Rédacteur	PL
Relecteur	EL
Date	
Référence	GRAPHIT-D2.7
Version	01/01/00

L'Ingénierie Dirigée par les Modèles et le Domain-Specific Modeling

Théories et applications dans le cadre du projet GraphiT

D2-7

■ Historique du document

Version	Date	Auteurs	Modifications
0.1	29/09/13	PL	Plan, objectifs et premiers contenus
0.2	11/05/15	PL	Contenu additionnel
1	04/07/15	PL	Contenu final + mise en forme finale

Table des matières

Index des Figures

1 Objectifs de ce document

Ce livrable traite de l'Ingénierie Dirigée par les Modèles (IDM) et du *Domain-Specific Modeling* (DSM). En effet ces domaines du Génie Logiciel représentent le cadre théorique et technique du projet GraphiT quant à la résolution de la problématique générale portant sur l'élaboration de langages graphiques de conception dédiés à des plateformes spécifiques.

Ce livrable a pour objectif de rendre compte des théories, techniques et outils de l'IDM/DSM pertinents pour le projet. Etant donné qu'il existe déjà des ouvrages de référence pour présenter l'IDM (Combemale, 09)(Jézéquel et al., 12), ce document se concentrera davantage sur l'application des concepts présentés que sur leur définitions et présentations détaillées. Ce livrable dépasse donc le cadre initial de la tâche 2 consacrée à l'état de l'art. Il s'agit davantage d'un support de présentation argumenté, orienté IDM, du positionnement du projet en terme de cadre de travail et de contributions envisagées.

2 Concepts élémentaires et applications dans rap!iT

2"1 #odèle

$éfinitions

Un modèle est une abstraction d'un système, modélisé sous la forme d'un ensemble de faits construits dans une intention particulière. Un modèle doit pouvoir être utilisé pour répondre à des questions sur le système modélisé. Un modèle ***représente*** *donc un système selon un certain point de vue, à un niveau d'abstraction facilitant par exemple la validation ou la conception de cet aspect particulier du système.*

Principe de substituabilité : Un modèle doit être suffisant et nécessaire pour permettre de répondre à certaines questions en lieu et place du système qu'il est censé représenter, exactement de la même façon que le système aurait répondu lui-même.

%pplication au projet

Le s&stème que l'on cherche à représenter est le cours a mettre en place sur la plate' forme. En effet, nous ne cherchons pas à modéliser la plateforme de formation à distance mais bien uniquement le cours que l'on peut concevoir directement en exploitant les interfaces de conception de la plateforme.

Le modèle qui représente ce cours est celui produit par les éditeurs graphiques : il s'agit donc du cours ***que l'on souhaite*** mettre en place sur la plateforme. Mais il existe à différents ni(eaux d)abstraction . En effet, le projet considère les éditeurs et les langages visuels de conception comme *externes* à la plateforme : ils devront donc proposer un service *d'export* compatible avec une *API* d'import/export que le projet produira afin d'étendre la plateforme par un service de communication avec l'extérieur. Vis-à-vis de cette API, le modèle est donc une représentation d)un cours * créer+mettre * jour sur la plateforme. Les éditeurs exporteront de tels modèles mais pourront manipuler leurs propres modèles internes. Ces derniers seront donc des représentations plus abstraites du cours à mettre en place que les modèles qui seront

interprétés par les API. Ainsi on peut considérer que les modèles des éditeurs graphiques seront des représentation des modèles manipulables par l'API (jouant alors le rôle de système, eux-mêmes représentant le cours à mettre en place sur la plateforme cible).

D'un point de vue vocabulaire, nous nommerons le modèle interne des éditeurs les scénarios pédagogiques (pour la plate-forme), tandis que les modèles exportés seront les scénarios pour la plate'forme . Cette terminologie met en avant le caractère davantage pédagogique des premiers par rapport aux seconds. Cela ne signifie pas qu'il n'y a pas de pédagogie dans les seconds, ni sur la plateforme, mais que l'expressivité des scénarios spécifiés dans les éditeurs couvrira davantage le point de vue pédagogique que les autres. En effet, un modèle est, par définition, une abstraction d'un système pour un point de vue particulier : il doit ,tre suffisant pour répondre aux questions selon ce point de (ue- en lieu et place du s&stème qu)il représente.

Ainsi le scénario de la plateforme sera manipulé par l'API : il devra donc permettre * l)%.l de conna/tre en quoi consiste le cours * créer /mettre à jour sur la plateforme. Ce point de vue nécessite donc un niveau d'abstraction faible de la plateforme : une abstraction de l'implémentation concrète des données dans la plateforme (souvent dans une base de données relationnelle) sera suffisant.

Enfin, le scénario pédagogique sera manipulé par l'enseignant-concepteur à travers les éditeurs graphiques : le niveau d'abstraction de la plateforme est beaucoup plus important. Il s'agit en effet pour ce modèle d'être capable de guider- aider et formaliser ce que le concepteur sou!aite mettre en 0u(re. Il s'agit autant de l'aider à concevoir (réfléchir, ajuster, expérimenter...) son cours que de le formaliser.

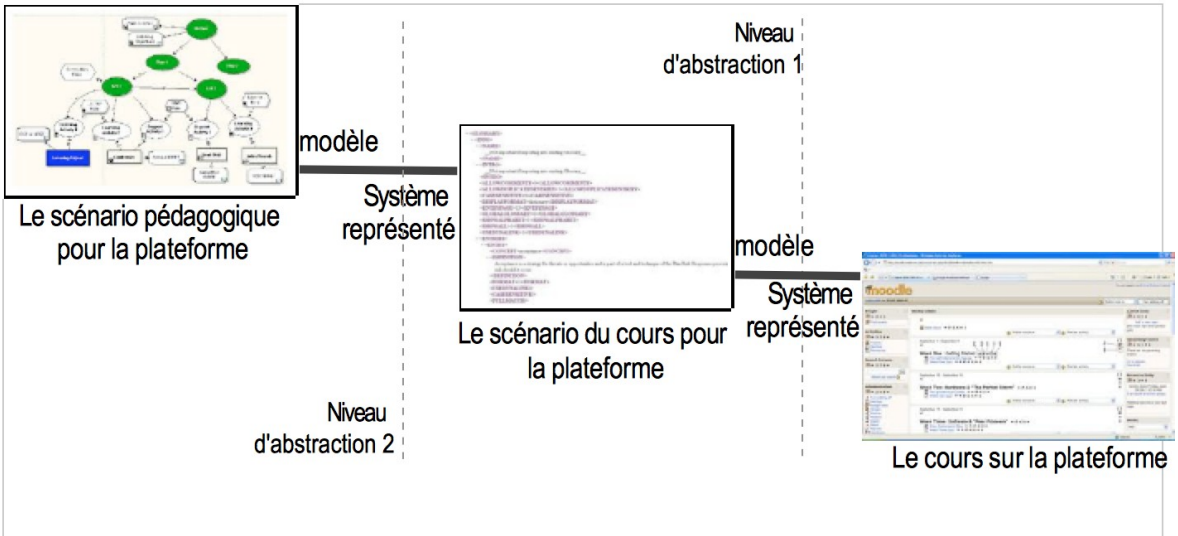

Illustration 1: modèles et systèmes pour les cours sur la plateforme

2"2 #éta'modèle

$éfinitions

La notion de modèle dans l'IDM fait explicitement référence à la notion de langage bien défini. En effet, pour qu'un modèle soit productif, il doit pouvoir être manipulé par une machine. Le langage dans lequel ce modèle est exprimé doit donc être clairement défini. De manière naturelle, la définition d'un langage de modélisation a pris la forme d'un modèle, appelé métamodèle.

Un métamodèle est un modèle qui définit le langage d'expression d'un modèle [46], c.-à-d. le langage de modélisation.

La notion de métamodèle conduit à l'identification d'une seconde relation, liant le modèle et le langage utilisé pour le construire, appelée conformeA.

%pplication au projet

Dans notre cas, chacun de nos deux types de modèles doit donc avoir un méta-modèle garantissant que les modèles qui lui seront conformes seront productifs et manipulables par la machine (l'éditeur comme l'API). Les modèles étant spécifiques * une plateforme de formation donnée, il en sera de même pour les méta-modèles correspondants.

Par la suite, nous considérons la plateforme Moodle à des fins d'illustration concrète sur un exemple de plateforme. Toutefois, nos arguments, schémas et figures, seront valables pour d'autres plateformes.

Le méta-modèle du premier niveau d'abstraction capitalise le domaine de connaissances de ce qui est possible de concevoir avec la plateforme Moodle : le méta'modèle de conception pédagogique de #oodle. Ce méta-modèle est unique au sens du projet car il sera le résultat de l'application d'un processus dédié à l'explicitation de ce métier de conception et à sa formalisation sous forme de méta-modèle. Toutefois, si l'on considère à juste titre que toute modélisation est subjective, de même que le processus que nous proposons, alors ce méta-modèle peut varier d'une communauté à l'autre à moins qu'il soit définitivement imposé par les propriétaires/créateurs de la plateforme.

