Rédacteurs	Aymen Abedmouleh – Pierre Laforcade
Relecteurs	Esteban Loiseau
Date	30/08/14
Référence	GRAPHIT-D5.1
Version	0.2

Spécifications et développement des VIDL de bas niveau

D5-1

■ Historique du document

Version	Date	Auteurs	Modifications
0.1	01/07/13	AA	Rédaction des spécifications comme chapitre de thèse
0.2	15/08/14	PL	Remise en forme et reprise du discours

Table des matières

Index des Figures

1 Objectifs de ce document

Ce livrable concerne la tâche 5 centrée sur la proposition de solutions IDM/DSM pour la spécification et le développement de *Visual Instructional Design Languages* (VIDLs) exploitant le métier identifié et formalisé pour les plateformes (en relation avec la tâche 4). Dans le cadre du projet GraphiT nous avons retenu deux formes de VIDLs :

- ceux exploitant directement le métier de conception de la plateforme existante visée ; nous les appelons les VIDLs de *bas* niveau ;
- ceux qui cherchent à proposer un *pont* entre ce que permet la plateforme en terme de conception, et ce que les utilisateurs souhaiteraient utiliser comme outils de conception : appelés VIDLs de *haut* niveau.

Ce document s'intéresse uniquement aux VIDLs de bas niveau. Ces VIDLs ont pour but d'exploiter le métamodèle du métier de conception de la plateforme directement comme syntaxe abstraite pour l'élaboration du langage de modélisation graphique. La sémantique du VIDL correspondra à celle de la plateforme. La notation graphique (syntaxe concrète du langage) sera le seul élément sur lequel le VIDL pourra tenter de se démarquer afin de proposer une ergonomie et une IHM exploitant les avantages de la modélisation graphique en opposition aux écrans de paramétrage proposée par la plateforme. Les icônes graphiques devront toutefois reprendre ceux de la plateforme quand ils existent afin de servir de repères aux futurs concepteurs.

Les plateformes MOODLE (version 2.0) et GANESHA sont les deux plateformes sur lesquelles nous avons expérimenté la spécification et le développement de VIDL de bas niveau. Ce livrable s'intéresse spécifiquement à présenter la spécification du VIDL dédié à Moodle.

2 Présentation de l'outillage : Eclipse Modeling Project

L'*Eclipse Modeling Project* (EMP) est un projet spécifique dédié à la communauté Eclipse supportant l'évolution et la promotion des technologies de développement dirigées par les modèles. Ce projet propose un panel très large de frameworks pour faciliter le développement des outils/prototypes selon une approche dirigée par les modèles **[EMP 2012]**. Les deux frameworks principaux utilisés, dans le cadre de ce travail de développement d'outils de conception graphiques, sont les frameworks EMF (*Eclipse Modeling Framework*) et GMF (*Graphical Modeling Framework*). Ces deux frameworks facilitent le développement des prototypes (des éditeurs de modèles) grâce à la génération automatique du code source Java. Leur utilisation est basée principalement sur la définition d'un ensemble de modèles (modèle du domaine et les modèles de la couche graphique (*tooling*, *graph* et *mapping*).

2.1 Eclipse Modeling Framework

EMF est un framework dédié à Eclipse pour la modélisation et la génération automatique ou semi-automatique du code. Il permet de spécifier des métamodèles et de gérer des modèles conformes à ces métamodèles **[Pelechano et al. 2006]**.

Les métamodèles peuvent être spécifiés en utilisant différentes syntaxes : Java annoté, schéma XML, diagramme de classe UML, etc. Une fois importés par EMF à partir de l'un de ces formats ou manuellement spécifiés sur le framework, les métamodèles sont transformés dans un format conforme au méta-métamodèle Ecore. Ce dernier, utilisé par EMF, est compatible avec le métamétamodèle MOF (*Meta Object Facility*) proposé par l'OMG (*Object Management Group*).

EMF intègre des éléments fondamentaux tels que *EMF.Edit* et *EMF.Codegen* et *EMF.Edit* comprend un ensemble de classes génériques et réutilisables pour la construction des éditeurs à partir des métamodèles EMF **[Bézivin et al. 2004]**. *EMF.Codegen* fournit le service de génération de code pour construire un éditeur complet à partir d'un métamodèle EMF.

L'assistant graphique du framework EMF permet de spécifier les options de génération du code des éditeurs **[Bézivin et al. 2004]**. L'éditeur généré utilise les classes du framework *EMF.Edit* qui fournissent un accès aux données des modèles afin de les afficher dans des interfaces graphiques de base : par exemple des tableaux contenant l'ensemble des éléments d'un modèle ou des feuilles de propriétés adaptées à chaque élément d'un modèle.

Le code généré par EMF n'est pas toujours suffisant. Pour cela, il est souvent nécessaire d'apporter des modifications sur ce code ou d'ajouter d'autres fonctionnalités **[Kelly 2004]**. Le framework EMF repose sur un générateur de code qui permet, à partir du modèle de domaine, de générer trois couches de l'outil d'édition. Ces couches sont :

- la couche modèle représente le modèle de domaine (le métamodèle) sous forme de classes Java ;
- la couche du *EMF.Edit* est l'ensemble de classes permettant la visualisation et l'édition des modèles ;
- la couche éditeur représente l'interface graphique de l'éditeur. La figure illustre le processus de génération d'un outil de conception (éditeur) par le framework EMF. Elle représente également l'ensemble des relations qui existent entre les couches et les modèles.

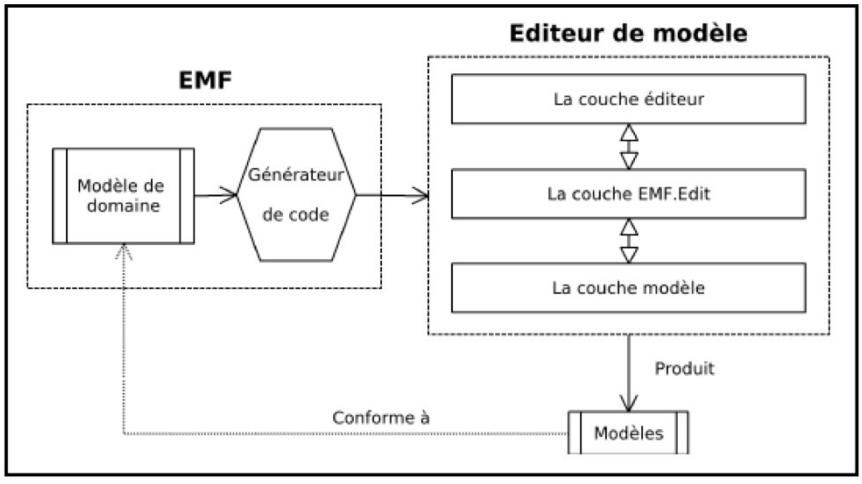

Figure 1: Processus EMF **[Zendagui 2010]**

L'interface graphique de l'éditeur générée par EMF (figure 2) est très basique. Elle permet la création des éléments du modèle par l'intermédiaire d'une arborescence composée des éléments de base du modèle du domaine. Le framework GMF intervient pour surmonter la faiblesse de l'interface graphique générée par EMF en proposant des solutions pour générer des interfaces plus élaborées et surtout graphiques.

