





## Table des matières

<b>Introduction.....</b>	<b>6</b>
<b>1 Analyse et étude technique.....</b>	<b>7</b>
1.1 Analyse du problème.....	7
1.1.1 Contexte.....	7
1.1.2 Problématique.....	8
1.2 PDMM, Graphe Conceptuel et patrons : les piliers de la méthode .....	8
1.2.1 Le Process Domain Meta Model.....	8
1.2.2 Le graphe conceptuel.....	9
1.2.3 Les patrons.....	10
1.3 Cahier des charges.....	11
1.3.1 Vue globale du projet.....	11
1.3.2 Les différentes fonctionnalités.....	12
1.3.2.1 Créer un nouveau projet.....	12
1.3.2.2 Choisir un concept.....	12
1.3.2.3 Intégrer un concept.....	13
1.3.2.4 Appliquer un patron.....	14
1.3.2.5 Visualiser le méta-modèle.....	14
1.3.2.6 Vérifier la cohérence du méta-modèle.....	14
1.3.2.7 Pistes techniques.....	15
1.4 Outils et logiciels étudiés.....	15
1.4.1 Java et Eclipse.....	15
1.4.2 ATLAS Transformation Language (ATL).....	15
1.4.3 XSLT.....	16
1.4.4 XMI.....	16
1.4.5 L'API JDom.....	17
1.4.6 Synthèse.....	17
<b>2 Spécifications.....</b>	<b>19</b>
2.1 Le diagramme de classes.....	19
2.2 L'architecture du logiciel.....	21
2.3 Fonctionnalités de l'application.....	23
2.3.1 Création d'un nouveau projet.....	23
2.3.2 Sauvegarder un projet en cours.....	25
2.3.3 Ouvrir un projet existant.....	25
2.3.4 Choisir un concept.....	25
2.3.5 Intégrer un concept.....	26
2.3.6 Vérifier la cohérence du PMUC.....	28
2.3.7 Exporter le PMUC dans un fichier XMI.....	29
<b>3 Réalisation.....</b>	<b>30</b>
3.1 Les différentes étapes de la réalisation.....	30
3.1.1 Création du squelette des classes principales.....	30
3.1.2 Créer un nouveau projet.....	30

3.1.3	Création de la classe Menu.....	31
3.1.4	Intégrer un concept.....	31
3.1.5	Sauvegarder/Ouvrir un projet.....	32
3.1.6	Vérifier la cohérence du PMUC.....	32
3.1.7	Appliquer un patron .....	33
3.1.8	L'export XMI.....	34
3.2	Structure d'un fichier XMI.....	34
3.2.1	L'en-tête .....	34
3.2.2	Les classes .....	34
3.2.3	Propriétés des classes.....	34
3.2.4	Description des associations.....	35
3.3	Tests effectués.....	36
3.4	Documentation.....	37
<b>Conclusion</b>	.....	<b>38</b>
<b>Glossaire</b>	.....	<b>39</b>
<b>Bibliographie / Webographie</b>	.....	<b>41</b>
Bibliographie.....	.....	41
Webographie.....	.....	41
<b>Annexes</b> .....	.....	<b>42</b>

## Index des illustrations

Illustration 1: Les quatre niveaux de modélisation.....	7
Illustration 2: Description des correspondances entre un concept du graphe conceptuel et une propriété UML du métamodèle à construire.....	10
Illustration 3: Diagramme des cas d'utilisation.....	11
Illustration 4: Diagramme d'activité "Choisir un concept".....	12
Illustration 5: Diagramme d'activité "Intégrer un concept".....	12
Illustration 6: Tableau des dépendances entre concepts.....	13
Illustration 7: Principe d'une transformation avec ATL (source : wiki.eclipse.org).....	14
Illustration 8: Architecture des technologies du logiciel.....	16
Illustration 9: Diagramme de classes.....	18
Illustration 10: Architecture Modèle-Vue-Contrôleur (source : http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/mvc.html).....	20
Illustration 11: Architecture de l'application.....	21
Illustration 12: Diagramme de séquences « Créer un nouveau projet ».....	23
Illustration 13: Diagramme de séquences « Intégrer un concept ».....	26
Illustration 14: Diagramme de séquences « Vérifier la cohérence du PMUC ».....	27
Illustration 15: Vue du graphe conceptuel avec Prefuse.....	36

## Introduction

Le programme pédagogique d'un Institut Universitaire Technologique (IUT) est composé d'un stage de fin d'étude. Ainsi, chaque étudiant du département informatique doit réaliser un stage en entreprise, d'une durée minimum de dix semaines, afin de prendre contact avec le monde professionnel et d'utiliser les connaissances acquises au cours de sa formation. Pour ma première expérience dans le monde du travail, j'ai décidé d'effectuer mon stage au Laboratoire d'Informatique de Grenoble (LIG), dans l'équipe Sigma.