Le second cristallise le domaine de conception pédagogique que l'on souhaite offrir aux concepteurs de cours pour Moodle. Il peut en exister *plusieurs* pour ce but. Il s'agit donc d' un méta'modèle de conception pédagogique pour #oodle . Dans le cadre du projet GraphiT nous allons proposé une approche d'abstraction qui permettra d'élaborer ce méta-modèle sur la base de celui de la plateforme.

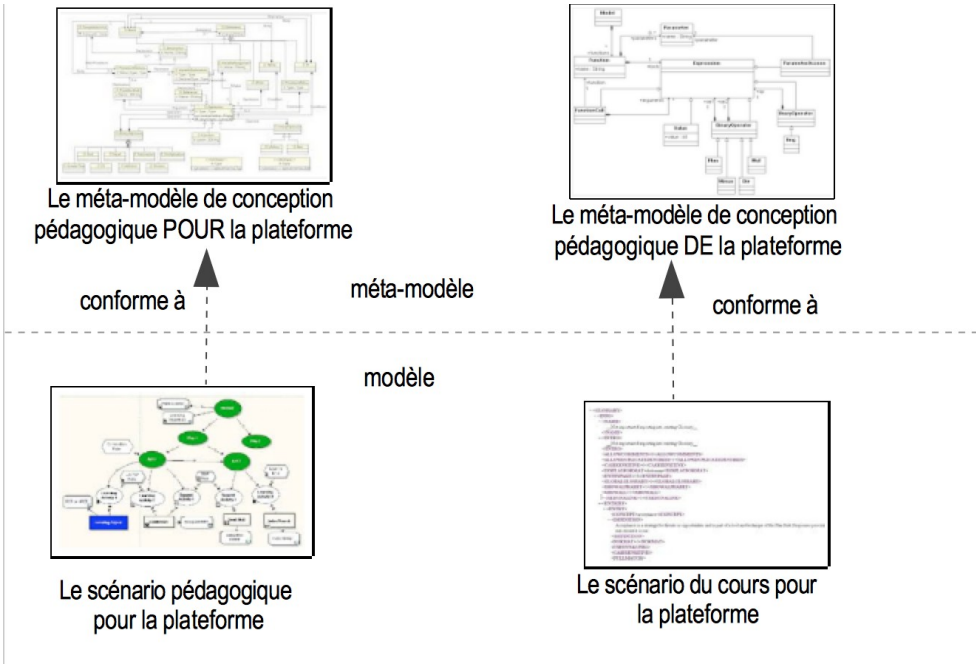

Illustration 2: Les différents méta-modèles de conception considérés dans le projet

2"1 #éta'méta'modèles et p&ramide de modélisation de l)O#

$éfinitions

Afin de contrôler et d'éviter l'émergence de méta-modèles incompatibles, le *Model Driven Architecture* (MDA) de l'OMG [1] a, entre autres, proposé un langage de définition de méta-modèle sous le forme d'un modèle : le méta-méta-modèle MOF (*Meta-Object Facility*). Aussi, pour limiter le nombre de niveaux d'abstraction, ce méta-méta-modèle a une propriété de *métacircularité* : il se décrit lui-même.

Un métamétamodèle est un modèle qui décrit un langage de méta-modélisation, c.-à-d. les éléments de modélisation nécessaires à la définition des langages de modélisation. Il a de plus la capacité de se décrire lui-même.

C'est sur ces principes que se base l'organisation de la modélisation de l'OMG généralement décrite sous une forme pyramidale. Le monde réel est représenté à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. Les méta-modèles permettant la définition de ces modèles (p. ex. UML) constituent le niveau M2. Enfin, le métamétamodèle, unique et métacirculaire, est représenté au sommet de la pyramide (niveau M3).

1 L'OMG (Object Management Group) est un consortium international d'industries et d'organisations académiques. L'OMG propose des standards et approches tels que UML, Corba, MOF, CWM, OCL, l'approche MDA, etc.

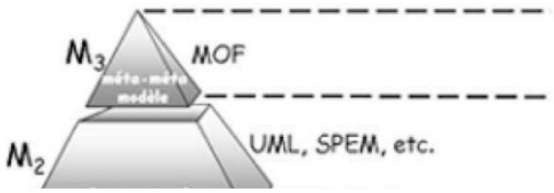

Illustration 3: La pyramide de modélisation de l'OMG

%pplication au projet

Nos deux méta-modèles devront être conformes au MOF (tout du moins conformes à une solution outillée existante équivalente au MOF, cf. sections suivantes).

Le positionnement de nos modèles et méta-modèles selon la pyramide de modélisation de l'OMG est assez trivial : les modèles (scénario pédagogique et scénario de cours) sont au niveau M1, tandis que les méta-modèles correspondants sont au niveau M2 (le MOF étant au niveau M3). Le niveau M0 correspondant au réel, on peut considérer y trouver l'espace-cours « souhaité » (avant utilisation de l'API) , comme celui « obtenu » (après utilisation de l'API d'import).

2"**2** **3**es différents modèles promulgués dans l)approc**!**e #**$%**

$éfinitions

Le MDA préconise l'élaboration de modèles :

– d'exigence (Computation Independent Model – CIM) dans lesquels aucune considération informatique n'apparait,

– d'analyse et de conception (Platform Independent Model – PIM),

– de code (Platform Specific Model – PSM).

L'objectif majeur du MDA est l'élaboration de modèles pérennes (PIM), indépendants des détails techniques des plate-formes d'exécution (J2EE, .Net, PHP, etc.), afin de permettre la génération automatique de la totalité des modèles de code (PSM) et d'obtenir un gain significatif de productivité.

Le passage de PIM à PSM fait intervenir des mécanismes de transformation de modèle et un modèle de description de la plateforme (Platform Description Model – PDM).

Cette démarche s'organise donc selon un cycle de développement « en Y »propre au MDD (Model Driven Development)

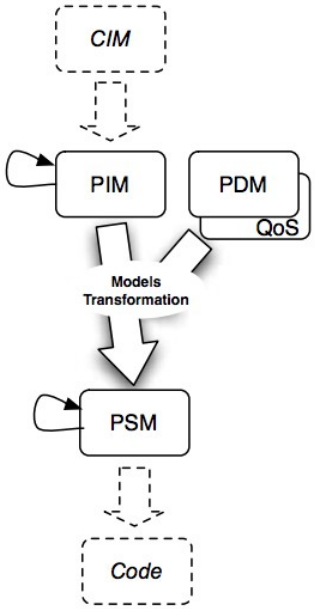

Illustration : !rocessus en " dans l'approc#e et processus M$%

%pplication au projet

La vision MDA de la scénarisation pédagogique vers les plateformes de formation à distance a déjà été considérée dans les premiers travaux croisant IDM et ingénierie pédagogique comme (Laforcade et al., 06) :

• les modèles PIM correspondent à des scénarios pédagogiques indépendants des plateformes de formation sur lesquels ils pourraient être déployés (indépendance en termes de fonctionnalités/ressources, de paradigme ou approche de conception que pourraient suivre des plateformes). Les PIM nécessitent toutefois des langages de scénarisation pour être formellement spécifiés ; les EML comme le standard IMS-LD sont de tels langages.

• les modèles PSM correspondent quant à eux à des scénarios spécifiques au métier de conception d'une plateforme de formation ;

• les modèles CIM finalement correspondent à des modèles indépendants des plateformes mais également indépendants des langages de scénarisation qui permettent de formaliser le scénario. Les CIM s'appuient sur des langages de scénarisation plus ou moins formels : le focus n'est pas en premier lieu sur l'exploitation informatique des modèles mais plutôt sur l'expressivité des modèles en relation avec des approches de conception souvent très spécifiques (par exemple un langage de conception CIM pour les situations-problèmes). Le CIM ne correspond pas nécessairement à un scénario pédagogique complet mais à à un modèle de conception préliminaire, selon un point de vue métier spécifique.

Cette première analogie scénarisation pédagogique / MDA est compatible avec l'approche processus du MDA consistant à modéliser d'abord le CIM (le modèle de conception pédagogique), puis à le retranscrire informatique sous forme de PIM (le scénario pédagogique formel), puis à le projeter vers une plateforme spécifique pour créer un PSM (le scénario au sens d'un modèle compatible avec la plateforme de formation, qui peut être exploité pour mettre en œuvre l'espace-

cours correspondant). Les premiers langages de scénarisation visuels allaient dans ce sens : ils permettaient de décrire des modèles CIM sans se préoccuper d'un formalisme (pour les modèles résultants) qui serait interprétable par la machine. Inversement les EML ont plutôt cherché à proposer une formalisation informatique exploitable : IMS-LD par exemple s'appuie sur une formalisation XML en 3 niveaux d'informations. En complément, les EML revendiquent également une forme d'indépendance vis-à-vis d'un métier de conception spécifique, en opposition aux langages de scénarisation spécifiques de l'époque. Cette indépendance relative était limitée à un cadre précis relevant d'un paradigme de conception sous-jacent : l'approche consensuelle des EML consistait à considérer la scénarisation selon une approche socio-constructiviste, plaçant *l'activité* des acteurs de l'apprentissage comme artefact de premier ordre pour l'élaboration des scénarios. Cette approche cherchait à s'opposer surtout aux approches centrées ressources (Learning Object comme SCORM) ou documentaire (orientée hypermédia) de l'époque.

