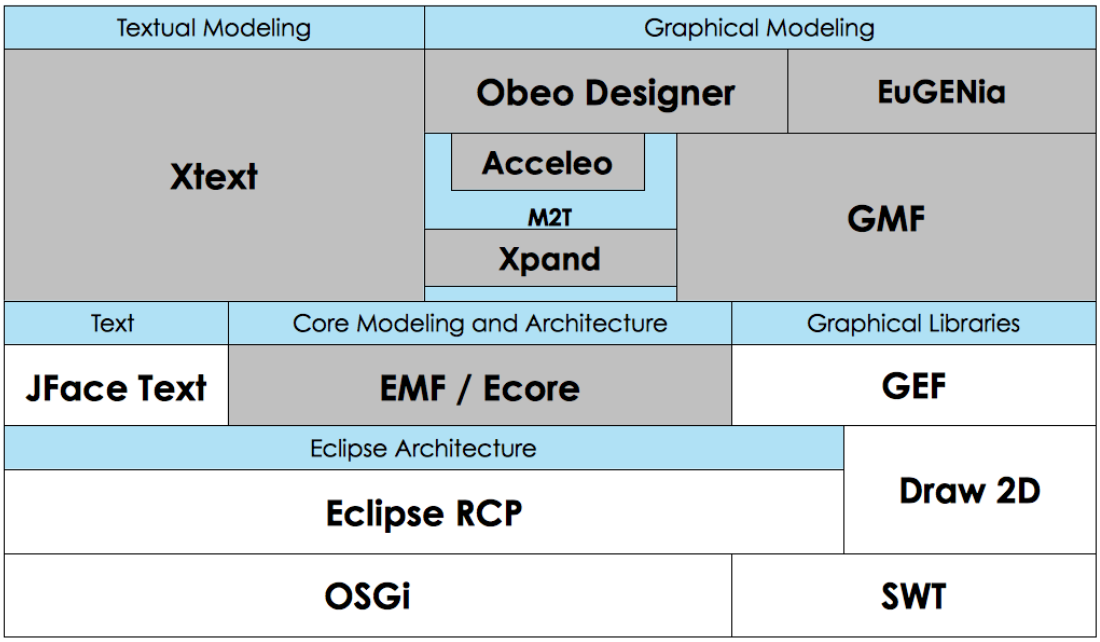


Big Picture



EMF dans le projet Modeling

Le projet Modeling dans Eclipse



Modeling

A. Eclipse Modeling Framework (EMF)

Principes

- EMF est le coeur du projet modeling.
- Outils de base permettant de faire de la modélisation
  - un langage de modélisation (Ecore)
  - des générateurs par défaut (bean, éditeur)

Objectifs

- Modéliser les données à manipuler d'une application
- Générer le code Java pour manipuler ce modèle
- Préparer la couche logicielle d'IHM pour visualiser/modifier les instances de ce modèle
- Proposer un éditeur par défaut du modèle
- Séparer le modèle du code java
- Faciliter les évolutions du modèle

Fonctionnalités de base

- Import de modèles depuis différents formats (Java, UML, XSD)
- Différentes éditeurs Ecore :
  - arborescent : par défaut

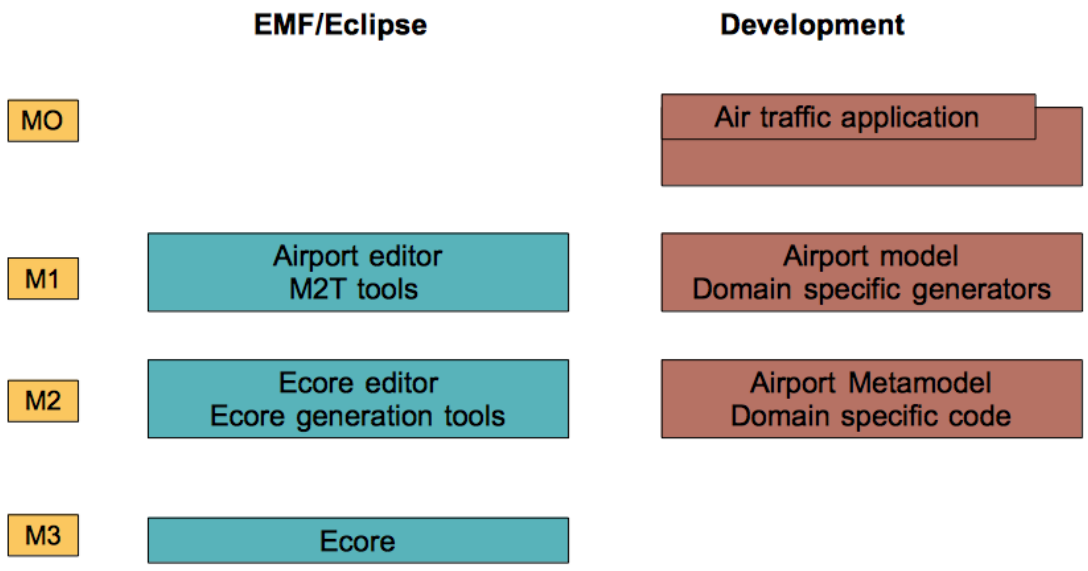
- graphique : ecoreTools
- textuel : emfatic