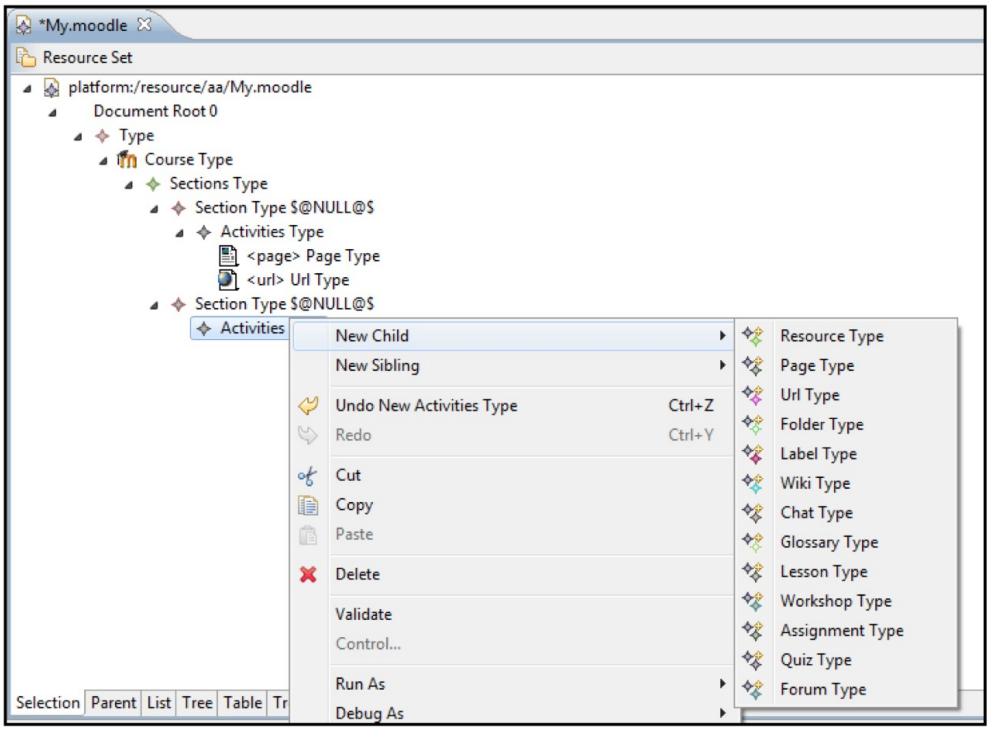

Figure 2: Aperçu de l'éditeur généré par EMF

2.2 Graphical Modeling Framework

GMF est un framework spécifique pour le développement et l'exécution des éditeurs graphiques. Il se base sur les frameworks EMF et GEF (*Graphical Editing Framework*). En prenant le rôle de pont entre ces deux frameworks, GMF fournit de nombreuses fonctionnalités permettant d'automatiser certaines tâches qui étaient réalisées à la main en utilisant directement les frameworks EMF et GEF. Les différents éléments créés au sein des frameworks EMF et GEF sont assemblés grâce à un modèle de mapping (*Mapping Model*).

GMF permet le développement d'éditeurs graphiques riches à partir d'un modèle de domaine formalisé avec EMF. Le développement d'outil DSM impose l'utilisation simultanée des frameworks EMF et GEF pour prendre en compte à la fois la modélisation des concepts d'un domaine et leur notation graphique. GEF fournit le support graphique requis pour la construction d'un éditeur de diagramme qui remplace la couche graphique générée par EMF **[Kelly 2004]**.

Le framework GMF est composé de deux composants principaux : un ensemble d'outils et une infrastructure d'exécution. L'outillage est constitué des éditeurs permettant de créer/modifier des modèles décrivant la notation graphique, la sémantique et l'aspect outillage d'un éditeur graphique ainsi que d'un générateur de code des éditeurs graphiques. L'infrastructure d'exécution permet l'exécution des plugins générés, elle met à disposition des fonctionnalités permettant de produire des éditeurs graphiques universels et extensibles. Elle offre :

- un ensemble de composants réutilisables pour la construction des éditeurs graphiques, telles que l'impression, l'exportation d'image, les barres d'outils, etc. ;

- un modèle standardisé pour décrire les éléments du diagramme, séparant ainsi la sémantique (domaine) de la notation (graphique) des éléments ;
- une infrastructure de commande qui synchronise les commandes d'EMF et de GEF ;
- une structure extensible qui rend les éditeurs graphiques générés ouverts et extensibles.

GMF propose la persistance des modèles créés via son éditeur dans un format XML synchronisé avec la vue graphique en cours d'édition. Ce mécanisme peut être configuré de manière à ce que le format XML ne soit pas du XMI mais du code XML. Cette caractéristique de GMF nous intéresse parce qu'elle permet la formalisation des futurs modèles produits en XML, conformément au schéma XML du modèle du domaine représentant le langage de conception pédagogique de la plateforme utilisée, et enfin supportable par les modules d'import/export.

GMF s'appuie sur des modèles principaux pour le développement des éditeurs graphiques. Ces modèles, basés sur le modèle de domaine, sont :

1. Le modèle de définition graphique (*Graphical Definition Model*), portant l'extension .gmfgraph. Il est généré à partir du modèle de domaine (.ecore) en utilisant l'assistant GMFGraph. Ce modèle embarque les informations de la représentation graphique (les figures, les nœuds, les compartiments, les liens, etc.). La couche graphique est composée de deux catégories d'éléments :
   - les composants de base de l'interface graphique qui sont indépendants du modèle de domaine et communs pour toutes les interfaces générées par les frameworks EMF et GMF tels que les menus, la palette, les tableaux, etc.
   - la notation graphique des éléments du modèle de domaine qui contient la définition de la représentation graphique des éléments du domaine. Elle dépend du modèle de domaine utilisé.

2. Le modèle de définition d'outillage (*Tooling Definition Model*), portant l'extension .gmftool. Il définit l'aspect outillage de l'éditeur graphique en utilisant l'assistant GMFTool. Il est utilisé pour la conception de la palette d'outils et d'autres outils périphériques comme les menus et les barres d'outils.

3. Le modèle de définition du Mapping (*Mapping Definition Model*), portant l'extension .gmfmap. Il définit en utilisant l'assistant GMFMap les liaisons entre le modèle du domaine, le modèle de la définition graphique et le modèle de la définition d'outillage. Il permet de lier un concept du domaine (défini dans le modèle de domaine) avec sa représentation graphique (définie dans le modèle de définition graphique) et les outils qui le gèrent (définis dans le modèle de définition de l'outillage).

4. Le modèle de génération du code de l'éditeur, portant l'extension .gmfgen. Il est généré à partir du modèle de définition de mapping. Il permet de paramétrer les détails d'implémentation qui seront utilisés lors de la génération du code final de l'éditeur.

La figure 3 illustre le processus GMF et les étapes nécessaires pour la génération des éditeurs graphiques. Elle représente également les modèles et les relations qui existent entre eux. Les différents modèles sont généralement incomplets et nécessitent plusieurs ajouts, modifications et paramétrages complétés à la main. Dans les sections suivantes, nous détaillons notre utilisation de ces frameworks et la définition des modèles pour le développement des outils de conception. Nous présentons également les différentes manipulations et modifications exercées sur ces modèles.

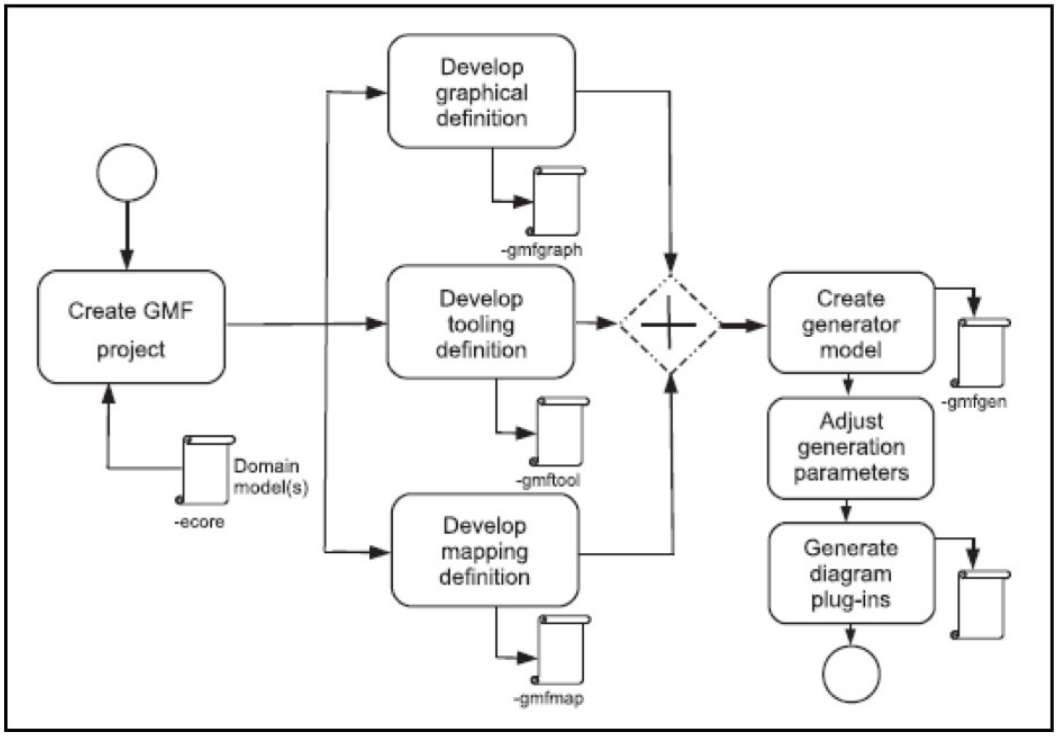

Figure 3: Aperçu du processus de GMF [Laforcade 2010]