Il s'agit de l'approche standard d'application/analogie du processus MDA entre modèles CIM-PIM-PSM dans le cadre de la scénarisation pédagogique vers des plateformes de formation. Le PIM est alors le point d'entrée dans la formalisation pour une exploitation informatique des scénarios. Les standards tels IMS-LD semblent être de bons candidats comme langage de PIM afin d'assurer l'interopérabilité des modèles de conception CIM pour tous types de plateformes. Dès lors, le PIM apparaît comme un langage pivot. De nombreux travaux en conception pédagogique vont alors chercher à s'appuyer sur IMS-LD pour tendre vers l'opérationnalisation de leurs modèles de conception (CIM vers PIM). Malheureusement, les pertes sémantiques lors des transformations CIM=>PIM, plus la difficulté que rencontrera IMS-LD à s'intégrer aux plateformes feront que cette approche n'aboutira pas au succès attendu.

Les VIDL (*Visual Instructional Design Language*) sont des langages de modélisation pédagogique ayant pour point commun une notation visuelle. Leur essor est arrivé après l'échec des EML standards comme IMS-LD qui cumulaient le verrou de l'opérationnalisation et celui de l'expressivité pédagogique (métaphore théâtrale pour IMS-LD par exemple). De nombreux VIDL ont été proposés (coUML, CPM, LDL, poEML, MOT+..., cf. livrable D2-6 correspondant). Ces VIDL visent en premier à traiter du verrou de l'expressivité pédagogique du point de vue des concepteurs : tous les VIDL sont alors spécifiques à des approches et/ou courant et/ou pratiques pédagogiques spécifiques. Les modèles de conception que ces VIDLs permettent d'exprimer correspondent directement au niveau CIM du MDA. Toutefois devant la multitude de ces VIDL des critères de classification sont apparus afin de pouvoir mieux catégoriser et comparer ces VIDL. Deux critères sont pertinents dans notre analogie avec le MDA : élaboration (niveau de détails des modèles de conception produits entre : *conceptuel*, *specification*, *implementation*) et formalisation (ensemble fermé de concepts et règles versus esquisses/dialogues/... : *formal*, *informal*). En fonction de la combinaison de ces deux critères et selon si l'on considère qu'un niveau de formalisation « formel » permet la manipulation informatique du modèle alors on peut faire correspondre le CIM à un modèle produit avec un VIDL de niveau conceptuel utilisant un langage informel, le PIM serait alors produit avec un VIDL de niveau spécification et proposant un langage formel, et le PSM serait produit avec un VIDL de niveau implémentation avec un langage formel. Remarquons que l'indépendance vis-à-vis de la plateforme de formation cible ne rentre pas en ligne de compte dans cette correspondance. Si l'on souhaite la prendre en compte il faut alors préciser pour les CIM et PIM que les VIDL associés n'est pas spécifique à une plateforme de formation.

Dans le cadre particulier du projet, nous considérons 2 niveaux d'abstraction pour le scénario pédagogique, les 2 étant dépendants d'une plateforme de formation donnée : le premier niveau est celui du scénario pédagogique à mettre en place sur la plateforme (SC1) mais dont le formalisme propose des primitives de conception de plus haut niveau que celles de la plateforme, et le second

est celui du modèle/scénario équivalent au précédent mais uniquement exprimé en termes de primitives de conception explicitées à partir de l'étude de la plateforme (SC2). L'utilisateur produira SC1 à travers un éditeur dédié, charge à l'éditeur d'exporter ce modèle dans un format compatible avec SC2 (transformation de modèle). SC2 sera importé dans la plateforme à travers l'API que nous nous proposons de développer et permettra la création de l'espace-cours correspondant. Toutefois dans le projet le langage de scénarisation pour spécifier des SC1 devra être dérivé de celui permettant de formaliser les SC2. Il n'y a donc pas d'indépendance à la plateforme de formation comme dans l'approche classique appliquée du MDA. Notre approche pourrait être comparée à la production de PSM ayant des niveaux d'abstraction de la plateforme cible différent.

1 3angages et applications dans rap!iT

1"1 #éta'modélisation

$éfinitions

La métamodélisation est l'activité consistant à définir le métamodèle d'un langage de modélisation. Elle vise donc à bien modéliser un langage, qui joue alors le rôle de système à modéliser.

%pplication au projet

La méta-modélisation est essentielle dans le cadre du projet. Cette activité est présente à différentes niveaux :

- Elle fait partie intégrante du processus d'explicitation du métier de conception des plateformes que le projet vise à proposer ; l'étape finale du processus en effet sera de formaliser le résultat du processus sous la forme d'un méta-modèle décrivant le langage de conception de la plateforme.
- Elle est également partie intégrante du processus d'abstraction du métier de conception de la plateforme qui nous permettra de proposer des langages de scénarisation abstraits mais dédié à une plateforme donnée.
- Elle devrait intervenir également aux niveaux des techniques de transformations et de composition de modèles que le projet devra étudier afin d'assurer la formalisation des correspondances entre les 2 types de modèles.

1"2 3angages

$éfinitions

*Un **langage** (L) est défini selon le tuple {S,Sem} où S est sa syntaxe et Sem sa sémantique.*

Dans le contexte de l'IDM, la syntaxe abstraite est placée au cœur de la description d'un langage de modélisation. Elle est donc généralement décrite en premier et sert de base pour définir la syntaxe concrète. La définition de la syntaxe concrète consiste alors à définir des décorations (textuelles ou graphiques) et à définir un lien entre les constructions de la syntaxe

abstraite et leurs décorations de la syntaxe concrète (Mac). Ce changement de sens du lien par rapport aux langages de programmation permet d'envisager de définir plusieurs syntaxes concrètes (M∗ac) pour une même syntaxe abstraite et donc d'avoir plusieurs représentations d'un même modèle. Le langage peut alors être manipulé avec différents formalismes mais avec les mêmes constructions et la même représentation abstraite. D'autre part, dans le cadre des langages de modélisation, la sémantique est exprimée à partir des constructions de la syntaxe abstraite par un lien vers un domaine sémantique (Mas).

*Un **langage de modélisation** (Lm) est défini selon le tuple {AS,CS∗ ,M∗ ac,SD,Mas} où AS est la syntaxe abstraite, CS∗ est la (les) syntaxe(s) concrète(s), M∗ ac est l'ensemble des mappings de la syntaxe abstraite vers la (les) syntaxe(s) concrète(s), SD est le domaine sémantique et Mas est le mapping de la syntaxe abstraite vers le domaine sémantique.*

L'IDM favorise la définition de langages de modélisation dédiés à un domaine particulier (*Domain Specific Modeling Languages* – DSML) offrant ainsi aux utilisateurs des concepts propres à leur métier et dont ils ont la maîtrise. Ces langages sont généralement de petite taille et doivent être facilement manipulables, transformables, combinables, etc. Selon ces principes, la définition d'un système complexe fait généralement appel à l'utilisation de plusieurs DSML ayant des relations entre eux, restreignant ainsi l'ensemble des combinaisons valides des modèles conformes à ces différents métamodèles (c.-à-d. construits à l'aide de ces différents DSML).

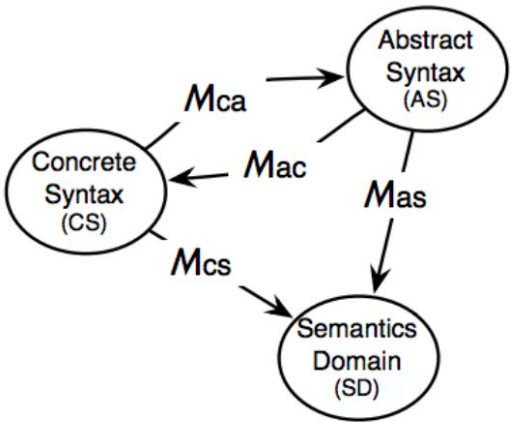

Illustration &: 'omposantes d(un lan)a)e *de pro)rammation et de modélisation+

%pplication au projet

Dans le cadre du projet, nous visons des langages de modélisation visuels d'au moins 2 sortes (pour chaque plateforme qui sera étudiée) :

- un Lm de premier niveau (appelons le Lm1) s'appuyant directement sur le méta-modèle de la plateforme (qui aura été explicité et formalisé) afin de proposer une conception de scénario de cours en relation directe avec le métier de conception de la plateforme ;

- un Lm de second niveau (Lm2) *dérivé*[2] du méta-modèle de la plateforme afin de proposer une conception de scénario pédagogique plus expressive en relation *indirecte* avec le métier de conception de la plateforme.