- Support des concepts RCP, du pattern Observer...
- API Réflective de manipulation du modèle
- Sérialisation XMI et XML automatique

Pourquoi utiliser EMF ?

- EMF fait le lien entre les mondes de la modélisation et de la programmation
  - Transforme les modèles en code Java
  - Permet d'exploiter le modèle dans une application réelle
  - Permet d'automatiser certaines taches de développement
- Open-source
- Technologie éprouvée (utilisée depuis 2002)
- Communauté très importante
- De très nombreux outils autour d'Ecore (validation, bdd, transformation).

Etapes de modélisation



EMF dans le processus de développement

Les projets à base d'EMF

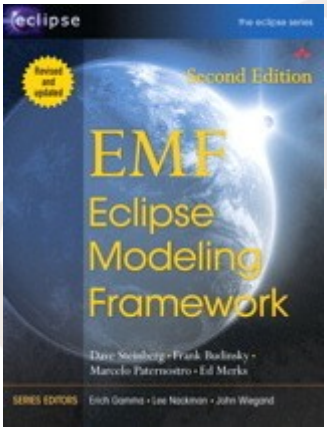
- Projets Eclipse :
  - Eclipse 4, WebTools, BIRT, DTP, Buckminster, ...
  - Un des piliers des projets Eclipse
- Produits commerciaux :
  - IBM, Oracle, SAP, Itemis, Obeo, ..
- Large communauté Open-Source :
  - Environ 125 000 téléchargements par mois
  - Cf : *http://www.eclipse.org/modeling/emf/downloads/stats.php*

Exemple d'utilisation d'EMF

- Création d'un modèle Ecore représentant votre domaine métier
  - Génération du code Java à partir du modèle
  - Génération sur mesure avec les outils M2T
- Création d'une instance du modèle en utilisant
  - l'éditeur généré
  - le code java généré
- Utilisation d'outils complémentaires intégrés dans l'application finale
  - GMF
  - Xtext
  - M2T (Xpand, Acceleo)
- Raffinage successif par évolution du modèle et regénération du code

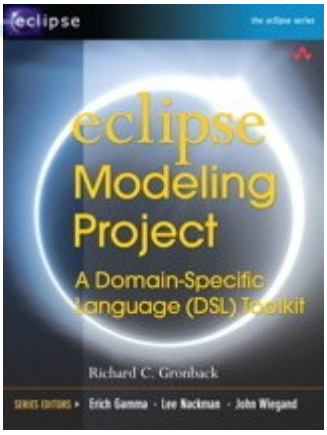
Livres de référence

Eclipse Modeling Framework



Livre EMF

Eclipse Modeling Project



Modeling Project

B. Edition Ecore

Création d'un modèle ECore

Le modèle ECore est la base de travail en EMF.
Il se créé soit par édition :