3 Mooditor : éditeur graphique pour Moodle

3.1 Introduction

Mooditor est un outil de conception de scénarios pédagogiques destiné aux enseignants-concepteurs utilisant la plateforme de formation à distance Moodle. Il consiste à proposer une interface graphique pour la conception des scénarios pédagogiques en se basant uniquement sur le langage de conception pédagogique spécifique à cette plateforme. Lors du développement du Mooditor, nous avons adopté l'approche *Domain-Specific Modeling*. Nous avons utilisé les deux frameworks présentés précédemment (EMF et GMF) comme un cadre pratique technique pour l'application de cette approche.

3.2 Les exigences des praticiens

Dans une première expérimentation, nous avons choisi de mettre l'accent sur certains objectifs et besoins des praticiens permettant la spécification et le développement d'un VIDL et d'un éditeur afin de vérifier notre approche DSM et son outillage **[Abedmouleh et al. 2012d] [Laforcade et Abedmouleh 2012]**. Nous avons identifié les exigences suivantes :

- Dessiner / modéliser graphiquement les sections des cours en les arrangeant dans l'espace de dessin sans spécifier un ordre définitif (des sections).
- Permettre le dessin des relations (sous forme de flèches) entre les sections pour représenter leurs futurs ordres sur la plateforme.

- Offrir, dans la palette de l'outil, les activités et les ressources de base proposées par la plate- forme.
- Permettre l'ajout des activités et des ressources au sein des sections et la définition de leurs utilisations sans avoir à spécifier toutes les données habituelles et requises pour chacune d'elles.

Concrètement, les praticiens visent des outils de création (*authoring-tool*) orientés diagrammes, spécifiques aux plateformes qu'ils utilisent et qui leur permettront de se concentrer sur la conception globale du cours.

3.3 Architecture de Mooditor

L'architecture de Mooditor est composée de trois composants principaux : le composant IHM, le composant de gestion de modèles et de métamodèles et le composant du module d'importation. La figure 4 illustre l'architecture de Mooditor, les différents composants et les différentes interactions entre eux. Pour certains éléments de cette architecture tels que le composant IHM et les interactions entre les composants, le code a été généré automatiquement par les frameworks EMF et GMF. Cependant, nous avons développé certaines parties afin de palier certains manques et pour répondre à certains besoins.

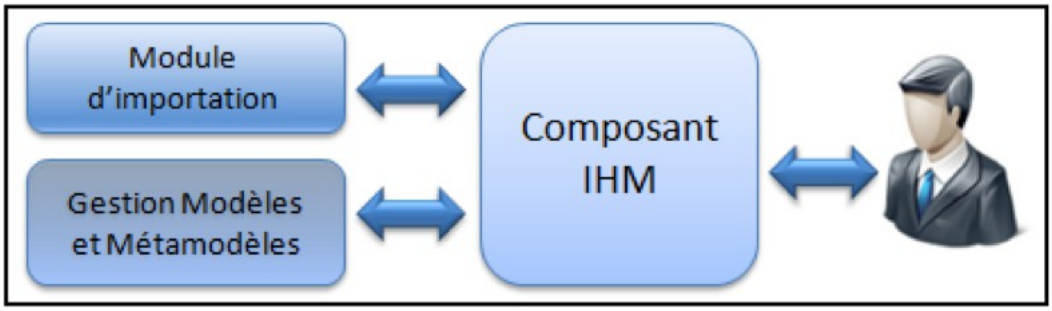

Figure 4: Architecture simplifiée de Mooditor

Le composant IHM (Interface Homme Machine) représente l'interface graphique de l'outil de conception (éditeur). Cette interface permet aux enseignants-concepteurs d'utiliser les fonctionnalités de l'éditeur. L'interface graphique de Mooditor est composée des objets graphiques basiques pour la définition de l'IHM et de la notation graphique des éléments du modèle du domaine (le métamodèle du langage de conception pédagogique de la plateforme Moodle). Le framework GMF est utilisé pour le développement de ce composant en parallèle avec le framework EMF qui définit le domaine métier de l'éditeur.

Le composant de gestion de la couche modèle et métamodèle gère les relations entre les modèles et les métamodèles. Les fonctionnalités de cette couche permettent l'édition des modèles conformément à leurs métamodèles. Le code java de ce composant généré par le framework EMF regroupe les fonctionnalités de la gestion du modèle du domaine.

Le module d'importation permet d'importer les scénarios pédagogiques sur l'espace de l'éditeur. Ces scénarios sont à l'origine ceux exportés de la plateforme Moodle via le module d'import/export ou bien spécifiés et sauvegardés par l'éditeur dans un format externe (XML).

3.4 Mooditor : les principales étapes et modifications

Le développement de l'éditeur est basé sur le modèle du domaine (mooditor.ecore), basé principalement sur le métamodèle du langage de conception pédagogique de la plateforme Moodle [Abedmouleh et al. 2011c] [Laforcade et Abedmouleh 2012]. La figure 5 illustre un extrait de ce modèle. Dans un premier temps, nous avons généré le modèle *mooditor.genmodel* requis par le framework GMF. Ensuite, nous avons généré les modèles d'outillage (*mooditor.gmftool*), de la couche graphiques (*mooditor.gmfgraph*), de mapping (*mooditor.gmfmap*) et le modèle de générateur (*mooditor.gmfgen*) par l'intermédiaire des assistants de création de modèles de GMF. Ces assistants permettent de générer une première version de cet éditeur. Cependant, plusieurs ajouts et modifications doivent êtres apportés sur les modèles et le code généré afin d'adapter l'éditeur aux exigences de développement et des praticiens.