On a donc Lm1 = *{AS,CS ,Mac, SD, Mas}* où AS est le méta-modèle qui sera formalisé pour une plateforme donnée, où CS est une syntaxe concrète graphique orientée diagrammes.

De même, Lm2 = *{AS,CS ,Mac, SD, Mas}* mais avec cette fois-ci AS qui sera un nouveau méta-modèle dérivé de celui de la plateforme, et CS étant une autre syntaxe concrète diagrammatique.

Lm1 et Lm2 sont 2 DSML qui permettent d'adresser la scénarisation pédagogique de situations d'apprentissage pour une même plateforme de formation mais à des niveaux d'abstraction différents.

1"1 4&ntaxes abstraites

$éfinitions

La syntaxe abstraite (AS) d'un langage de modélisation exprime, de manière structurelle, l'ensemble de ses concepts et leurs relations. Les langages de métamodélisation tels que le standard MOF de l'OMG, offrent les concepts et les relations élémentaires qui permettent de décrire un métamodèle représentant la s&ntaxe abstraite d'un langage de modélisation.

Pour définir cette syntaxe, nous disposons à ce jour de nombreux environnements et langages de métamodélisation : Eclipse-EMF/Ecore, GME/MetaGME, AMMA/KM3, XMF-Mosaic/Xcore ou Kermeta. Tous ces langages reposent toutefois sur les mêmes constructions élémentaires. S'inspirant de l'approche orientée objet, les langages de métamodélisation objet offre le concept de classe (*Class*) pour définir les concepts d'un DSML. Une classe est composée de propriétés (*Property*) qui la caractérisent. Une propriété est appelée référence lorsqu'elle est typée (*TypedElement*) par une autre classe, et attribut lorsqu'elle est typée par un type de donnée (p. ex. booléen, chaîne de caractère et entier).

La représentation graphique offerte par les langages de métamodélisation ne permet pas de capturer formellement l'ensemble des propriétés du langage (c.-à-d. les context conditions). Dans le cadre d'un langage de modélisation, cette seconde utilisation est exprimée par le biais de règles de bonne formation (***Well-Formed Rules*** – WFR), au niveau du métamodèle. Ces règles devront être respectées par les modèles conformes à ce métamodèle.

Pour exprimer ces règles, l'OMG préconise d'utiliser OCL (*Object Constraint Language*). Appliqué au niveau du métamodèle, il permet d'ajouter des propriétés, principalement structurelles, qui n'ont pas pu être capturées par les concepts fournis par le métamétamodèle. Il s'agit donc d'un moyen de préciser la sémantique du métamodèle en limitant les modèles conformes.

%pplication au projet

Les langages de modélisation visés par le projet seront basés sur des syntaxes abstraites à base de méta-modèle. Nous utiliserons l'environnement de méta-modélisation Eclipse-EMF/Ecore. Si nécessaire nous utiliserons les contraintes OCL pour formaliser les règles de bonne formation complémentaires aux méta-modèles.

---

2 Cette dérivation est un des objets d'étude du projet : en quoi elle consiste, quelles méthodes/techniques adaptées, etc.

1"2 4&ntaxes concrètes

$éfinitions

Les syntaxes concrètes (CS) d'un langage fournissent à l'utilisateur un ou plusieurs formalismes, graphiques et/ou textuels, pour manipuler les concepts de la syntaxe abstraite et ainsi en créer des « instances ». Le modèle ainsi obtenu sera conforme à la structure définie par la syntaxe abstraite. La définition d'une syntaxe concrète consiste à définir un des mappings de M∗ ac, Mac : AS ↔CS, et permet ainsi « d'annoter »chaque construction du langage de modélisation définie dans la syntaxe abstraite par une (ou plusieurs) décoration(s) de la syntaxe concrète et pouvant être manipulée(s) par l'utilisateur du langage. La définition du modèle d'une syntaxe concrète est à ce jour bien maîtrisée et outillée. Il existe en effet de nombreux projets qui s'y consacrent, principalement basés sur EMF (*Eclipse Modeling Framework*) : GMF (*Generic Modeling Framework*), TOPCASED, Merlin Generator, GEMS, TIGER, SIRIUS, etc. Si ces derniers sont principalement graphiques, des projets récents permettent de définir des modèles de syntaxe concrète textuelle. Nous citons par exemple les travaux des équipes INRIA Triskell (Sintaks) et ATLAS (TCS) qui proposent des générateurs automatiques d'éditeurs pour des syntaxes concrètes textuelles. Ces approches génératives, en plus de leurs qualités de généricité, permettent de normaliser la construction des syntaxes concrètes.

%pplication au projet

Les langages de modélisation visés par le projet seront basés sur des syntaxes concrètes graphiques dérivée des méta-modèles (une seule syntaxe concrète par langage bien que plusieurs soient possibles). Nous utiliserons les environnements de méta-modélisation #F (pour Lm1) et **4I5I64** (pour Lm2). SIRIUS est un framework similaire à GMF (se base sur des méta-modèles au format ecore réalisés avec EMF) mais il se distingue dans son approche interprété en opposition à GMF qui nécessite de générer du code java pour les éditeurs graphiques et la manipulation des modèles. Le framework GMF ayant déjà été utilisé dans d'autres projets de l'équipe impliquée dans le projet GraphiT, il sera utilisé pour Lm1 le langage de modélisation graphique construit directement par dessus le méta-modèle de conception pédagogique que nous aurons capturé pour une plateforme de formation donnée. Le framework SIRIUS sera plus pertinent par son approche interprété pour explorer par prototypage la conception des langages de scénarisation plus abstraits.

1"7 4émantique

$éfinitions

La seule volonté de définir des modèles contemplatifs utilisés uniquement pour communiquer avec des constructions plus abstraites sur le système, ne demande pas une définition plus complète du langage de modélisation. En effet, les syntaxes abstraite et concrète sont suffisantes pour manipuler graphiquement (ou textuellement) un langage et réfléchir de manière plus abstraite à la construction du système. Dans ce contexte, ce sont les noms choisis pour nommer les concepts qui sont le support de la sémantique, et donc de la réflexion. Toutefois, la manipulation automatique des modèles par des outils (pour des simulations, vérifications, tests, générations, etc.) nécessite une formalisation de la sémantique jusque là implicite.

La sémantique d'un langage définit de manière précise et non ambiguë la signification des constructions de ce langage. Elle permet ainsi de donner un sens précis aux modèles construits à

partir de celui-ci. Une sémantique est dite formelle lorsqu'elle est exprimée dans un formalisme mathématique et permet de vérifier la cohérence et la complétude de cette définition.

Définir la sémantique d'un langage revient à définir le domaine sémantique et le mapping Mas entre la syntaxe abstraite et le domaine sémantique (AS ↔ SD). Le domaine sémantique définit l'ensemble des états atteignables par le système, et le mapping permet d'associer ces états aux éléments de la syntaxe abstraite. Dans le contexte de l'IDM, au même titre que les autres éléments d'un langage de modélisation, la définition du domaine sémantique et du mapping prend la forme de modèle.

%pplication au projet

A l'échelle du projet nous nous contenterons de définir la sémantique statique à travers les constructions de nos méta-modèles, les éventuelles règles de bonne formation complémentaires, et des informations supplémentaires en langage naturel.

1"8 Transformations de modèles

$éfinitions

Transformer un modèle *Ma* en un modèle *Mb*, que les métamodèles respectifs *MMa* et *MMb* soient identiques (transformation endogène) ou différents (transformation exogène), apparaît comme primordial (génération de code, refactoring, migration technologique, etc.). Aussi, l'approche MDA repose sur le principe de la création d'un modèle indépendant de toute plateforme (PIM) pouvant être raffiné en un ou plusieurs modèle(s) spécifique(s) à une plateforme (PSM).

Les méthodes de transformation sont là aussi indispensables pour c!anger de ni(eau d9abstraction (transformation (erticale), dans le cas du passage de PIM à PSM et inversement, ou pour rester au m,me ni(eau d9abstraction (transformation !ori:ontale) dans le cas de transformation PIM à PIM ou PSM à PSM. Ces différentes classes de transformation sont reprises dans la figure 6 en indiquant leurs cas d'utilisation.

Enfin, la transformation de modèle est également utilisée dans la définition des langages de modélisation pour établir les mappings et des traductions entre différents langages. Ces différentes classes de transformation sont résumées dans la figure 7.