Le LIG a été créé le 1er Janvier 2007 afin de restructurer et de regrouper dans un même organisme les unités de recherche grenobloises en informatique. Il rassemble actuellement près de cinq cents chercheurs, enseignants-chercheurs, doctorants et techniciens, répartis en 24 équipes. Le laboratoire s'implique dans quatre grands thèmes scientifiques majeurs :

- Les infrastructures informatiques, du réseau aux données
- Logiciels : fondements, modèles et ingénierie
- L'interaction : perception, Action, **IHM**, parole, robotique ...
- Connaissances : communication, traitement ...

Parmi les 24 équipes du LIG, il y a l'équipe Sigma. Cette équipe se concentre sur les systèmes d'information, que ce soit au niveau organisationnel ou opérationnel. Ses recherches sont centrées sur la formalisation, la conception et les infrastructures des systèmes d'information. Constituée de neuf enseignants-chercheurs et dix-sept doctorants, l'équipe est installée dans des locaux situés sur le Campus de Saint Martin d'Hères.

L'objectif de mon stage était de réaliser un prototype de recherche en ingénierie des systèmes d'information. La première partie du stage fut consacrée à l'étude du domaine des processus d'ingénierie de système d'information et du cahier des charges. Après cela, un dossier d'analyse expliquant les choix conceptuels et techniques du prototype devait être présenté.

Durant la seconde partie du stage, je devais concevoir le prototype en utilisant l'environnement Eclipse et le langage de programmation Java.

J'ai tout de suite été attiré par l'annonce qui proposait ce stage car il permettait de bien mettre en œuvre l'enseignement reçu à l'IUT, c'est-à-dire, une partie d'analyse avec le langage de modélisation **UML** et une partie de conception. De plus, je savais qu'en intégrant une équipe du Laboratoire d'Informatique de Grenoble (LIG), je serai bien entouré. C'est pourquoi, après un entretien, j'ai décidé de faire mon stage au LIG, dans l'équipe Sigma.

Le 6 Avril, j'ai été accueilli par Charlotte Hug, doctorante de l'équipe Sigma. Cette dernière est en troisième année de thèse. Durant ses années de recherche, elle a mis au point une méthode permettant de construire des **méta-modèles** de processus pour l'ingénierie des systèmes d'information. L'objectif de mon stage était donc de réaliser la première partie d'un logiciel ayant pour but d'informatiser cette méthode.

Afin de mettre en évidence les différentes étapes de la réalisation de ce projet, j'évoquerai brièvement l'analyse et l'étude technique réalisées qui m'ont ensuite conduit aux spécifications du logiciel pour en finir avec sa conception.

---

*Les mots écrits en vert sont définis dans le glossaire à la page 38.*

# 1 Analyse et étude technique

## 1.1 Analyse du problème

### 1.1.1 Contexte

Un Système d'Information (SI) est un « ensemble organisé de ressources (matériel, logiciel, personnel, données, procédures) permettant d'acquérir, traiter, stocker, communiquer des informations (sous forme de données, textes, images, sons etc.) dans des organisations » (*Robert Reix. Systèmes d'Information et management des organisations. Vuibert, 2000*).

Aujourd'hui, un système d'information est un élément indispensable au sein d'une organisation. L'évolution des technologies de l'informatique et de la communication lors de ces dernières années, ont entraîné une forte évolution des SI qui deviennent de plus en plus complexes et évolutifs.

La première étape pour construire un système d'information est la modélisation des données, les différentes entités et leurs relations au sein du SI. Il existe deux approches pour la modélisation :

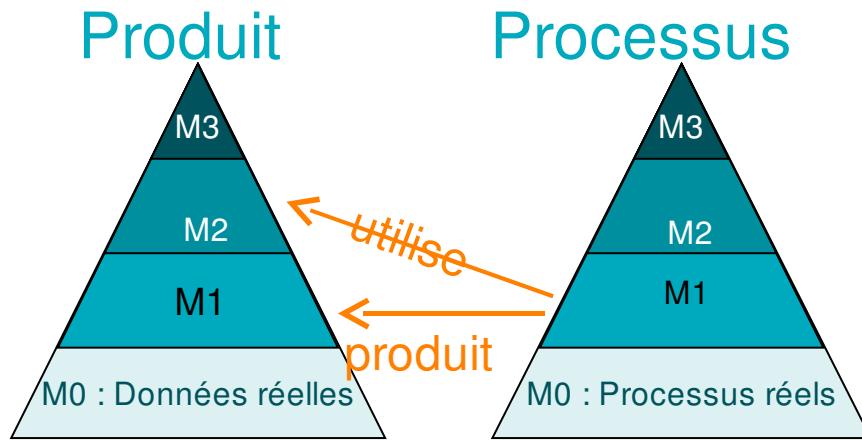
- l'approche par produit, qui se concentre essentiellement sur des objectifs de résultats spécifiquement rapportés au produit.
- l'approche par processus, qui se focalise sur les différents états de l'organisation d'une opération.