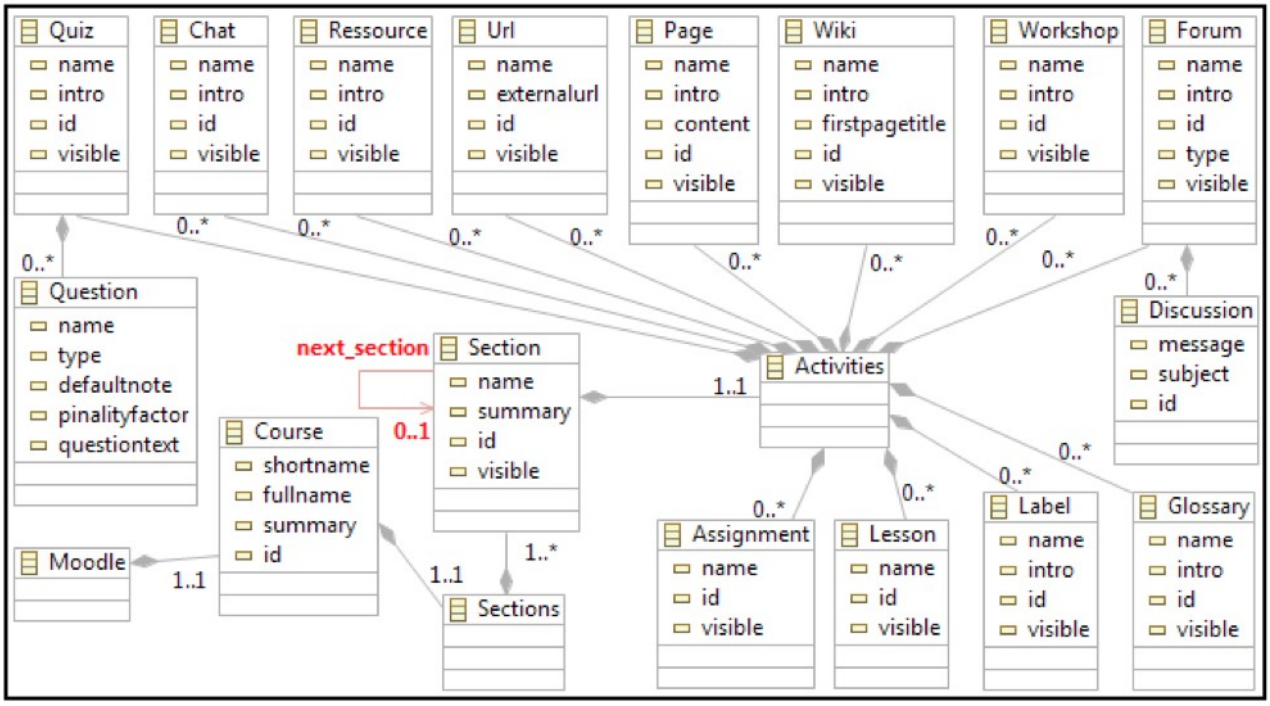

Figure 5: Extrait du modèle de domaine final

3.5 Développement de la couche graphique de Mooditor

Cette partie est consacrée à la définition des éléments composant la couche graphique de l'éditeur. Elle nécessite la définition de plusieurs modèles dont certains sont dépendants du modèle du domaine (tels que le modèle d'outillage et de la définition graphique). La figure 6 illustre de façon simplifiée des modèles utilisés que nous détaillons dans les sections suivantes.

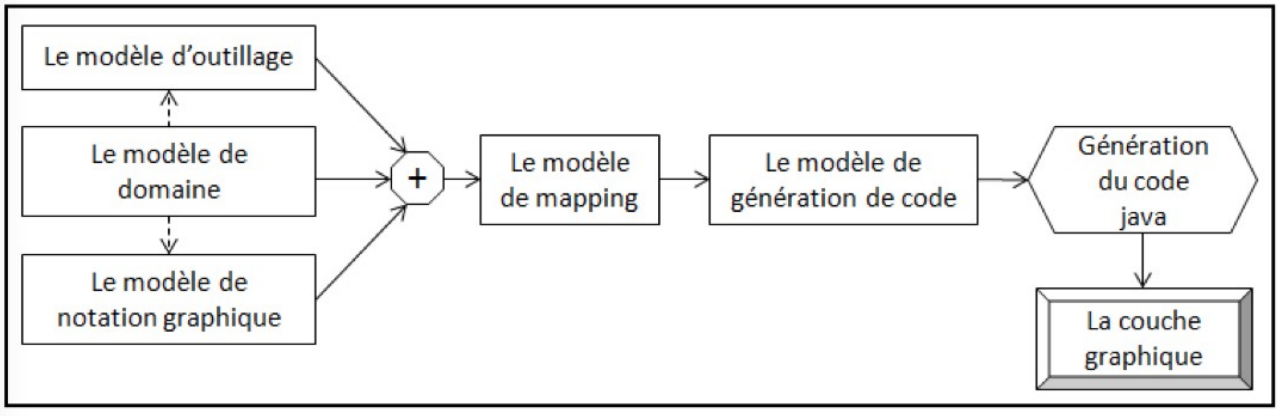

Figure 6: les modèles de définition de la couche graphique

3.5.1 La définition de l'outillage : le modèle mooditor.gmftool

Le modèle d'outillage consistera à définir les éléments mis à disposition de l'enseignant-concepteur au sein de l'éditeur pour la spécification de son cours ou de son scénario pédagogique. Ces éléments sont définis au sein de la palette, des menus ou des barres d'outils de l'éditeur. L'éditeur que nous avons développé est caractérisé par une seule palette contenant tous les éléments de conception du cours spécifiés dans le modèle du domaine.

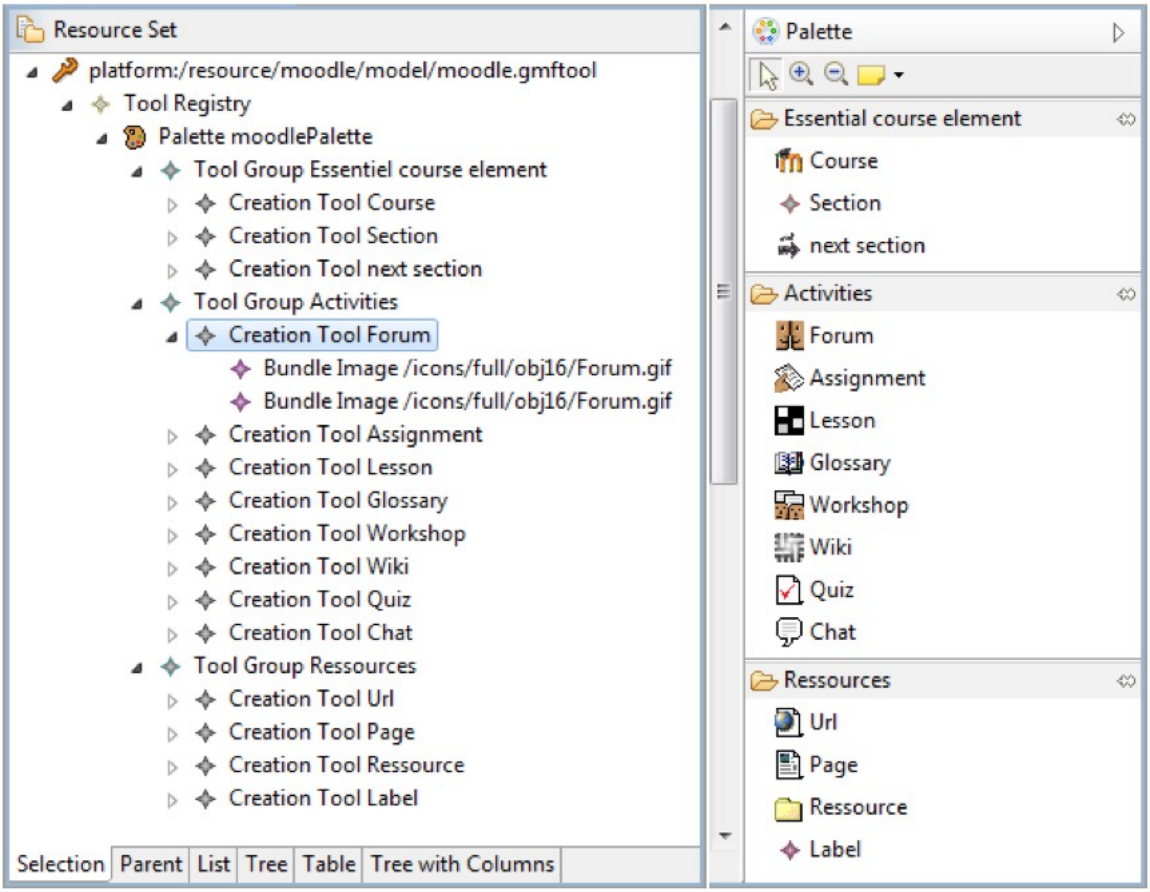

Figure 7: Modèle de définition d'outillage et la palette de l'éditeur

Dans cette palette sont définis trois groupes d'outils. Le premier contient les outils de création des éléments de base (un cours, des sections, les groupes et le lien entre les sections). Les autres groupes contiennent les outils de création des concepts de cours en termes d'activités et de ressources définies dans le modèle du domaine. Afin de faciliter le design sur l'éditeur, nous avons associé une icône spécifique à chaque outil de création d'un élément du cours. Nous avons choisi de reprendre les mêmes icônes utilisées au sein de la plateforme Moodle. La figure 80 illustre le modèle de définition d'outillage (*mooditor.gmftool*) (la partie à gauche) et la palette graphique (la partie à droite).