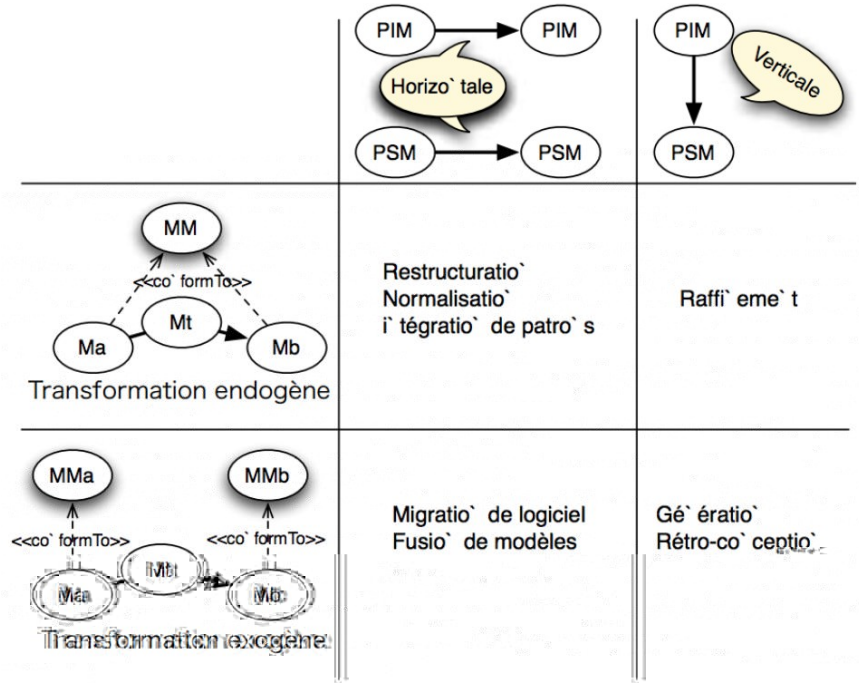

Illustration , : -ypes de transformations et leurs principales utilisations

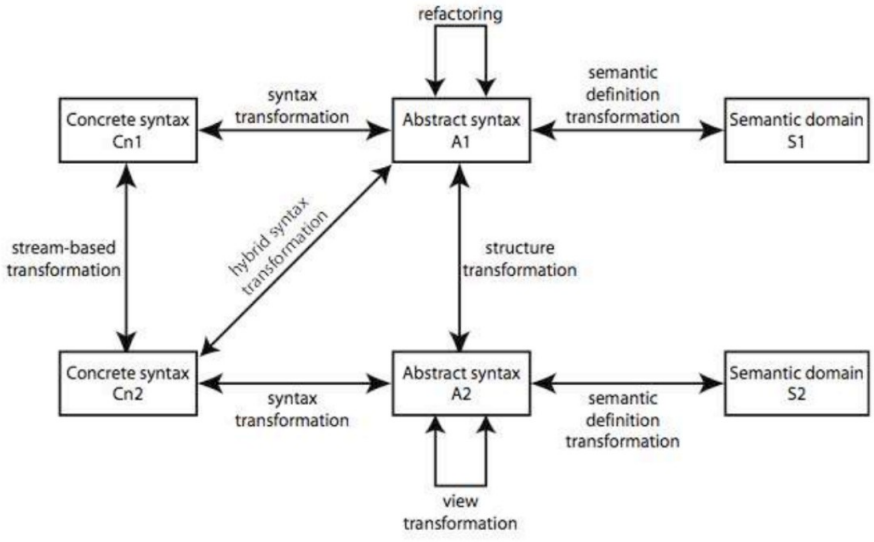

Illustration . : ' lasses de transformations dans la définition d'un $/ML

De nombreux langages sont à ce jour disponibles pour écrire des transformations de modèle. On retrouve d'abord les langages généralistes qui s'appuient directement sur la représentation abstraite du modèle. On citera par exemple l'API d'EMF qui, couplée au langage Java, permet de

manipuler un modèle sous la forme d'un graphe. Dans ce cas, c'est à la charge du programmeur de faire la recherche d'information dans le modèle, d'expliciter l'ordre d'application des règles, de gérer les éléments cibles construits, etc.

Afin d'abstraire la définition des transformations de modèle et rendre transparent les détails de mise en œuvre, l'idée a été de définir des DSML dédiés à la transformation de modèle. Cette approche repose alors sur la définition d'un métamodèle dédié à la transformation de modèle et des outils permettant d'exécuter les modèles de transformation. Par exemple une telle approche est suivie par l'outillage ATL (*ATLAS Transformation Language*). Il s'agit d'un langage hybride (déclaratif et impératif) qui permet de définir une transformation de modèle à modèle (appelée Module) sous la forme d'un ensemble de règle. Il permet également de définir des transformations de type modèle vers texte (appelée Query). Une transformation prend en entrée un ensemble de modèles (décrits à partir de métamodèles en Ecore ou en KM3).

Afin de donner un cadre normatif pour l'implantation des différents langages dédiés à la transformation de modèle, l'OMG a défini le standard QVT (Query/View/Transformation). Le méta-modèle de QVT est conforme à MOF et OCL est utilisé pour la navigation dans les modèles.

%pplications au projet

Pour rappel, dans le cadre du projet, nous avons prévu de concevoir et d'outiller les 2 langages Lm1 et Lm2 exploitant respectivement le métier de conception de la plateforme à 2 niveaux : le premier s'appuie directement sur le méta-modèle de la plateforme de formation, le second propose une abstraction supplémentaire afin de proposer aux futurs concepteurs des éléments de conception pédagogique moins proche de la sémantique et du paradigme implicite de conception de la plateforme. Toutefois, afin de rendre opérationnalisables sur la plateforme les modèles conformes à Lm2 ils devront être importés à travers l'API et donc devront être conformes au méta-modèle explicité et formalisé pour la plateforme.

Afin de retrouver cette conformité, nous envisageons qu'une première transformation de modèle sera nécessaire. Il s'agit d'une forme de transformation exogène (méta-modèles MM1 et MM2 différents de Lm1 et Lm2) et **(**erticale (différent niveaux d'abstraction) : transformation de génération. La figure suivante illustre ce cas et précise également l'outillage envisagé : ATL comme outil/méta-modèle de transformation.

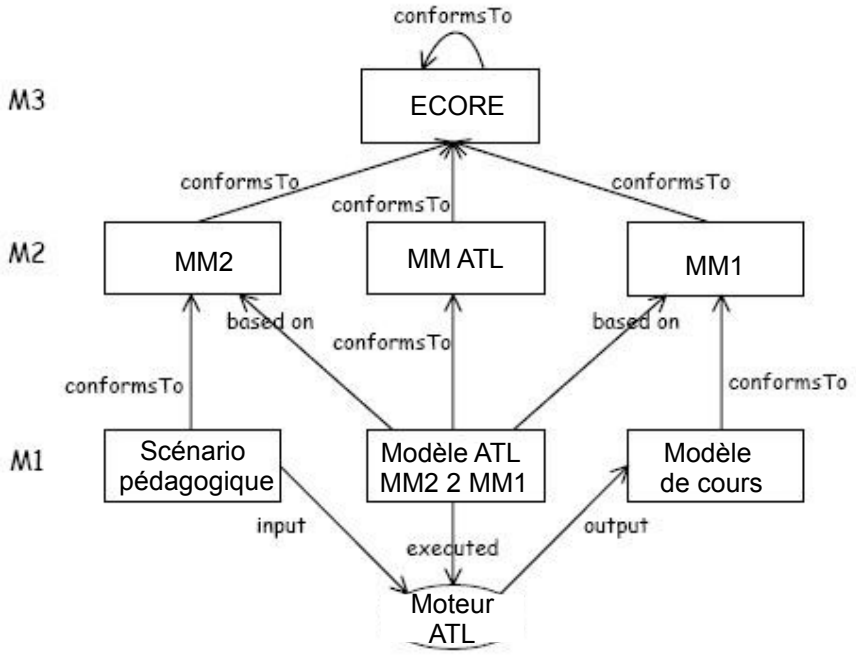

Illustration 0: -ransformation de modèle entre Lm2 et Lm1pour une plateforme de formation donnée

Aussi, nous savons d'ores et déjà, au vues des premières expérimentations du projet, que Lm2 devra permettre aux concepteurs de mixer des constructions abstraites de conception pédagogique avec des constructions plus élémentaires proposées par la plateforme de formation. Cette mixité sera parfois volontaire et explicitement réalisée par le concepteur, d'autres fois elle sera involontaire et explicitée par l'éditeur afin de guider et expliquer aux concepteurs que tels ou tels choix de conception seront réifiés avec tels ou tels caractéristiques et constructions sur la plateforme. Ce deuxième cas nécessite alors un mécanisme interne à Lm2 qui permettra de générer des éléments de modèles en fonction d'autres, *dynamiquement.* Il s'agit d'une deuxième forme de transformation endogène (tout cela devra être permis par la méta-modèle MM2 de Lm2 qui devra donc inclure le méta-modèle MM1 de Lm1 et l)étendre d'une manière spécifique que le projet va étudier) et (erticale (puisqu'entre 2 niveaux d'abstraction différents) : transformation de type raffinement. Cette seconde transformation fait l'objet d'études dans le projet afin de rechercher la meilleure solution technique pour la concevoir et la réaliser. La figure suivante schématise cet aspect : le scénario est enrichi (au niveau des activités pédagogiques par exemple), au *runtime*, d'informations détaillées sur les correspondances en terme d'outils/ressources et paramétrages spécifiques à réaliser sur la plateforme pour mettre en œuvre ces activités pédagogiques.