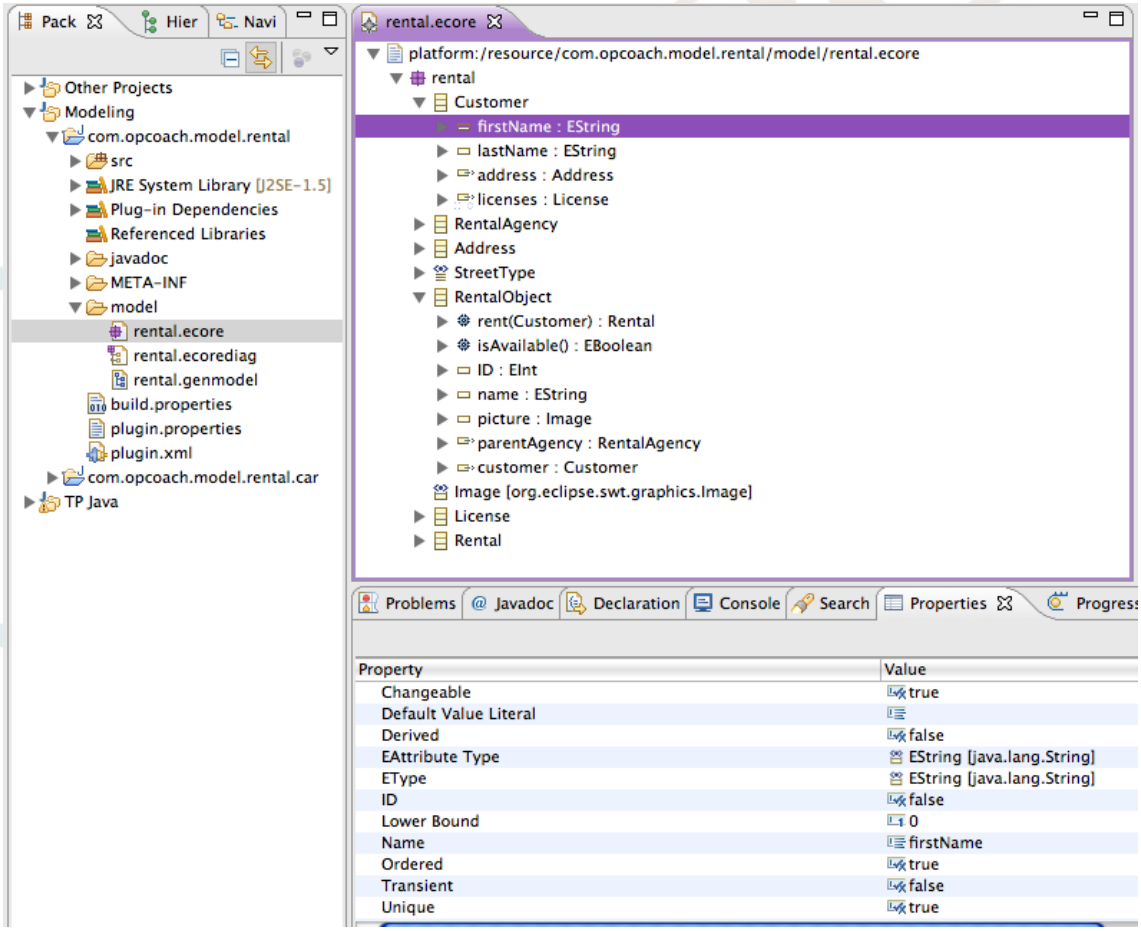
- arborescente
- graphique
- textuelle

Soit par import :

- code java annoté (@model)
- modèle UML
- XML Schema

Editeur arborescent

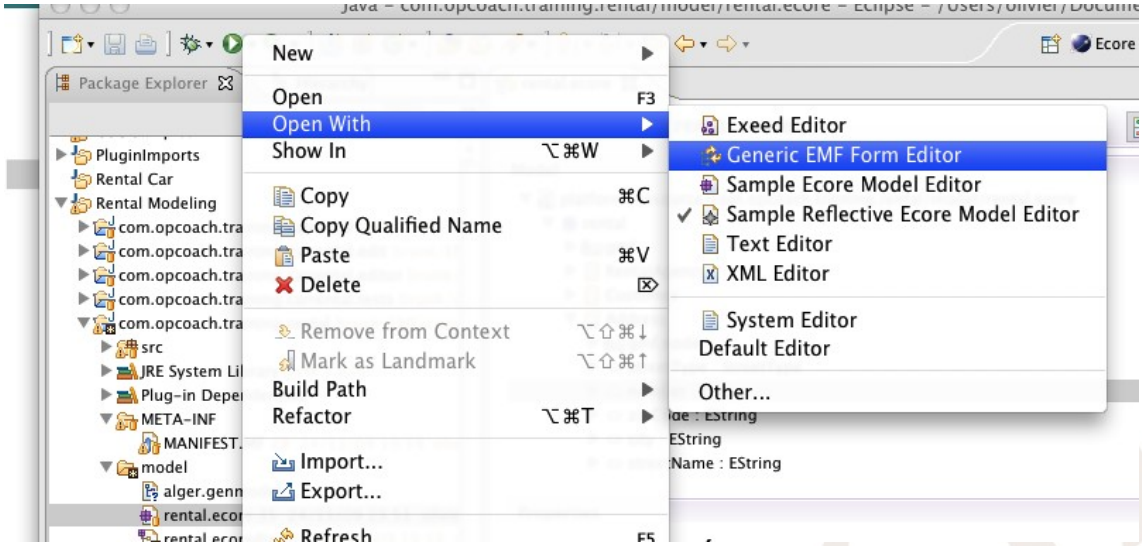
C'est l'éditeur par défaut.