3.5.2 La définition graphique : le modèle mooditor.gmfgraph

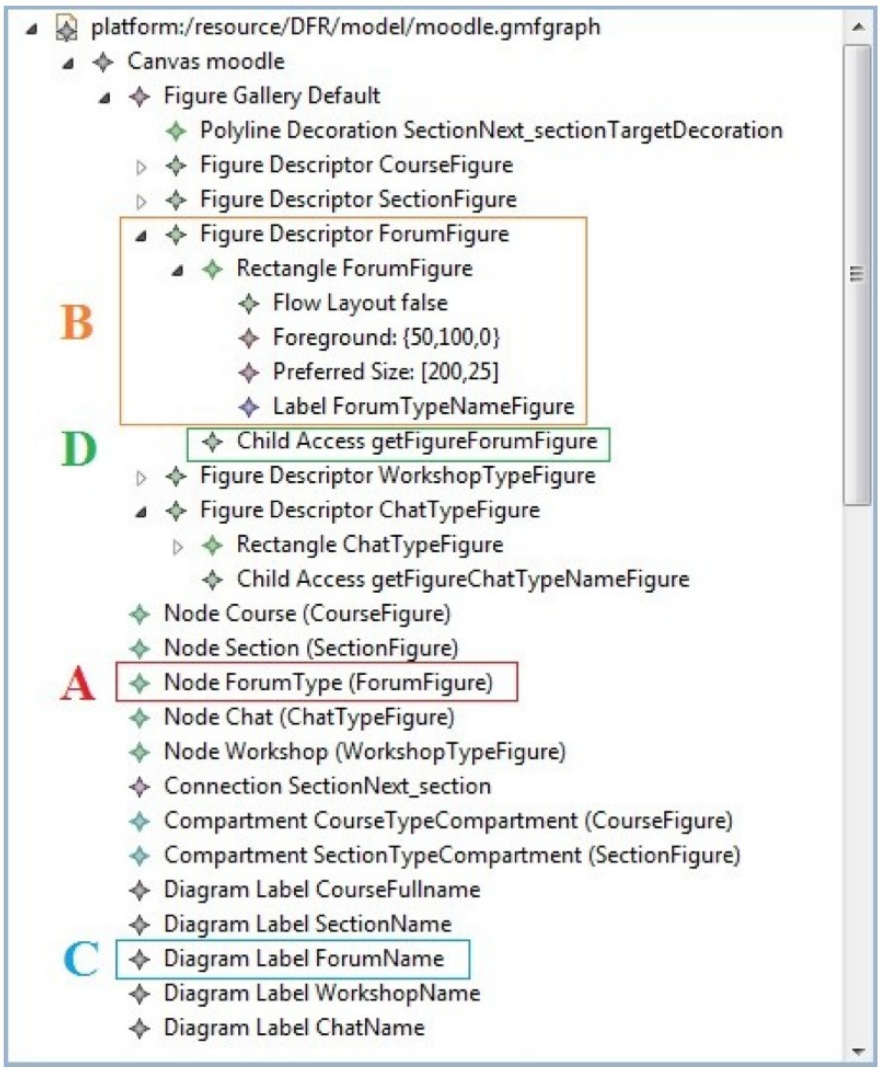

Figure 8: Représentation graphique d'un forum

Le modèle de définition graphique (*mooditor.gmfgraph*) consiste à définir les éléments à utiliser au sein de l'espace de design de l'éditeur graphique. Quelques éléments de ce modèle, tels que les figures et les rectangles descriptifs, sont définis automatiquement par l'intermédiaire de l'assistant *GMFGraph*. Cependant, plusieurs améliorations doivent être apportées à la définition des éléments générés par l'assistant et les éléments manquants.

Le modèle *mooditor.gmfgraph* est conséquent. Dans ce document, nous ne pouvons pas présenter l'ensemble de la modélisation graphique que nous avons définie dans le modèle graphique. En revanche, nous illustrons en exemple la définition graphique du concept du forum. La figure 8 illustre un extrait des éléments du modèle *gmfgraph* associés au forum et que nous décomposons en quatre parties.

Chaque élément du modèle du domaine doit être défini par un nœud qui désigne la racine de

la définition graphique. La partie entourée par le cadre (A) montre le nœud *forum*. Il servira pour le mapping au sein du modèle *mooditor.gmfmap* (nous détaillerons cela dans la section suivante). Il est décrit par l'intermédiaire d'un élément de type *Figure Descriptor*.

Les éléments de type *Figure Descriptor* consistent à définir les détails de la représentation graphique. Chaque élément est composé de deux blocs. Par exemple sur la figure 8, le premier bloc entouré par le cadre (B) définit la structure de la représentation graphique du concept (*forum*). Elle est décrite par des éléments graphiques de base. A titre d'exemple, le *forum* est décrit par l'intermédiaire d'un *rectangle* et dispose d'un objet de type *label*. La couleur de fond et la taille du rectangle sont des exemples des propriétés du *rectangle* représentant le *forum*. Les *labels* (étiquettes) sont utilisés pour initialiser et afficher un texte et/ou une icône au sein du *rectangle* suite à l'ajout d'un élément (*forum* par exemple). Le contenu de ces objets peut être défini au moment de la définition du modèle graphique ou saisi par l'enseignant-concepteur au moment de l'édition du modèle (le cours). Le deuxième bloc entouré par le cadre (C) montre les *labels* utilisés pour la représentation graphique du concept *forum*.

La partie entourée par le cadre (D) représente la définition des méthodes qui permettent de fournir les valeurs des différents labels définis dans le premier bloc. Une méthode doit être définie pour chaque label ajouté dans le modèle. Ensuite, le code de ces méthodes est généré automatiquement par le framework GMF.

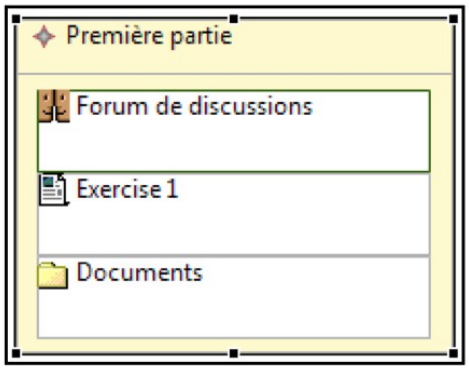

Figure 9: représentation graphique du concept section

La figure 9 montre un exemple d'une section définie dans l'espace de définition graphique de l'éditeur. Cette section se présente comme un rectangle dont le fond et les bordures sont de couleur beige clair. La représentation graphique de la section est définie dans le modèle *mooditor.gmfgraph* en utilisant un élément graphique de type *Rectangle*. La section est munie d'un label représentant le nom de la section (Première partie). À l'intérieur de ce rectangle se trouvent les activités et les ressources ajoutées par l'enseignant-concepteur en utilisant la palette de l'éditeur. Chaque élément tel que *forum* est muni d'un *label* pour spécifier son nom.

Afin de rendre les sections capables d'embarquer plusieurs éléments, nous avons associé à chaque section un élément appelé compartiment. Les compartiments nécessaires pour l'éditeur n'étaient pas générés par les assistants de GMF. Ils ont été définis manuellement dans la couche graphique de l'éditeur puis ils ont été associés aux éléments concernés.