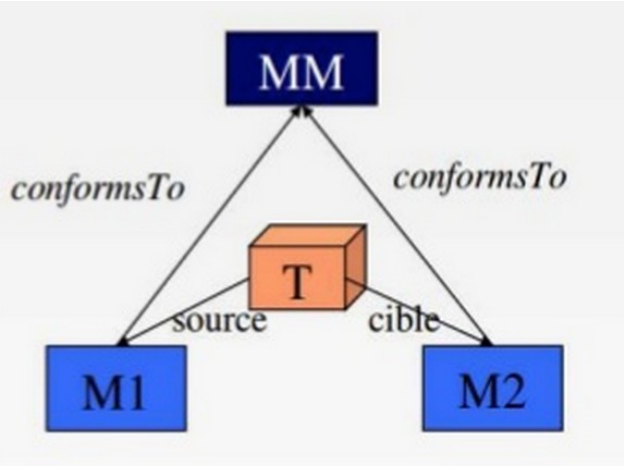

Illustration 1: Transformation de modèle interne et dynamique pour Lm2

La solution retenue est d'appliquer de façon exécutable, le tissage de modèle. A partir d'un modèle de correspondances (modèle de tissage noté WM), nous produisons via une transformation de haut niveau (*High Order Transformation*), un ensemble de transformations (MT) qui pourront être exécutées au runtime de l'éditeur afin d'enrichir le scénario pédagogique (PS). Ce fonctionnement est très semblable à celui du framework AMW (The AMW Project, 10), qui n'est aujourd'hui plus maintenu.

Techniquement, nous exploitons les langages du projet Eclipse Epsilon (The EPSILON project, 14). La HOT consiste en une transformation Model-to-text (M2T), écrite en EGL, qui produit le code source (en EOL) d'une ou plusieurs transformations Model-to-Model (M2M).

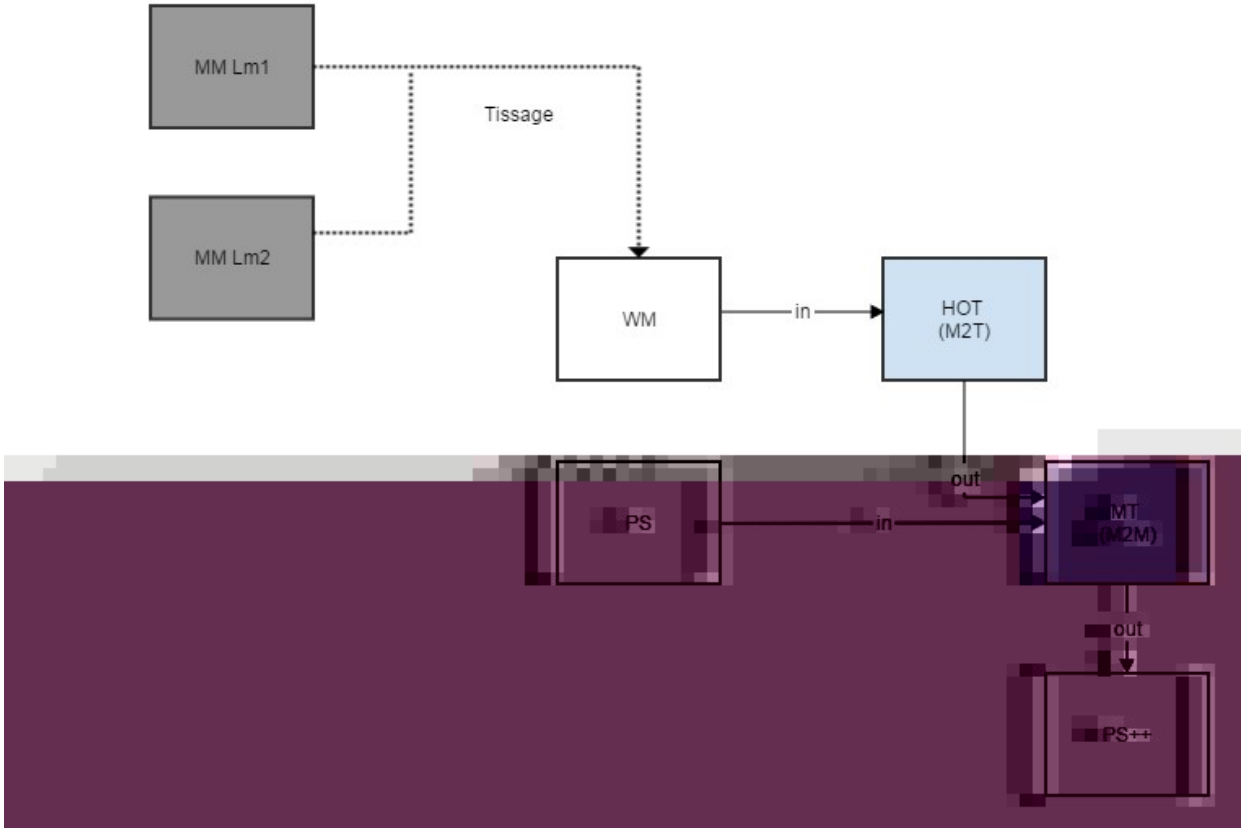

Illustration 13: Processus global d'enrichissement du scénario pédagogique

1"; Composition de modèles

$éfinitions

Model composition is an operation that combines two or more models into a single one.

Model composition in its simplest form refers to the mechanism of combining two models into a new one.

Ces deux définitions peuvent être traduites diagrammatiquement comme le schéma suivant. Conformément à celui-ci, on peut dire que la composition de modèle est un processus qui prend deux ou plusieurs modèles en entrée, les intègre au travers d'une opération de composition et produit un modèle composite en sortie.

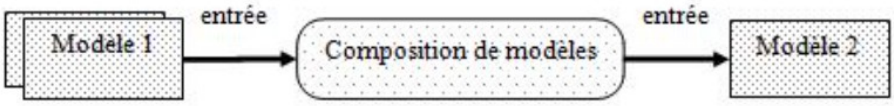

Illustration 11: *!* rocessus de composition de modèles

Classification des approc**!**es

1 Classification selon les caractéristiques des modèles en entrée

1.1 La quantité des modèles en entrée : deux ou plusieurs.

1.2 Le r**<**le des modèles composés : symétrique ou asymétrique.

1.3 Le t**&**pe des modèles composés : structurel ou comportemental.

1.4 **3)!**étérogénéité des modèles source : composition de même méta-modèle (endogène) ou différents méta-modèles (exogène).

2 Classification selon les caractéristiques des modèles en sortie

2.1 La quantité des modèles en sortie : un (le modèle composite) ou plusieurs (modèles d'entrée + composite).

3 Classification selon les caractéristiques de l'opération de composition

3.1 T**&**pe de composition : par *opérateurs* (fusion, le remplacement, l'union, le tissage etc.) ou par *relations* (l'association, l'agrégation, l'héritage, etc.).

Dans le premier type, la composition est réalisée en exécutant les opérateurs de composition sur les modèles sources; alors que dans le second type, les modèles source sont composés en utilisant les relations pour les connecter. La différence est que les opérateurs ne sont pas une partie du modèle final ; tandis que les relations font vraiment partie de ce modèle.

3.2 Les éléments de composition : ce sont les éléments supplémentaires participant aux compositions. Il y a deux axes de classification : le type et le formalisme de ces éléments.

- Le t**&**pe d)éléments de composition : spécifications de composition ou relations Correspondant aux types de composition identifiés ci-dessus, nous distinguons deux types d'élément de composition possibles : les spécifications de composition et les relations. Les relations expriment les liens de connexion entre les modèles. Elles sont à la fois les éléments de composition et les éléments du modèle final.

Les spécifications de composition sont les éléments de composition mais par contre elles ne sont pas des éléments du modèle final. Leur rôle est de modéliser les connaissances nécessaires au processus de composition. En général, un processus de composition a besoin de connaître :

- Quoi : spécifie quels éléments de modèle seront composés (e.g., une classe, un attribut, un méthode etc.) et leurs correspondances.
- Comment : spécifie des opérateurs de composition (e.g., correspondance (*match*),

remplacement (*replace*), redéfinition (*override*) etc.) qui vont être utilisés pour composer des éléments indiqués.

- Le langage de composition : les éléments de composition ont besoin de formalismes pour les exprimer. Ces formalismes sont très variés car chaque approche a ses propres éléments de composition. Ils peuvent être un langage de tissage, un méta-modèle des règles de composition, un profil UML pour la composition de modèles, etc. Malgré leur diversité, on peut généralement évaluer un formalisme de composition sur deux points : les abstractions de composition qu'il fournit et son extensibilité. Le premier point concerne ses primitives alors que le second point concerne sa capacité à permettre à l'utilisateur de définir ses propres abstractions de composition.
  - Abstractions de composition prédéfinies :

    De manière détaillée, il y a plusieurs variations autour ce point qui font les différences entre les approches. Par exemple, nous pouvons faire la classification selon les trois axes suivants :

    - La variété de types des abstractions : {correspondance, tissage, la fusion, remplacement, surcharge, ...}.