Editeur Arborescent

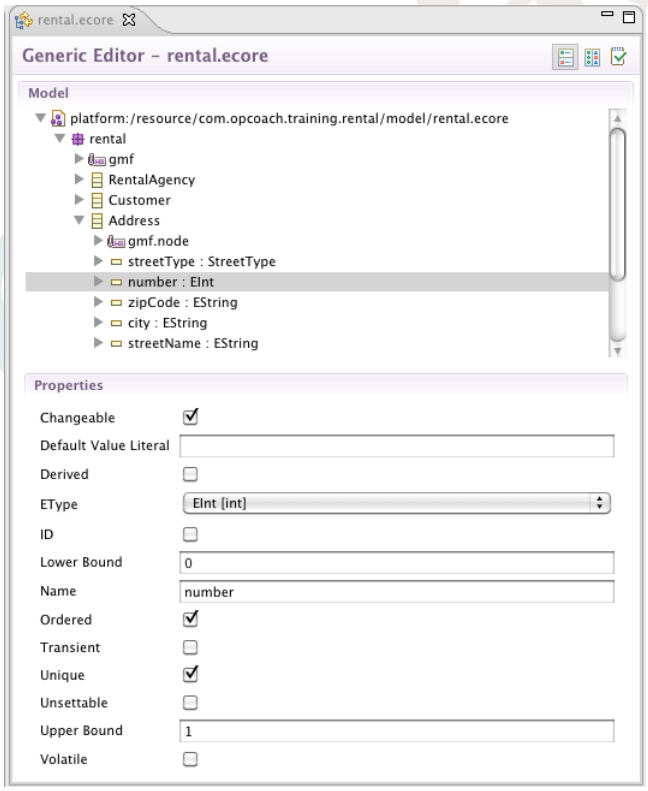
Generic Form Editor

En cliquant sur le fichier ecore, on peut l'ouvrir avec le Generic Form Editor



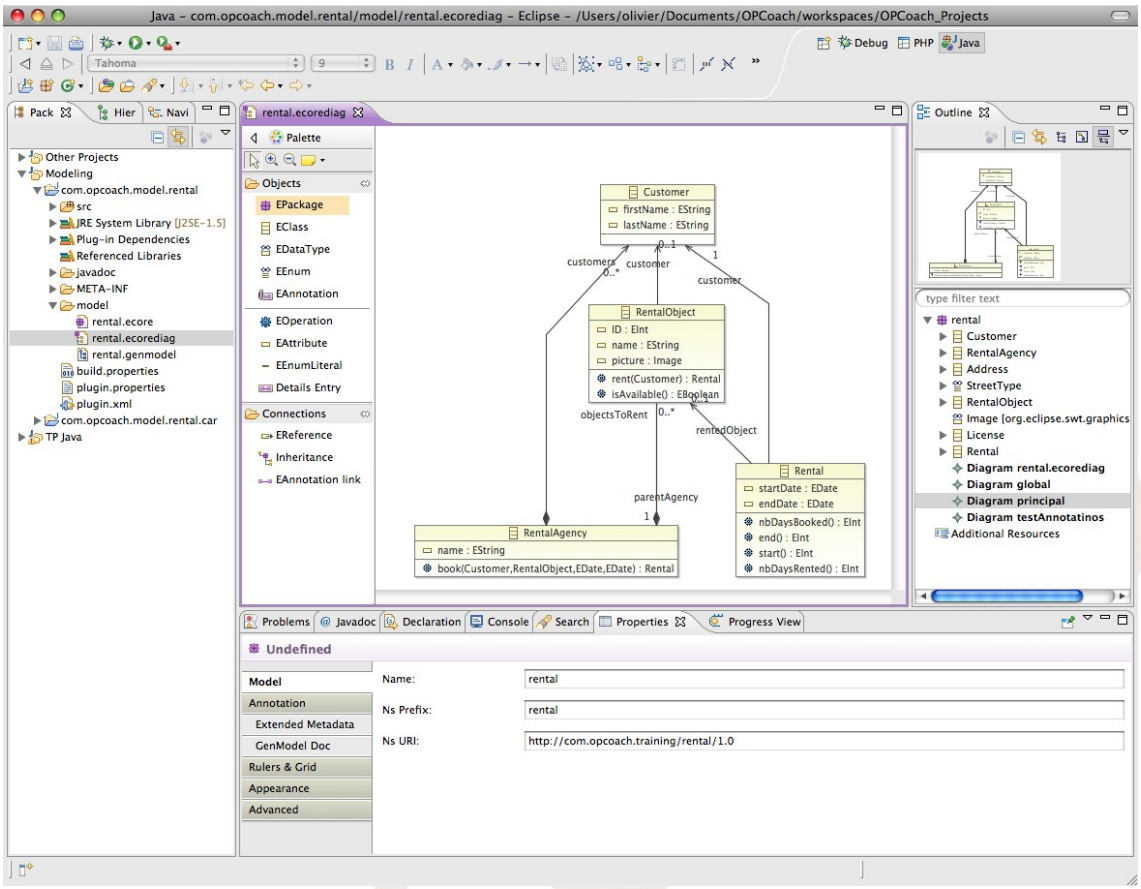
