

TP3 : Génération de Code Java et manipulation de modèles + OCL

NB. : Ce TP est à commencer en salle et à finir chez vous au cas où vous manqueriez de temps. Il est important que vous fassiez l'effort de répondre à toutes les questions. Cela est primordial pour la suite des TPs. Ne comptez pas sur une solution de notre part, vous serez déçus ;-)

En TP2 vous avez appris à faire un méta-modèle avec EMF ainsi que des modèles instances dynamiques à travers l'éditeur arborescent. Le tout sans générer de code Java. Aujourd'hui, vous allez apprendre à créer et à manipuler vos modèles instances à travers l'API Java que EMF va générer pour vous. Vous utiliserez également ce que vous avez vu en cours avec OCL afin d'exprimer des règles/contraintes sur vos modèles.

Génération de code Java

Pour générer du code Java à partir du méta-modèle **.ecore**, EMF a besoin de rajouter quelques informations nécessaires à la génération Java. Afin de ne pas polluer/annoter votre **.ecore**, EMF crée un autre modèle à partir de ce dernier contenant ce qu'il faut pour Java. Ce modèle s'appelle le **.genmodel**. Pour obtenir un **.genmodel** à partir de votre **.ecore**,

- 1) click droit sur votre **.ecore** dans le Package Explorer ->**new->other...->Eclipse Modeling Framework-> EMF Generator Model** puis Next
- 2) nommer votre genmodel « **turtle.genmodel** » puis Next
- 3) Laisser l'option par défaut i.e., à partir d'un Ecore Model puis Next
- 4) Cliquez sur **Load...**, puis sur Next
- 5) Puis sur Finish !
- 6) Parcourez le nouveau model généré, vous verrez qu'il ressemble en tout point au **.ecore**. Par contre, si vous ouvrez la version xmi des deux modèles, vous pourrez constater une différence au niveau des concepts/balises utilisés (un autre schéma). Si vous modifiez votre **.ecore**, il vous suffira de faire un click droit sur le modèle (**.genmodel**) puis Reload...
- 7) Maintenant, vous êtes prêts pour lancer la génération de code ! Ouvrez votre **genmodel** dans l'éditeur arborescent, puis click droit sur la racine->**Generate All** pour générer toute l'API Java pour manipuler vos modèles, un éditeur arborescent et les tests qui vont avec. Attention, si vous avez des erreurs de compilation après la génération, il suffit parfois juste de relancer la génération de code. Si ça ne marche toujours pas, demander un coup de pouce à votre enseignant.
- 8) Parcourez le code généré, essentiellement les packages Turtle et Turtle.**impl** (peut être un autre nom **.impl**). Le package Turtle contient principalement les interfaces et les factories qui vous permettront de créer des modèles instances de votre **.ecore**. Le package **.impl** contient les classes d'implémentation de vos interfaces.
- 9) Ouvrez le fichier **plugin.xml** puis allez dans le tab « **Dependencies** », dans la partie **Required plugins**, cliquez sur **Add..** puis tapez **org.eclipse.emf.ecore.xmi**, rajoutez le à la liste et sauvegarder.
- 10) Finalement créez un package **turtle.tools** sous **src** qui contient une classe « **TurtleSerializer** » avec un main pour lancer les exécutions.

IMPORTANT : à chaque fois que vous modifiez le **.ecore** il faudra régénérer le **.genmodel** puis le code. Si entre temps vous avez modifié le code de l'API ou que vous avez rajouté votre propre code en implémentant Display par exemple, il faudra l'annoter avec **generated**

NOT afin que EMF ne l'efface pas à la prochaine génération de code. Prenez le comme une règle, avant de modifier/implanter une méthode, je modifie son annotation de **generated** à **generated NOT**.