      Dans le cas de la composition par relations, ces abstractions sont les types de relation tels que l'association, l'héritage, l'agrégation.

      Dans le cas de la composition par opérateurs, ces abstractions sont les opérateurs de composition primitifs tels que la correspondance (*matching*), le tissage (*weaving*), la fusion (*merging*), le remplacement (*replace*), le surcharge (*override*) etc.

    - La variété de quantité des abstractions :

      Le nombre des abstractions est différent selon les approches.

    - La variété d'implémentation des abstractions :

      Plusieurs approches peuvent proposer un même type d'abstraction de composition mais l'implémenter différemment.

      Généralement, le processus de fusion de modèles peut être divisé en trois stades : avant, pendant et après la fusion. La description d'une telle fusion est comme suit :

      1. Avant la Fusion : il est probablement nécessaire de transformer (ou adapter) les modèles pour que la fusion puisse être réalisée sans conflit.
      2. Pendant la Fusion : il y a deux phases :

      – Phase de correspondances : établit les correspondances entre les éléments de modèles. La découverte des correspondances peut être automatique, manuelle ou semi-automatique.

      – Phase de fusion : intègre les éléments en correspondance. Il est probable d'avoir les conflits (e.g, les éléments correspondants n'ont pas le même type). La résolution de conflit est soit automatique (basée sur certaines stratégies de résolution prédéfinies) soit manuelle (e.g, faire apparaître un panel pour que l'utilisateur résolve le problème).

3. Après la Fusion : il est nécessaire de restaurer la cohérence du modèle final.

Cependant, dans la réalité, l'implémentation des approches ne respectent pas toujours ce processus. Certaines approches proposent explicitement l'adaptation mais certaines autres non. La manière de découvrir des correspondances peut-être faite automatiquement ou semi-automatiquement. De la même manière, la résolution de conflit peut varier entre manuel et automatique, même les stratégies de résolution sont différentes.

- Extensibilité du formalisme : extensible ou non extensible.

Les abstractions prédéfinies avec les sémantiques prédéfinies permettent la réalisation d'un grand nombre de scénarios de composition. Cependant, dans le cas général, ceci ne veut pas dire que la composition se limite à ces actions. L'utilisateur a probablement besoin de définir ses propres sémantiques de composition. Ceci entraîne la nécessité d'extension du formalisme de composition par la définition de nouvelles abstractions ou la redéfinition de la sémantique d'une abstraction prédéfinie.

Nous distinguons ainsi les formalismes extensibles et non extensibles. Le premier type est plus flexible que le deuxième. Il permet de réaliser les compositions de manière particulière. Dans le cas extensible, nous distinguons aussi deux manières d'extension du langage : la définition de nouvelles abstractions ou la redéfinition de la sémantique d'une abstraction prédéfinie.

3.3 La manière de créer un modèle composite : l'exécution des opérateurs ou l'établissement des relations.

Correspondant aux deux types de composition par opérateur et par relations, il y a deux manières de créer un modèle composite : soit par l'exécution des opérateurs sur les modèles sources ; soit par l'établissement des relations entre eux.

4 %utres caractéristiques de classification

4.1 **3)**effet de composition : structure de modèle transformée ou préservée. Les compositions peuvent transformer la structure des modèles source ou la préserver.

- Composition poids lourd : La composition transforme la structure des modèles composés (e.g., la fusion des modèles)
- Composition poids léger : La composition ne transforme pas la structure des modèles composés (e.g., établir des relations entre les modèles)

4.2 **3)**orientation de composition : Composition orientée transformation ou composition pure.

Ceci est assez lié à la caractéristique de l'effet de composition mentionné ci-dessus. Dans le cas de tissage, ou de fusion où il y a une transformation effectuée sur les éléments sources pour produire les éléments cibles, la composition est dite orienté transformation, alors que dans le cas d'établissement des relations, la composition est dite pure composition.

4.3 Le domaine de rec**!**erc**!**e : actuellement il y a trois domaines majeurs qui travaillent activement sur la composition de modèles :

1. Modélisation orientée aspect (AOM) : L'originalité de AOM est l'application du principe

de séparation de préoccupation à la modélisation. L'opération de composition centrale de AOM est le tissage. Il consiste à composer les modèles d'aspects à un modèle de base. La relation entre le modèle d'aspect et le modèles de base est relatif. Un modèle peut être un aspect et une base à la fois. De ce fait, deux types de tissage ont été identifiés : aspect avec base (asymétrique) et base avec base (symétrique).

2. Gestion de modèle : Ce domaine émerge dans le contexte IDM. Il s'intéresse aux plateformes IDM fournissant les opérateurs génériques de manipulation de modèles tels que la fusion, la comparaison, la différence, la génération etc. En ce qui concerne la composition, ces opérateurs peuvent divisés en trois groupes :
   - Pour la découverte des correspondances : tels que *match*, *relate*, *compare.*
   - Pour l'intégration de modèles : tels que *merge*, *compose*, *weaving.*
   - Pour la liaison de modèles, i.e. relier les modèles sans changer leur structure : *sewing.*
3. Méta-modélisation : Les approches de méta-modélisation permettent la définition des méta-modèles. Ces approches peuvent posséder eux même les mécanismes de composition permettant de composer les méta-modèles. Comme la relation entre modèle/méta-modèle est relative, il est possible de croire que si ces mécanismes sont applicables au niveau méta-modèle, leur principe devraient être applicables également au niveau de modèle.

   De plus, lorsqu'une approche permet la composition de méta-modèles, il est important aussi d'étudier si l'approche est capable ou non de composer les modèles créés par ces méta-modèles.

%pplications au projet

Dans le cadre IDM de notre projet, différents verrous techniques sont à étudier. Le premier concerne l)abstraction du méta-modèle de conception de la plateforme (tâche 5) quand ce dernier aura été formalisé (tâche 4). Cette abstraction devrait aboutir à la proposition de blocs de conception sémantiquement plus « distant » vis-à-vis de la sémantique plus technique, bas niveau et orientée mise en œuvre, de la plateforme. Ces blocs de haut niveau seront potentiellement à composer statiquement avec d'autres éléments de conception plus courants pour les langages de conception (par exemples structures d'activités, le concept de session, de rôles, d'objectifs...). La principale composition concernera ces éléments abstraits avec ceux du méta-modèle de la plateforme de manière à permettre aux concepteurs de mixer éléments de haut et bas niveau dans leurs scénarios. En effet, l'orientation actuelle du projet (après le travail exploratoire qui a été mené) est de permettre aux concepteurs de mélanger des éléments de conception qui ne font pas référence directe (exemple : « faire un brainstorming », « répondre à un sondage »...) à des éléments de la plateforme Moodle, avec d'autres éléments qui y font référence (« chat », « wiki »,...).

Le second verrou concerne la spécification de règles de correspondances entre éléments de haut niveau et éléments de bas niveau. Il est nécessaire de composer un modèle de correspondances entre ces éléments afin de produire via le processus illustré dans la figure X les transformations qui seront exécutées dynamiquement au runtime (lors de la conception des scénarios).

La figure suivante illustre les 2 verrous.

- Composition 1 (MM Lm2) = éléments récurrents + éléments conception abstraits + MM Lm1
- Composition 2 : Tissage entre éléments récurrents et éléments du métamodèle de la plateforme.

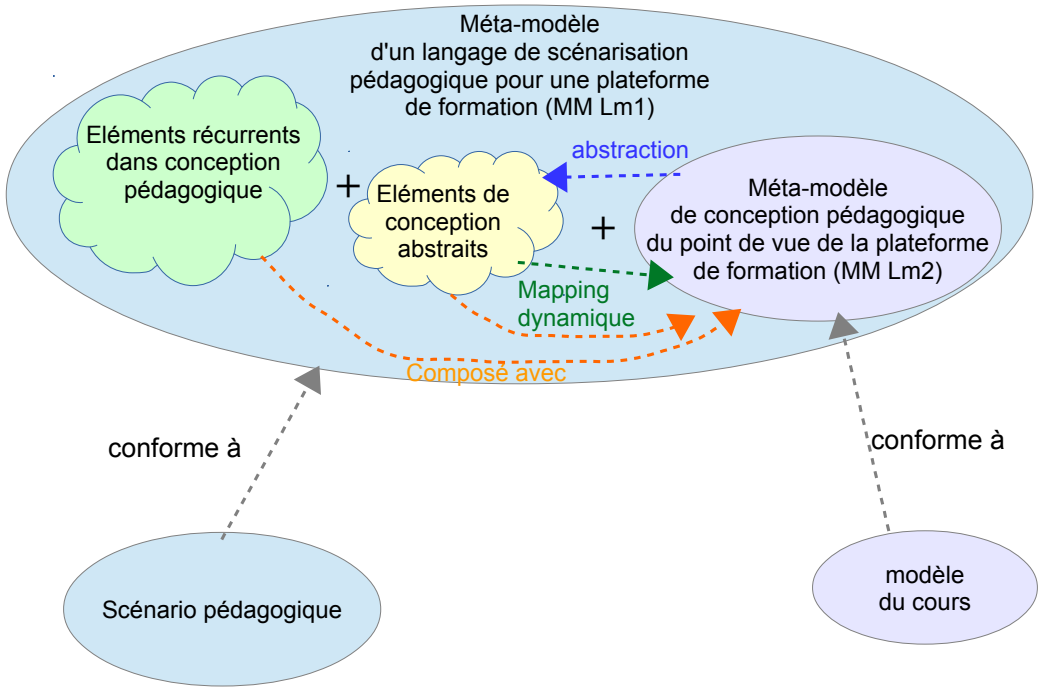

Illustration 12: /c#ématisation des 5errous I$M de composition de modèles du projet