Appel du generic Form Editor

Generic Form Editor



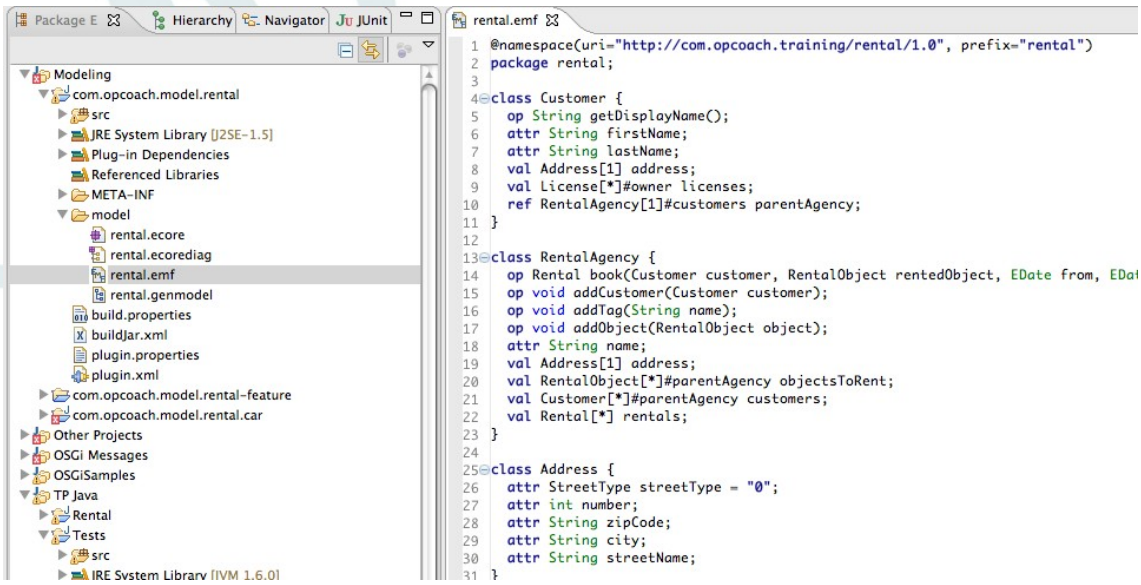
Generic Form Editor

Editeur graphique



Editeur Graphique

Editeur de texte (Emfatic)