3.5.3 Le mapping entre les modèles de base de GMF : le modèle mooditor.gmfmap

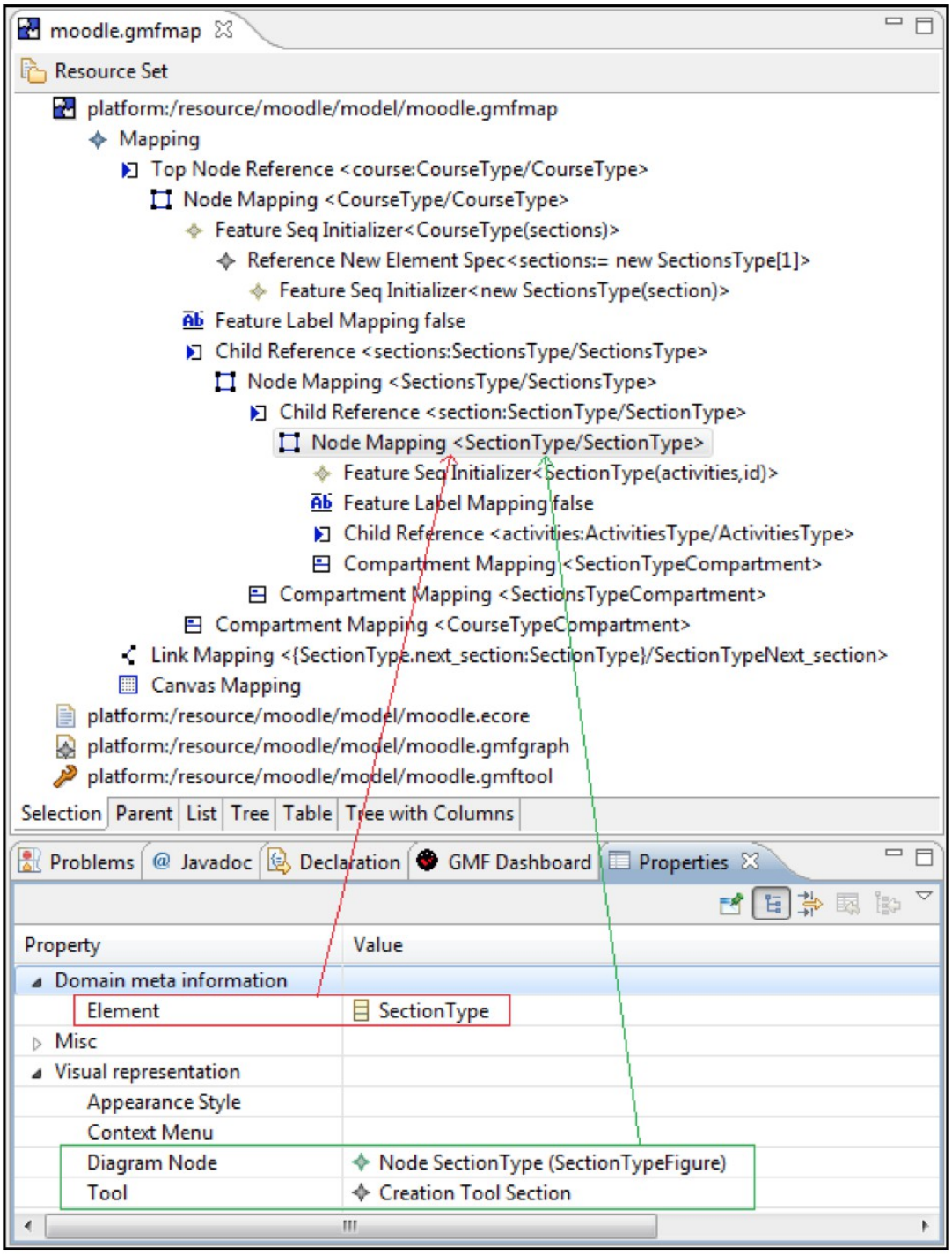

Figure 10: Extrait du modèle mooditor.gmfmap

Le modèle *mooditor.gmfmap* consiste à définir les relations entre les modèles de domaine, de définition graphique et de définition de l'outillage. Il permet de relier un concept du modèle de domaine (défini dans le *.ecore*) avec leurs représentations graphiques (définies dans le *.gmfgraph*) et les outillages définis pour leurs créations (définis dans le *.gmftool*). L'assistant de GMF permet de générer une première version de ce modèle à partir des trois modèles précédents. Cependant, cette version n'est pas complète et nécessite l'ajout de plusieurs éléments manquants tels que les compartiments, les initialisations, le mapping aux labels, etc. La figure 83 illustre un extrait complet de ce modèle.

Nous avons spécifiquement présenté cet extrait du modèle parce qu'il montre deux types d'éléments importants. Le premier est *Feature Seq Initializer* qui consiste à initialiser certains éléments du modèle pendant leurs créations. Dans cet exemple, l'initialisation permet de créer une section dès la création d'un nouveau cours. Le deuxième élément est de type *Node Mapping*. Il permet de relier un élément du modèle de domaine avec sa représentation graphique et les outillages définis pour sa création. Dans cet exemple, l'élément du modèle de domaine section (cf. la partie de la figure 10 encadrée en rouge) qui est relié avec sa représentation graphique et l'outillage défini pour la création d'une nouvelle section (cf. la partie de la figure 10 encadrée en vert).

3.5.4 Génération du code Java de la couche graphique : le modèle mooditor.gmfgen

Après la définition des modèles requis par le processus GMF (modèle du domaine, modèle d'outillage, modèle de la définition graphique et le modèle de mapping), l'assistant de création de modèle de GMF permet de générer un nouveau modèle appelé le modèle générateur (*mooditor.gmfgen*). Nous avons apporté quelques modifications sur ce modèle en ajustant les paramètres responsables à la génération du contenu du modèle et de sa présentation graphique dans deux fichiers séparés. Le premier fichier représente la définition graphique du modèle spécifié. Il contient les propriétés graphiques du modèle, par exemple les coordonnées des différents objets. Le deuxième fichier représente les éléments du modèle de domaine, utilisés pour la spécification du cours. Il est conforme au modèle de domaine (correspondant au langage de conception pédagogique de la plateforme Moodle).

3.5.5 Mooditor : un plugin d'Eclipse

Après la définition de tous les modèles du processus GMF, il est possible de réaliser la génération automatique du code Java des éditeurs graphiques en tant qu'un *plugin* d'Eclipse ou une application *standalone*. La figure 84 représente une capture d'écran de l'interface de l'outil généré composé de trois parties principales :

- La partie entourée par le cadre (A) représente la palette de l'outil.
- La partie entourée par le cadre (B) représente l'espace de dessin et de conception. Dans cette palette, les modèles sont créés et édités sous forme graphique, par les enseignants-concepteurs.
- La partie (C) représente la grille de spécification des attributs et des propriétés des éléments du cours.