Manipulation de modèles instances avec l'API Java générée

A vous de jouer maintenant ! Voici ce qui est demandé :

- 1) On aimerait charger (en mémoire sous forme d'objets Java) le modèle instance que vous avez créé en TP2, le parcourir et afficher ce qu'il fait. En gros, on aimerait connaître les dimensions du Stage, les Choreographies qu'il contient, les actions de chaque Choreography (le type de l'action, la longueur si c'est un Forward, l'angle si c'est un Rotate et la position du pen si c'est un SetPen) ainsi que le nom de la tortue qui exécute l'action ! Pour cela, une fois que vous aurez chargé le modèle avec la méthode load à partir du main de TurtleSerializer, appelez la méthode **Display** de Stage. Si tout se passe bien la méthode devra juste retourner un String « End » ainsi que la chaîne qui suit :

```
SetPen :DOWN  
Turtle :Franklyn  
Forward :100  
Turtle :Franklyn  
SetPen :UP  
Turtle :Franklyn  
End
```

Pour la méthode Load, nous sommes sympas, on vous la donne (voir aussi le poly du cours). Voir partie Aide ci-dessous.

- 2) La deuxième étape consiste à modifier votre modèle, toujours en utilisant l'API Java, et à le sauvegarder. Pour cela vous devrez créer une nouvelle Choreography, de nouvelles actions, des turtles, etc et relier le tout au Stage (du modèle précédent). **Attention** vous devez avoir le méta-modèle en tête pour ne pas oublier d'affecter les références entre les différents concepts de votre langage (Stage, Turtle, Action, etc.). Vos premières erreurs d'exécution seront souvent liées à cela. Vérifiez bien votre graph d'objets avant de le sauvegarder. Nous sommes **mega** sympas, on vous offre la méthode save(). Voir partie Aide ci-dessous. Exécutez de nouveau votre programme après modification de votre modèle pour voir si la nouvelle Choreography ainsi que les nouvelles actions, turtles, s'affichent bien comme il le faut
- 3) **Pour les Geeks et à faire à la maison (PAS EN TP, pas le temps) :** L'idéal est bien sûr d'utiliser les frames Java + l'API 2D de Java pour nous afficher le résultat de l'exécution de votre modèle (voir pdf introduction Java2D + projet Eclipse)

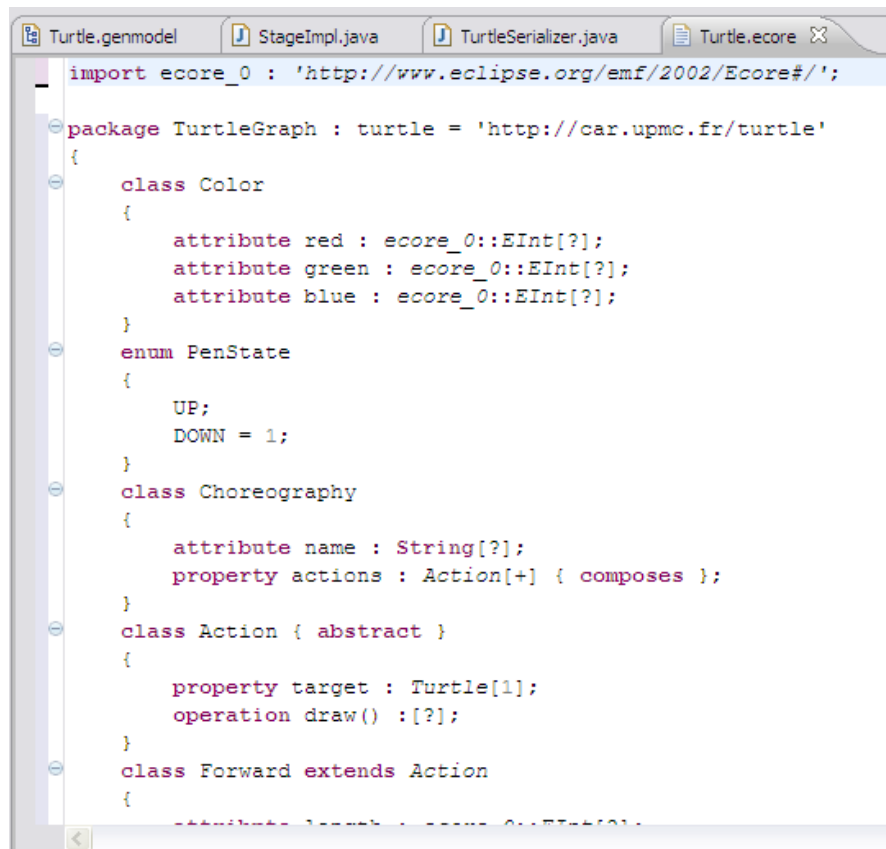
Apportez plus de précision à vos méta-modèles avec OCL