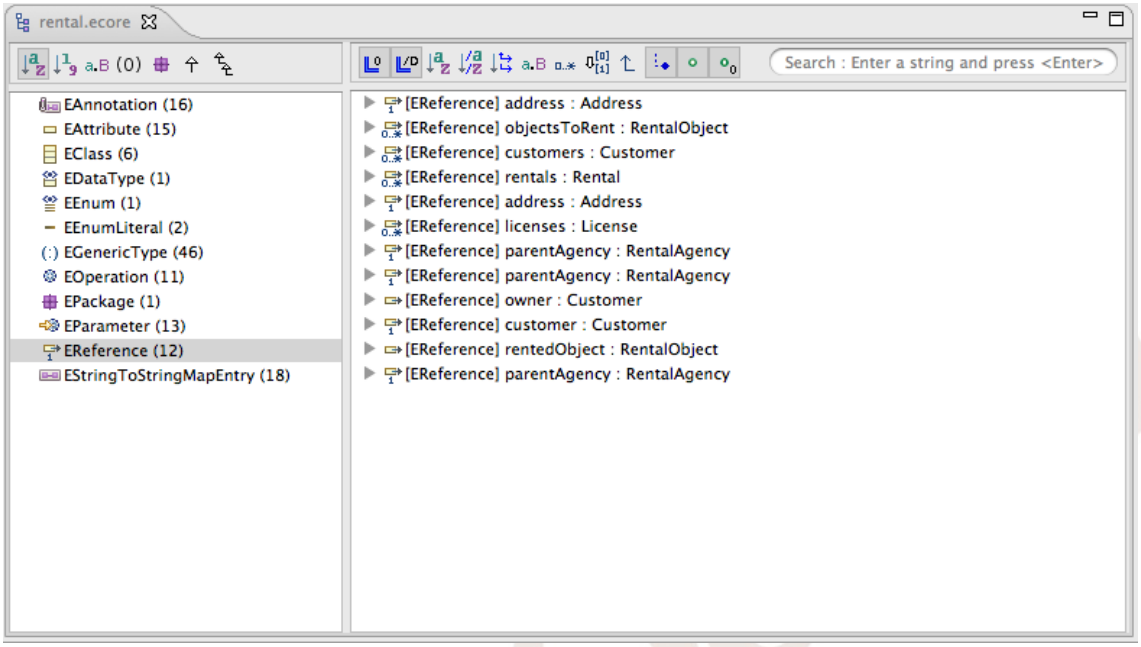


Editeur textuel EMFatic

Attention, la synchronisation est manuelle... Il faut regénérer le Ecore manuellement (menu Generate Ecore File...)

Editeur Modisco

Il permet de naviguer plus facilement sur les objets du modèle



Editeur Modisco

Se télécharge sur les update sites de modisco.
Cf : *http://www.eclipse.org/gmt/modisco/updates/*

C. Model To Text (M2T)

Présentation

Le projet M2T rassemble les technologies pour transformer un modèle en texte
C'est un projet en incubation car créé récemment (mais les projets sont matures).
On y trouve 3 projets :

- JET
- Xpand
- Acceleo

Update site général : **http://download.eclipse.org/modeling/m2t/updates/releases/**

Java Emitter Template (JET)

- Génération avec des balises à la JSP.
- Templates très verbeux
- Utilisé par EMF Cf : org.eclipse.emf.codegen.ecore
- Compliqué à mettre au point
- Paramètre reçu en paramètre quelconque
- Lancement manuel (il faut écrire l'action d'appel) -> code Java

Exemple JET

```
<%@ jet package="translated" imports="java.util.* article2.model.*"
class="TypeSafeEnumeration" %>
<% TypesafeEnum enum = (TypesafeEnum) argument; %>
package <%=enum.getPackageName()%>;
/**
* This final class implements a type-safe enumeration
* over the valid instances of a <%=enum.getClassName()%>.
* Instances of this class are immutable.
*/
public final class <%=enum.getClassName()%> {
<% for (Iterator i = enum.instances(); i.hasNext(); ) { %>
<% Instance instance = (Instance) i.next(); %>
// instance definition
public static final <%=enum.getClassName()%> <%=instance.getName()%> =
new <%=enum.getClassName()%>(<%=instance.constructorValues()%>);
<% } %>
<% for (Iterator i = enum.attributes(); i.hasNext(); ) { %>
<% Attribute attribute = (Attribute) i.next(); %>
// attribute declaration
private final <%=attribute.getType()%> m<%=attribute.getCappedName()%>;
<% } %>
```

Acceleo

- Editeur implémentant la norme MTL (Model Text Language) de l'OMG
- Templates moins verbeux (polymorphisme)
- Outils aboutis :
  - syntax highlighting
  - completion
  - lien avec le modèle
  - launch configuration dédiée
- Alternative plus complète que JET.

Exemple Acceleo 3

generateJava.mtl | common.mtl | classBody.mtl

```
[comment]
  Copyright © 2008 Obeo
  All rights reserved.  This program and the accompanying materials
  are made available under the terms of the Eclipse Public License 1.0

  Any license can be applied to the files generated with this template.

  author <a href="mailto:stephane.bouchet@obeo.fr">Stephane Bouchet</a>
[/comment]
[module classBody('http://www.eclipse.org/uml2/2.1.0/UML')/]
[import common/]

[template public generateClassBody(c : Class)]
public[if (c.isAbstract)] abstract[/if] class [c.name.toUpperFirst()/][for (supe
[for (p : Property | c.getAllAttributes())]
[if (p.upper = -1 or p.upper > 1)]
    /**
     * the [p.name/] attribute.
     */
    private List<[p.type.name/]> [p.name/];
[else]
    /**
     * the [p.name/] attribute.
     */
    private [p.type.name/] [p.name/];
[/if]
[/for]
[for (p : Property | c.getAllAttributes())]
    /**
     * the [p.name/] getter.
     * @return the [p.name/].
     */
    public [if (p.upper = -1 or p.upper > 1)]List<[p.type.name/]>[else][p.type.n
        return this.[p.name/];
```