La première composition n'est pertinente qu'à niveau méta-modèle : les scénarios conformes à Lm2 seront composés d'instances des différentes parties du méta-modèle de Lm2. Il n'y aura pas de besoin de composer des modèles uniquement conformes à ces sous-parties. Il s'agit donc d'une composition particulière. Vis-à-vis des critères de classification de la composition de modèles ce premier cas a les caractéristiques suivantes :

Modèles en entrée	*quantité*	plusieurs	
	rôle	symétrique	
	type	structurel	
	hétérogénéité	exogène	
Sortie	*quantité*	un	
Caractéristiques de l'opération de composition	*type*	relations	
	éléments de conception	*type*	relations
		langage	type EMF
	manière	Établissement de relations	
Autres	*effets*	Composition poids léger	

	orientation	Composition pure
	domaine	Méta-modélisation

La deuxième composition consiste à produire manuellement (en suivant cependant une méthode spécifique) un modèle de correspondances entre les types d'éléments de conception abstraits et les types d'éléments de conception issus de la plateforme. Concrètement, il s'agit de produire un modèle de tissage qui relie chaque classe issue du métamodèle MM Lm1 à une ou plusieurs classe du métamodèle MM Lm2. Le modèle de tissage en question sera lui conforme à un métamodèle que nous avons spécifié.

Modèles en entrée	*quantité*	plusieurs	
	rôle	symétrique	
	type	structurel	
	hétérogénéité	exogène	
Sortie	*quantité*	un	
Caractéristiques de l'opération de composition	*type*	relations	
	éléments de conception	*type*	relations
		langage	type EMF
	manière	Établissement de relations	
Autres	*effets*	Composition poids léger	
	orientation	Composition pure	
	domaine	Tissage de modèle	

2 Conclusion

L'IDM et le DSM offrent un cadre théorique et outillée riche pour la résolution de problèmes d'ordre scénarisation pédagogique. Dans le contexte du projet GraphiT ce cadre va nous permettre d'aborder et de traiter les différents problématiques scientifiques par la recherche de solutions technologiques issues de l'IDM/DSM.

7 5éférences

The ATLAS MegaModel Management (AM3) Project Home Page. http://www.eclipse. org/gmt/am3, 2007. INRIA ATLAS.

Colin Atkinson and Thomas Kuhne. Model-Driven Development : A Metamodeling Foundation. IEEE Software, 20(5) :36–41, 2003.

INRIA ATLAS. KM3 : Kernel MetaMetaModel. Technical report, LINA & INRIA, Nantes, August 2005.

The ATLAS Model Weaver (AMW) Project Home Page. https://eclipse.org/gmt/amw/, 2010.

Jean Bézivin. La transformation de modèles. INRIA-ATLAS & Université de Nantes, 2003. Ecole d'Eté d'Informatique CEA EDF INRIA 2003, cours #6.

Jean Bézivin, Frédéric Jouault, Peter Rosenthal, and Patrick Valduriez. Modeling in the Large and Modeling in the Small. In U. Aßmann, M. Aksit, and A. Rensink, editors, Model Driven Architecture, European MDA Workshops : Foundations and Applications, MDAFA 2003 and MDAFA 2004, June 26-27, 2003 and Linköping, Sweden, June 10-11, 2004, Revised Selected Papers, volume 3599 of Lecture Notes in Computer Science, pages 33–46, Twente, The Netherlands, 2005. Springer.

Xavier Blanc. MDA en action. EYROLLES, March 2005.

Franck Budinsky, David Steinberg, and Raymond Ellersick. Eclipse Modeling Framework : A Developer's Guide. Addison-Wesley Professional, 2003.

Benoit Combemale, Ingénierie Dirigée par les Modèles (IDM) – Etat de l'art. 2008.

Krzysztof Czarnecki and Simon Helsen. Classification of Model Transformation Approaches. In OOPSLA'03 Workshop on Generative Techniques in the Context of ModelDriven Architecture, 2003.

Karsten Ehrig, Claudia Ermel, Stefan Hänsgen, and Gabriele Taentzer. Towards Graph Transformation Based Generation of Visual Editors Using Eclipse. Electr. Notes Theor. Comput. Sci, 127(4), 2005.

The Eclipse Modeling Framework (EMF). http://www.eclipse.org/emf, 2007. Eclipse.

The EPSILON Project Home Page. http://www.eclipse.org/epsilon/, 2014.

Patrick Farail, Pierre Gaufillet, Agusti Canals, Christophe Le Camus, David Sciamma, Pierre Michel, Xavier Crégut, and Marc Pantel. The TOPCASED project : a Toolkit in OPen source for Critical Aeronautic SystEms Design. In 3rd European Congress EMBEDDED REAL TIME SOFTWARE (ERTS), Toulouse, France, January 2006.

Jean-Marie Favre, Jacky Estublier, and Mireille Blay. L'Ingénierie Dirigée par les Modèles : au-delà du MDA. Informatique et Systèmes d'Information. Hermes Science, lavoisier edition, February 2006.

The Generic Eclipse Modeling System (GEMS). http://sourceforge.net/projects/ gems, 2007.

The Graphical Modeling Framework (GMF). http://www.eclipse.org/gmf, 2007. Eclipse.

David Harel and bernhard Rumpe. Modeling Languages : Syntax, Semantics and All That Stuff, Part I : The Basic Stuff. Technical report, Mathematics & Computer Science, Weizmann Institute Of Science, Weizmann Rehovot, Israel, August 2000.

David Harel and Bernhard Rumpe. Meaningful Modeling : What's the Semantics of "Semantics" ? Computer, 37(10) :64–72, 2004.

Jean-Marc Jézéquel, Benoit Combemale, Didier Vojtisek. *Ingénierie Dirigée par les Modèles : des concepts à la pratique*. Ellipses (Ed.), 2012.

Frédéric Jouault. Contribution à l'étude des langages de transformation de modèles. PhD thesis, Université de Nantes, September 2006. 17

Frédéric Jouault and Ivan Kurtev. Transforming Models with ATL. In Satellite Events at the MoDELS 2005 Conference, Proceedings of the Model Transformations in Practice Workshop, volume 3844 of Lecture Notes in Computer Science, pages 128–138, Montego Bay, Jamaica, 2005. Springer.

Anneke Kleppe, Jos Warmer, and Wim Bast. MDA Explained. The Model Driven Architecture : Practice and Promise. Addison-Wesley, 2003.

Pierre Laforcade, C. Choquet, "Next Step for Educational Modeling Languages: The Model Driven Engineering and Reengineering Approach", pp. 745-747, Proceedings of ICALT'06, July, Kerkrade, The Netherlands: IEEE, 2006.

Joaquin Miller and Jishnu Mukerji. Model Driven Architecture (MDA) 1.0.1 Guide. Object Management Group, Inc., June 2003.

Object Management Group, Inc. Meta Object Facility (MOF) 2.0 Core Specification, January 2006. Final Adopted Specification.

Object Management Group, Inc. Object Constraint Language (OCL) 2.0 Specification, May 2006.

Object Management Group, Inc. Unified Modeling Language (UML) 2.1.2 Infrastructure, November 2007. Final Adopted Specification.

Object Management Group, Inc. Unified Modeling Language (UML) 2.1.2 Superstructure, November 2007. Final Adopted Specification.

Object Management Group, Inc. Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, version 1.0, April 2008. 18

The Sintaks Project Home Page. http://www.kermeta.org/sintaks, 2007. INRIA TRISKELL.

Richard Soley. Model Driven Architecture (MDA), Draft 3.2. Object Management Group, Inc., November 2000.

TCS. http://www.eclipse.org/gmt/tcs/, 2007. INRIA ATLAS.

Tiger. http://tfs.cs.tu-berlin.de/~tigerprj, 2007.

Toolkit in OPen source for Critical Application & SystEms Development. http://www. topcased.org/, 2007. TOPCASED Consortium.