Avant d'appliquer OCL, un premier exercice consiste à exprimer une contrainte sur votre méta-modèle (qui doit être satisfaite par tous les modèles instances) et à la coder en Java. Ensuite, vous devez reformuler cette contrainte (ainsi que pleins d'autres) à l'aide d'OCL afin de voir le gain et l'abstraction offerte par OCL et son interpréteur.

La première contrainte est que dans chaque Choreography, il est interdit d'avoir deux actions de suite de type Rotate. Exprimez cette contrainte en Java puis testez que cela fonctionne correctement avec un modèle instance de méta-modèle de Turtle.

Utiliser OCL (Partie Optionnelle, voir le cours sur le site CAR pour la faire)

- 1) Ce tuto a été réalisé avec une version Eclipse (Helios) (depuis le TP2 vous devriez normalement l'avoir installé sur vos machines). Ensuite rajoutez le projet OCL comme suit : **Install New Software...** puis sélectionnez **Helios** - <http://download.eclipse.org/releases/helios> update site (remplacer par Galileo si vous êtes sur cette version). Choisissez ensuite **OCL Examples and Editors** sous la catégorie **Modeling**. Vous pouvez également télécharger un zip du projet OCL à cette @: <http://www.eclipse.org/modeling/mdt/downloads/?project=ocl> **Attention:** Vérifiez bien la compatibilité avec la version de votre Eclipse et celle de l'Eclipse Modeling Tool project
- 2) Importez votre projet dans le workspace de l'Eclipse que vous venez de télécharger File-> Import...-> Existing Project into workspace
- 3) Click droit sur votre turtle.ecore-> open with OCLinEcore. Cela va vous afficher une définition de votre méta-modèle.ecore sous la forme d'un autre DSL cette fois-ci, spécialement conçu pour y intégrer vos contraintes OCL et que pour cela puisse être interprétable par le moteur OCL d'Eclipse. (voir figure ci-dessous)



```
import.ecore_0 : 'http://www.eclipse.org/emf/2002/Ecore#';

package TurtleGraph : turtle = 'http://car.upmc.fr/turtle'
{
    class Color
    {
        attribute red :.ecore_0::EInt[?];
        attribute green :.ecore_0::EInt[?];
        attribute blue :.ecore_0::EInt[?];
    }
    enum PenState
    {
        UP;
        DOWN = 1;
    }
    class Choreography
    {
        attribute name : String[?];
        property actions : Action[+] { composes };
    }
    class Action { abstract }
    {
        property target : Turtle[1];
        operation draw() :[?];
    }
    class Forward extends Action
    {
        attribute length :.ecore_0::EInt[?];
    }
}
```

- 4) On va maintenant tester une contrainte très simple pour vérifier que l'interpréteur OCL fonctionne correctement. A l'intérieur de l'éditeur OCLinEcore, positionnez vous au niveau de la metaclass Stage et rajoutez la contrainte qui dit qu'un Stage ne doit jamais dépasser une largeur de 400 pix. (contrainte pour les plateformes mobile par exemple). Voici le code de la contrainte :