Exemple Acceleo

Xpand

- Suite de génération totalement outillée
- Editeur complet (syntax highlight, auto completion, refactoring)
- Totalement relié au modèle et au métamodèle
- Invocation des templates polymorphique
- Templates concis et clairs (pas de code de structure)
- Notion de workflow (java, ant)
- Possibilité d'appeler du java (extensions)
- Possibilité de transformer le modèle avant transformation
- Intégration des aspects

Exemple Xpand

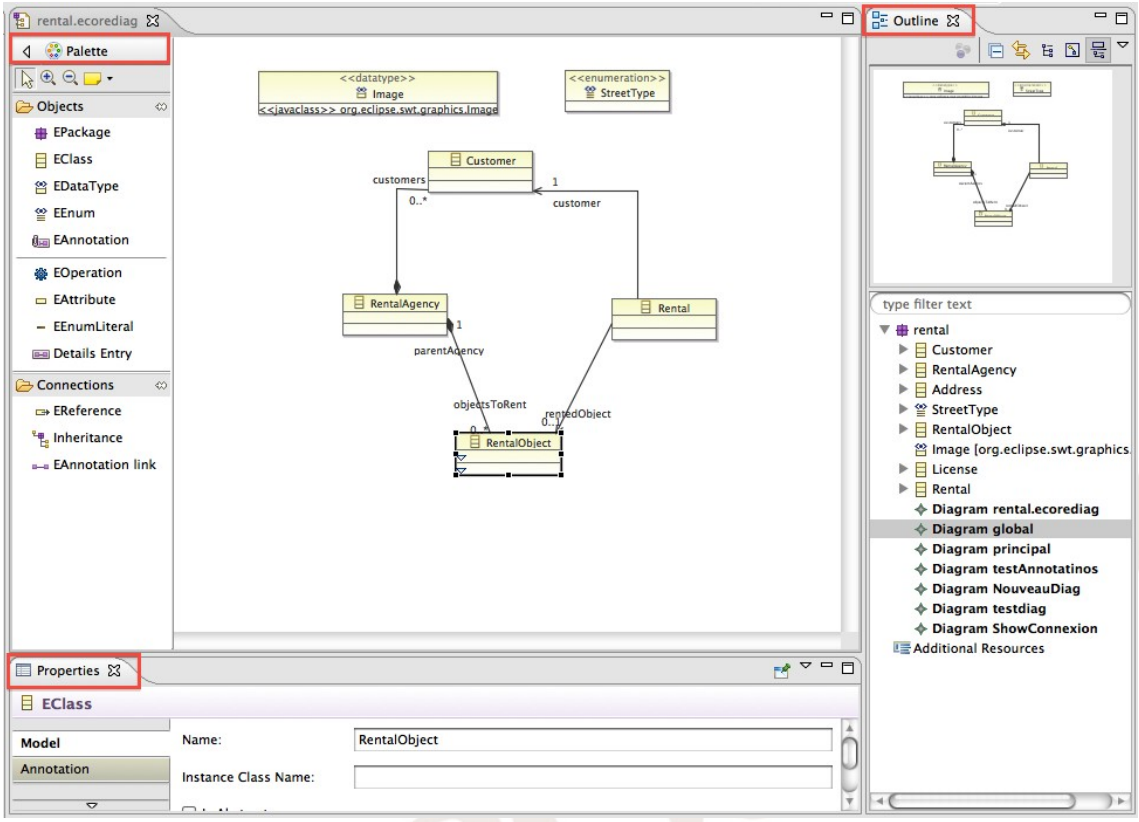
```
1«IMPORT metamodel»
2
3«DEFINE javaClass FOR Entity»
4    «FILE name+".java"»
5        public class «name» {
6            «FOREACH attributes AS attr»
7                public «attr.type» «attr.name»;
8            «ENDFOREACH»
9
10           «FOREACH references AS ref»
11               public «ref.type.name» «ref.name»;
12           «ENDFOREACH»
13       }
14   «ENDFILE»
15«ENDDEFINE»
```

Exemple Xpand

D. GMF, XText

Présentation

- Graphical Modeling Framework (GMF) est un projet d'Eclipse permettant la réalisation d'éditeur graphique à partir de modèles.
- Le diagramme est le nom de la représentation graphique du modèle.
- GMF dépend principalement des projets EMF et GEF
- Les diagrammes sont similaires à ceux de GEF :