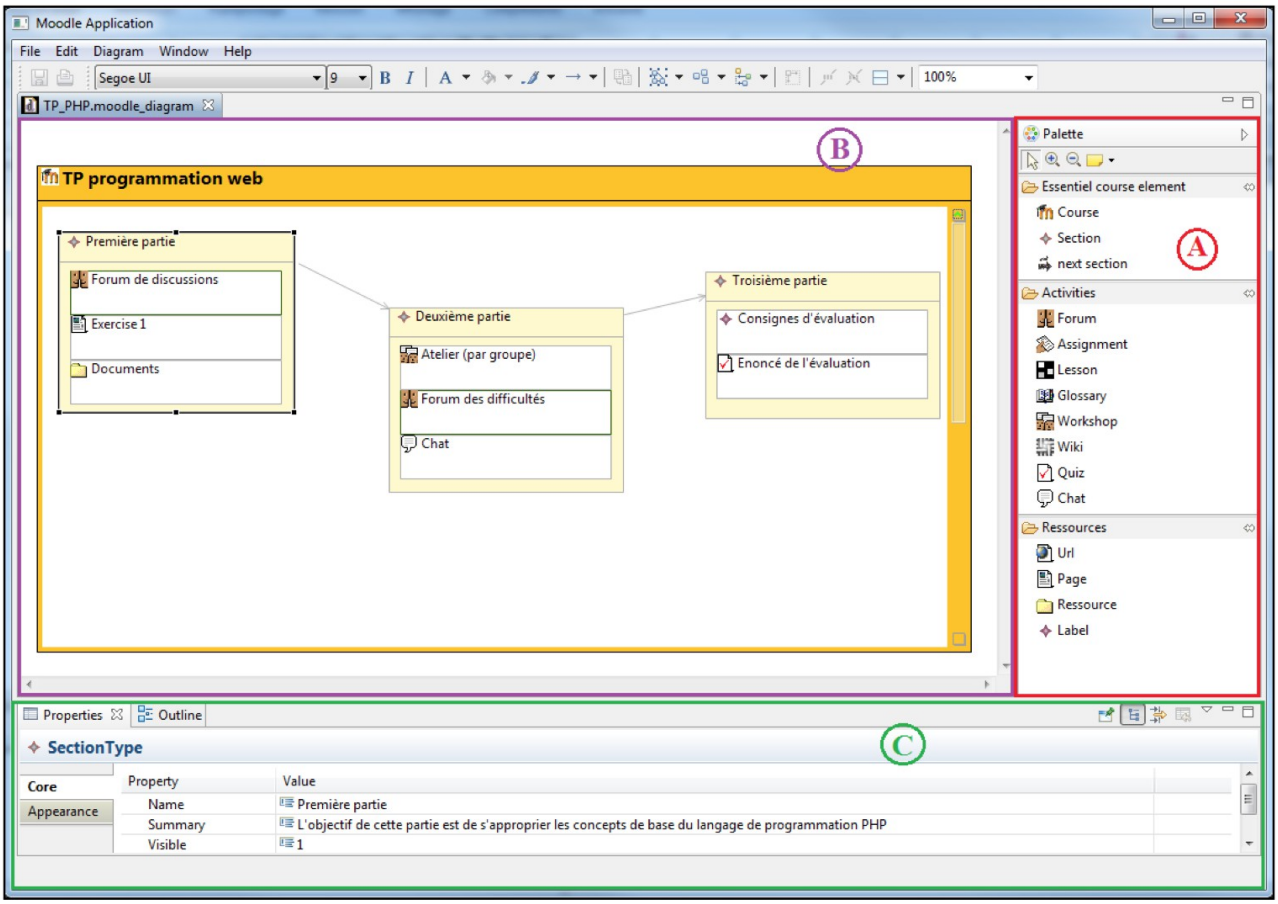

Figure 11: Capture d'écran de Mooditor

L'éditeur final peut être utilisé (1) pour spécifier graphiquement des scénarios d'apprentissage en tant que modèles graphiques et (2) pour visualiser un scénario pédagogique à partir d'un autre outil, mais en satisfaisant la condition d'être conforme au schéma utilisé. Les modèles sont visualisés dans une vue orientée diagramme et sérialisés de façon synchrone en tant que fichier XML interprétable pas la machine conformément avec le schéma XML.

Cet éditeur graphique répond aux exigences des praticiens précédemment citées). Les enseignants-concepteurs peuvent compléter leur conception directement sur Moodle en spécifiant les détails de bas niveau qui ne sont pas traités par cet éditeur. Ils peuvent également exporter les cours de la plateforme pour les concevoir à nouveau sur l'éditeur sans perdre les détails de bas niveau (qui seront fusionnés pendant l'importation par le module spécifique de Moodle que nous avons élaboré).

3.5.6 Résolution de certains problèmes fonctionnels

Après la génération automatique du code de l'éditeur via les frameworks EMF et GMF, en particulier la première version, nous avons constaté certains manques/défauts de conception nécessitant plusieurs améliorations. Afin de palier ces défauts, nous avons amené des modifications, des ajouts et des para- métrages d'éléments dans les modèles de ces frameworks.

Concrètement, nous avons identifié certains défauts de design dont les trois principaux sont :

(i) L'éditeur ne permet pas la définition des relations graphiques entre les sections de cours. Les sections

(ii) Lors de l'importation d'un fichier XML d'un cours sur l'espace graphique de l'éditeur, spécifié via le même éditeur ou exporté via le module de communication ajouté à Moodle, les relations entre les sections ne sont pas initialisées.

(iii) Lors de l'enregistrement du cours spécifié via l'éditeur, les sections sont ordonnées par ordre de création et non par ordre pédagogique. En effet, l'enseignant a besoin parfois de spécifier les sections de son cours. Ensuite, il détermine leur ordre définitif par des flèches. La logique de EMF consiste à cartographier (*mapper*) la notion d'ordre en accord avec la création des sections dans le compartiment d'un cours. Il s'agit concrètement d'un problème parce que les praticiens ne savent pas nécessairement à l'avance l'ordre concret des sections qu'ils conçoivent. Il est ainsi important de différencier l'ordre d'instanciation des sections de leur ordre pédagogique. Afin de remédier ce manque, nous avons apporté des modifications sur le modèle du domaine (le métamodèle de conception pédagogique de Moodle) et sur le code java généré par EMF **[Abedmouleh et al. 2011C]**.

Dans un premier temps, les modifications ont consisté à :

- Ajouter une *EReference nextSection* dans la classe *section* (*section eClass*) avec des bornes inférieure / supérieure défini à "0..1" (voir la relation *next_section* en rouge sur la figure 5).
- Définir l'attribut *transient* à vrai ('*true*') dans le but d'informer le mécanisme de persistance EMF de ne pas traiter l'élément *nextSection*.
- Définir la notation correspondante (ligne avec une flèche) à l'élément *nextSection* dans le modèle graphique et la palette de l'éditeur.
- Associer les deux nouveaux éléments ensemble (la représentation graphique et l'élément de la palette associés à *nextSection*) en spécifiant le lien entre eux sur le modèle de mapping.
- Ajouter des contraintes OCL (*Object Constraint Language*) au modèle de Mapping, afin de ne pas autoriser (1) la relation réflexive à une section et (2) les cycles de sections. Dans cette partie, nous n'avons pas modifié le code généré. Ce code, représentant le plugin de l'outil, peut être utilisable pour spécifier des modèles conformes au modèle de domaine.

Dans un deuxième temps et pour résoudre le problème de (ii) et (iii), nous avons apporté des modifications sur le code généré par EMF afin de redéfinir le comportement des méthodes de sauvegarde (*save*) et d'import (*load*). La méthode de sauvegarde se charge de l'enregistrement du modèle sous un format XML. Les instructions de code ajoutées permettent à cette fonction d'enregistrer les instances de *sections* ordonnées à partir de la *section* parent en tenant en compte les relations *nextSection* spécifiées. La figure 12 illustre un extrait du nouveau code associé à la fonction *save*().

La méthode d'import *load*() doit définir l'élément *nextSection* sur l'espace graphique du cours en tenant en compte l'ordre des balises section parcourues à partir du fichier XML. Nous avons défini les instructions indispensables pour la spécification des relations entre les sections. La figure 13 représente la partie du code associée à la fonction *load*().

```java
public void save(Map<?, ?> options) throws IOException {
        if (this.getContents() != null)
            if (this.getContents().get(0) != null) {
                MoodleType momo = (MoodleType) this.getContents().get(0);
                SectionsType  s = null;
                if (momo.getCourse() != null)
                    if ((s = momo.getCourse().getSections()) != null)
                        s.reordonneSections2();
            }
        super.save(options);
    }
```

```java
/**
     * @generated NOT
     */
    public void reordonneSections2() {
        EList<SectionType> listeSection = getSection();

        int ty = listeSection.size();
        SectionType prems = null; //le premier qu'on va chercher

        for (int i=0; i< ty ; i++){
            prems = (SectionType)listeSection.get(i);
            if ((prems.getPrevious_section()) == null){
                        //pas de precedent => c'est le premier (s'ils sont tous chainés)
                while (prems.getNext_section() != null){
                    SectionType next = prems.getNext_section();
                    listeSection.move(listeSection.indexOf(prems), listeSection.indexOf(next));
                    prems = next;
        }}}}
```

Figure 12: Extrait du nouveau code de la fonction save()

```java
@Override
/**
     * @generated NOT
     */
    public void load(Map<?, ?> options) throws IOException {
        super.load(options);
        if (this.getContents() != null)
            if (this.getContents().get(0) != null) {
                MoodleType momo = (MoodleType) this.getContents().get(0);
                SectionsType  s = null;
                if (momo.getCourse() != null)
                    if ((s = momo.getCourse().getSections()) != null)
                        s.createNextLinks();
    }}
```

```java
/**
     * @generated NOT
     */
    public void createNextLinks() {
        if (this.getSection() == null)
            return;
        Iterator<SectionType> it = this.getSection().iterator();
        if (!it.hasNext())
            return;
        int ty = getSection().size();
        SectionType prec = null;
        for (int i=0;i<ty;i++){
            SectionType s = getSection().get(i);
            if (prec != null && prec!=s)
                prec.setNext_section(s);
            prec = s;
    }}
```

Figure 13: Extrait du nouveau code de la fonction load()

4 Conclusion

Ce livrable a été consacré à la représentation du premier outillage que nous avons développé afin d'instrumenter nos propositions théoriques. Le développement de l'éditeur pour la plate-forme Moodle a représenté un nouveau terrain pour expérimenter l'approche DSM. Cet outil/prototype a représenté un cas de figure d'utilisation du métier d'une plateforme. Nous nous sommes basés sur le modèle de conception pédagogique de la plateforme Moodle, identifié grâce à l'application du processus que nous avons proposé. Ce métamodèle a servi comme base (le domaine métier) pour le développement de l'éditeur Mooditor : outil de conception des scénarios pédagogiques spécifiques à la plateforme Moodle. Ce métamodèle est utilisé par les frameworks EMF et GMF de l'*Eclipse Modeling Project* pour la génération du code Java de Mooditor. La première version de Mooditor a été générée automatiquement. Cependant, plusieurs modifications et ajouts ont été nécessaires afin de surmonter les défauts d'utilisation. Ces modifications ont concerné principalement le métamodèle de base, les modèles de la définition graphique et un nombre important d'instructions du code générés par les frameworks utilisés.