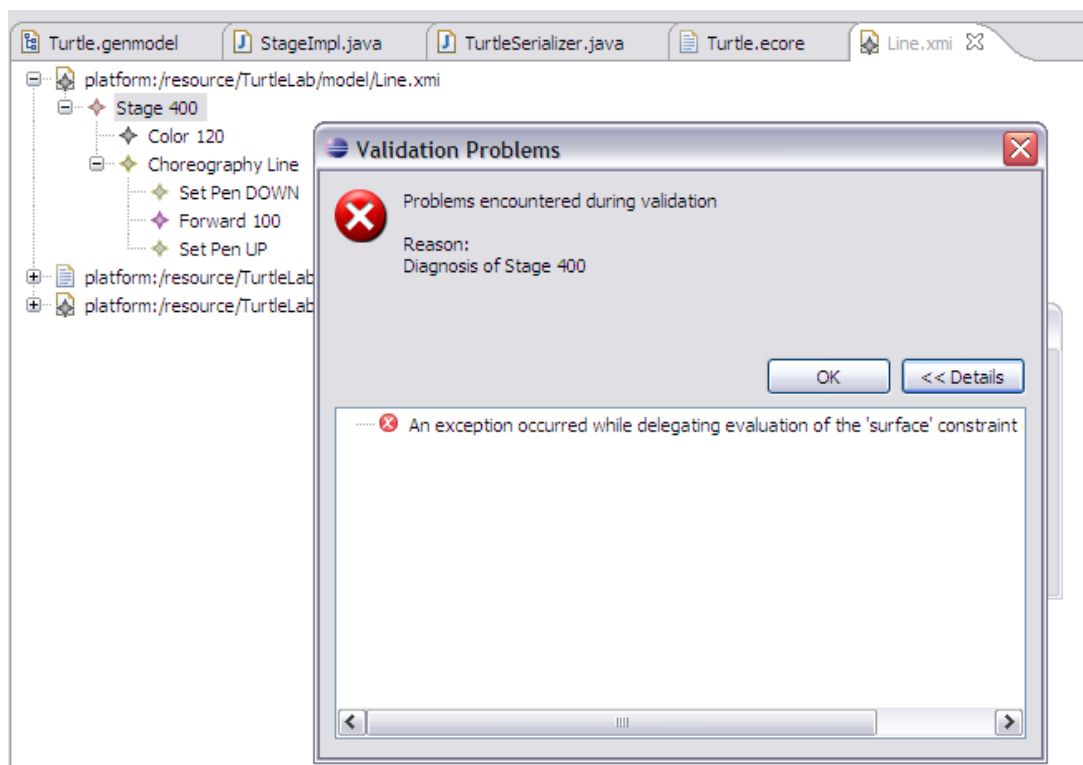
```

Turtle.genmodel | StageImpl.java | TurtleSerializer.java | Turtle.ecore | Line.
}
class SetPen extends Action
{
    attribute state : PenState[?];
}
class ColouredEntity { abstract }
{
    property color : Color[1] { composes };
}
class Stage extends ColouredEntity
{
    invariant surface:
    self.width<=350;

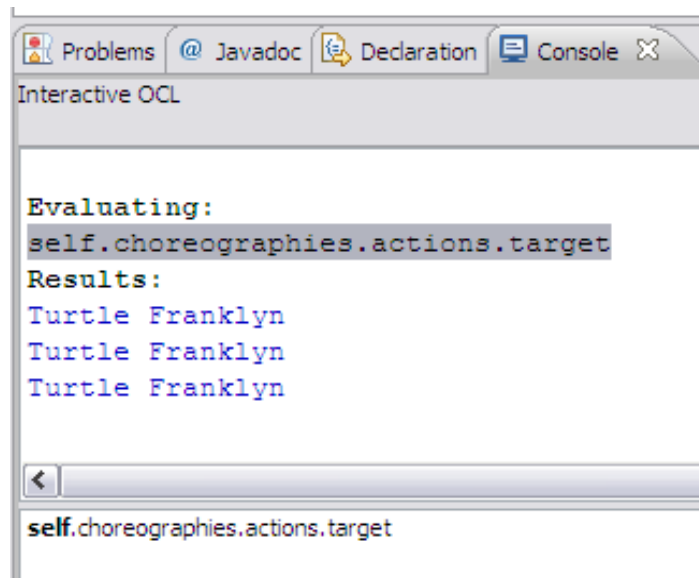
    attribute width : ecore_0::EInt[?];
    attribute height : ecore_0::EInt[?];
    property choreographies : Choreography[+] { composes };
    property farm : Farm[+];
    operation display() : String[?];
}