Sample diagram

Contenu

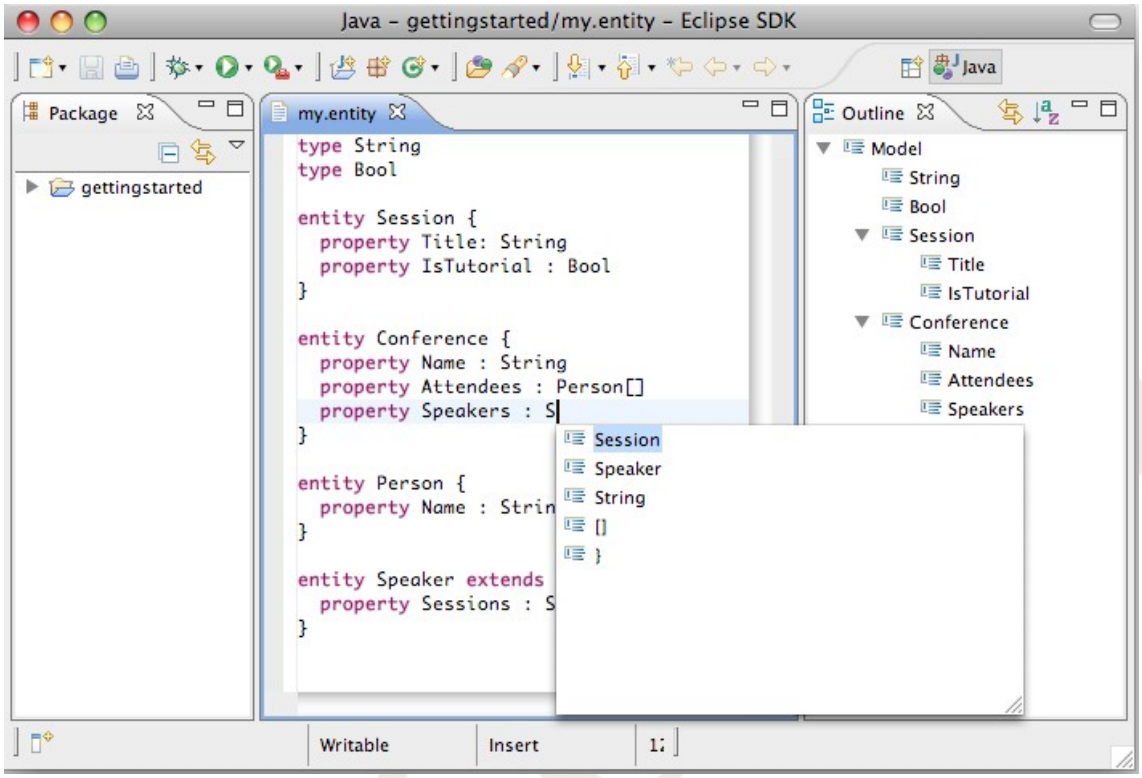
GMF se compose de deux parties :

- une partie outil, permettant de créer un éditeur à partir d'un modèle
  - propose différents modèles d'aide à la génération de l'éditeur
  - propose des générateurs sur ces modèles
  - guide le développement de l'éditeur
- une librairie runtime, utilisée par l'éditeur
  - un ensemble de commandes prédéfinies
  - des properties améliorées (tabbed properties)
  - un page setup et print preview
  - export diagramme en fichier
  - ...

Xtext

- Nouveau projet permettant de générer un éditeur textuel
- 'Language IDE framework'
- Génère l'éditeur à partir d'une grammaire et d'un modèle ecore
- Fournit des outils d'édition et de génération
- Utilise Xtend (ex Xpand) pour la génération
- S'interface avec l'édition java
- Permet de générer le modèle Ecore à partir de la grammaire

Définition de la grammaire



Exemple d'éditeur

Caractéristiques des éditeurs

Tous les éditeurs possèdent les caractéristiques suivantes :

- syntax highlighting
- auto completion
- index pour la recherche
- validation et quickfixes
- gestion des erreurs et warnings
- gestion des templates
- gestion de l'outline
- définition des hyperlink (cross languages)

Synthese

Bonnes pratiques

- Utiliser ecore comme langage de modélisation (import xsd, édition)
- Générer le code EMF ou le code sur mesure
- Architecturer en plugins, bundles, fragments
- Définir et utiliser des points d'extension adaptés
- Gérer proprement les workspaces (working set, team projet set)
- Intégrer les tests dans des fragments
- Ecrire au plus tôt les builds (maven tycho, buckminster)
- Utiliser l'intégration continue
- Organiser et former les équipes