Nous avons également mené une deuxième expérimentation de l'approche DSM en développant un outil de conception de scénarios pédagogiques, similaire à celui de Moodle mais

spécifique pour la plateforme Ganesha. Nous n'allons pas montrer les étapes de développement parce qu'elles sont similaires à celle du développement de l'éditeur Mooditor. L'éditeur de Ganesha repose sur le langage de conception pédagogique dédié à cette plateforme, identifié via l'application du processus proposé (livrables D4.1 et D4.2). Le développement de cet éditeur s'appuie également sur les frameworks EMF et GMF. Nous avons défini les modèles requis à la génération de cet éditeur (les modèles de domaine, d'outillage, de notation graphique, de mapping et de génération de code). Toutefois, plusieurs modifications sont apportées sur les modèles et le code généré automatiquement par les frameworks tels que la redéfinition des fonctions *save*() et *load*(). Cet éditeur spécifique à Ganesha est illustré en figure 14.

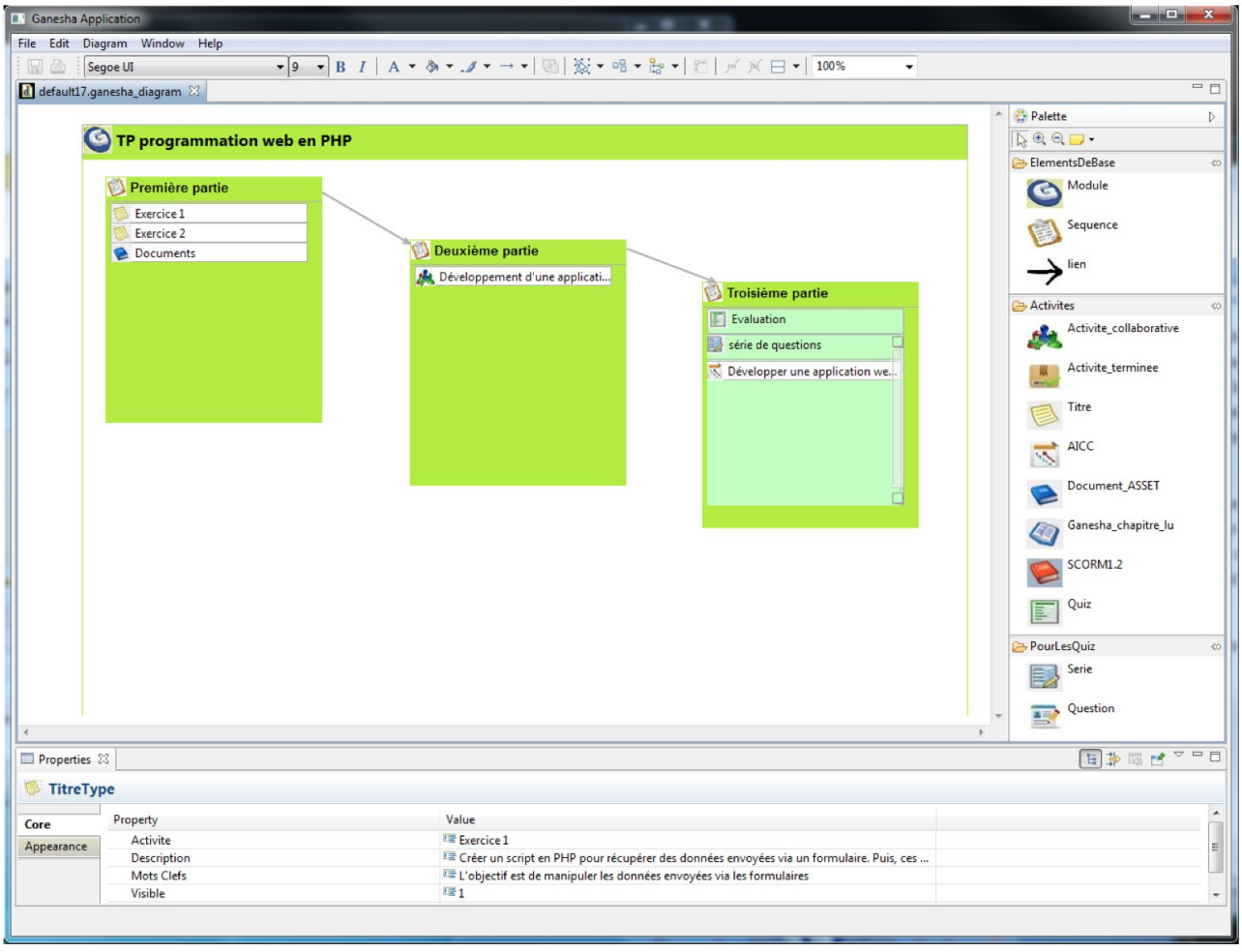

Figure 14: Exemple de conception de scénario sur l'éditeur de Ganesha

5 Références

[Abedmouleh et al. 2011c]Abedmouleh A., Laforcade P., Oubahssi L. et Choquet C. (2012). Operationalization of learning sce- narios on existent learning management systems: the Moodle case-study. In: International Conference on Software and Data Technologies In Proceedings of the 6th International Conference on Software and Data Technologies, isbn : 978-989-8425-77-5, Seville, Espagne, 18-21 juillet 2011, p.143-148.

[Abedmouleh et al. 2012d] Abedmouleh A., Laforcade P. et Oubahssi L.. (2012). Specification of visual instructional design lan- guages dedicated to Learning Management Systems. In: International Conference on Software and Data Technologies, Rome, Italie, 24-27 juillet 2012, p. 199-204.

[Bézivin et al. 2004] Bézivin J., Blay-Fornarino M., Bouzeghoub M., Estublier J. et Favre J.M. (2004). Rapport de synthèse de l'AS CNRS sur le MDA (Ingénierie Dirigée par les Modèles), novembre 2004.

[Kelly 2004] Kelly S. (2004). Comparison of Eclipse EMF/GEF and MetaEdit+ for DSM. In : 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, Workshop on Best Practices for Model Driven Software Development, Vancouver, Canada, 24-28 octobre 2004,.

[Laforcade et Abedmouleh 2012] Laforcade P. et Abedmouleh A. (2012). Improving the design of courses thanks to graphical and ex- ternal dedicated languages: a Moodle experimentation. In: Moodle Research Conference 2012, Heraklion, Greece, 14-15 septembre 2012, p. 94-101.

[Pelechano et al. 2006] Pelechano V., Albert M., Muñoz J. et Cetina C. (2006). Building tools for model driven development comparing microsoft DSL tools and eclipse modeling plug-ins. In : Proceedings of the 11th Confer- ence on Software Engineering and Database (JISBD'06), Sitges, Spain, 3-6 octobre 2006.

[Zendagui 2010] Zendagui B. (2010). Modélisation de l'observation dans un contexte de réingénierie – Une approche dirigée par les modèles. Doctorat de l'université du Maine.

5.1 Références sur le WEB

[DSM 2012] DSM forum (2012). http://www.dsmforum.org/. Consulté le 15/09/2012

[EMP 2012] Eclipse Modeling Project (2012). Accessible à : http://www.eclipse.org/modeling/ (consulté en dé- cembre 2012).

[Ganesha 2013] http://www.ganesha.fr/

[GMF 2013] http://www.eclipse.org/modeling/gmp/. Consulté le 25/01/2013.