```

- 5) Sauvegarder votre ecore, puis ouvrez votre modèle instance. Modifiez la largeur de Stage à 400, sauvegardez puis validez votre modèle. Vous devriez avoir cette erreur :



- 6) On peut également faire des requêtes/contraintes d'une manière interactive avec le modèle instance. Pour le tester, click droit sur Stage dans le modèle instance puis Show OCL Console. Vous pourrez ensuite taper vos requêtes dans la fenêtre du bas + Enter et voir le résultat dans celle du haut. Vous pouvez également tester la complétion automatique. Voir figure ci-dessous :



7) A vous de jouer maintenant, formulez les contraintes suivantes :

- Une Choreography ne doit pas contenir plus de 10 actions
- Dans une Choreography, il ne doit pas y avoir deux actions Rotate qui se suivent
- La Turtle « Franklyn » ne fait que des actions de type Forward ou SetPen

La suite au prochain TP :

- Un peu de Transformation de modèles
- Un peu de génération automatique d'éditeur textuel avec JET

Aide :

ATTENTION : tout ce qui est en rouge dans le code qui suit est fonction de votre **.ecore** et du code généré par EMF. Ici, cela suppose que le package de haut niveau de mon **.ecore** s'appelle **TurtleGraph**. Si dans votre cas c'est turtle par exemple, il faudra remplacer dans le code suivant toutes les instances de **TurtleGraphPackage** emp par **turtlePackage**.

```
public class TurtleSerializer {

    public Stage load(File f) {
        ResourceSet rs = new ResourceSetImpl();
        Resource.Factory.Registry registry =
rs.getResourceFactoryRegistry();
        Map<String, Object> m = registry.getExtensionToFactoryMap();
        m.put("xmi", new XMIResourceFactoryImpl());
        rs.getPackageRegistry().put(TurtleGraphPackage.eNS_URI,
TurtleGraphPackage.eINSTANCE);
        URI uri = URI.createFileURI(f.getAbsolutePath());
        Resource resource = rs.getResource(uri, true);

        if (resource.isLoaded() && resource.getContents().size() > 0) {
            return (Stage) resource.getContents().get(0);
        }
        return null;
    }

    public void save(Stage stage, File f) {
        ResourceSet rs = new ResourceSetImpl();
        Resource.Factory.Registry registry =
rs.getResourceFactoryRegistry();
```

```

        Map<String, Object> m = registry.getExtensionToFactoryMap();
        m.put("xmi", new XMIResourceFactoryImpl());
        m.put("ecore", new EcoreResourceFactoryImpl());
        rs.getPackageRegistry().put(TurtleGraphPackage.eNS_URI,
TurtleGraphPackage.eINSTANCE);
        Resource packageResource =
rs.createResource(URI.createFileURI("Turtle.ecore"));
        packageResource.getContents().add(TurtleGraphPackage.eINSTANCE);
        try {
            packageResource.load(null);
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        URI uri = URI.createFileURI(f.getAbsolutePath());
        Resource resource = rs.createResource(uri);
        resource.getContents().add(stage);
        try {
            Boolean>();
            HashMap<String, Boolean> options = new HashMap<String,
            options.put(XMIResource.OPTION_SCHEMA_LOCATION, Boolean.TRUE);
            resource.save(options);
        } catch (IOException e) { e.printStackTrace(); }
    }

    public void UpdateStage (Stage stage, Choreography choreography){
        stage.getChoreographies().add(choreography);
    }

    public static void main(String[] args) {
        // A VOUS DE JOUER !!!
    }
}

```