Chacune de ces deux approches est constituée de trois niveaux de modélisation :

- méta-méta-modèle (M3), qui est l'état le plus abstrait de représentation. Il est défini de manière à ce que l'on puisse créer une infinité de méta-modèles en s'inspirant du méta-méta-modèle.
- méta-modèle (M2), qui est construit sur la base du méta-méta-modèle, il est moins abstrait que celui-ci, mais reste tout de même une entité très générale qui doit pouvoir être la base d'une multitude de modèles.
- modèle (M1), la dernière couche d'abstraction avant la conception réelle d'un système.

Pour simplifier, le méta-méta-modèle est instancié pour construire le méta-modèle, qui est à son tour instancié pour construire le modèle.

Les deux approches (par produit et par processus) sont étroitement liées. En effet, la qualité d'un modèle de produit dépend directement de la qualité du modèle de processus.



*Illustration 1: Les quatre niveaux de modélisation*

Actuellement, les méta-modèles de processus sont inadaptés aux besoins des organisations et donc, le modèle de produit n'est pas obtenu de manière optimale.

### 1.1.2 Problématique

Un processus d'ingénierie de système d'information peut être construit en partant de différents points de vue :

- point de vue activité
- point de vue produit
- point de vue stratégie
- point de vue contexte
- point de vue décision

L'inconvénient des méta-modèles de processus actuels est qu'ils ne représentent les processus que d'un seul point de vue. La méthode développée tend à éliminer cet inconvénient. Elle a pour but de proposer aux **ingénieurs des méthodes** de construire leurs propres méta-modèles de processus multi-points de vue, adaptés aux besoins et spécificités des organisations.

## 1.2 PDMM, Graphe Conceptuel et patrons : les piliers de la méthode

Pour répondre de manière efficace à ce problème, la méthode développée par ma tutrice se base sur le graphe conceptuel, le Process Domain Meta-Model, appelé PDMM, et les patrons.

### 1.2.1 Le Process Domain Meta Model

L'analyse de différents méta-modèles de processus (orientés activité, produit, contexte, décision ou stratégie) existants a permis de définir le PDMM, un méta-modèle de processus multi-points de vue. Celui-ci contient uniquement les classes et les associations principales des méta-modèles de processus existants. Ce méta-modèle est représenté en respectant les normes des

diagrammes de classes UML.

Le diagramme UML qui représente le PDMM est disponible en Annexe page 41.

### 1.2.2 Le graphe conceptuel

Le graphe conceptuel a été construit à partir du PDMM. Il a pour but de faciliter à l'ingénieur des méthodes la sélection des classes du PDMM. Pour cela, les concepts, qui sont les nœuds de ce graphe, sont organisés selon un espace en trois dimensions :

- complétude
- abstraction
- précision

Ainsi, tous les concepts du graphe sont reliés entre eux par des liens de type complétude, précision, abstraction ou concrétisation (ce dernier étant l'inverse d'abstraction). L'idée est, qu'en partant d'un concept évident, on puisse construire entièrement un méta-modèle de processus en navigant d'un concept à l'autre dans le graphe. Chaque type de lien entre les concepts ont une signification bien précise :

- Un lien de type complétude permet de recouvrir un ou plusieurs points de vue. Par exemple, le concept C1 recouvre le point de vue contexte et l'ingénieur des méthodes souhaiterait avoir un autre point de vue. Pour cela, il va choisir, en partant du concept C1 une relation de complétude qui va lui permettre d'atteindre un concept C2 qui va recouvrir un point de vue différent. Un concept atteint par ce type de lien représente une classe du PDMM.
- Un lien de type abstraction/concrétisation permet d'atteindre différents niveaux d'abstractions :
  - intentionnel, qui permet de modéliser les intentions, les objectifs du processus d'ingénierie de SI.
  - opérationnel, qui permet de modéliser les opérations à réaliser pour atteindre les objectifs définis au niveau intentionnel.

Par exemple, le concept C1 se situe au niveau opérationnel et, en choisissant d'abstraire C1, on va atteindre le niveau intentionnel. Un concept atteint par une relation d'abstraction représente lui aussi une classe du PDMM.

- Un lien de type précision permet, comme son nom l'indique, de préciser un concept. En réalité, un concept atteint par une relation de précision correspond à l'application d'un patron sur le concept source de ce lien.

Le graphe conceptuel est disponible en Annexe page 41.

### 1.2.3 Les patrons

« Chaque patron décrit à la fois un problème qui se produit très fréquemment dans votre environnement et l'architecture de la solution à ce problème de telle façon que vous puissiez utiliser cette solution des millions de fois sans jamais l'adapter deux fois de la même manière ». (*Christopher Alexander. The Timeless Way of Building. Oxford University Press, 1979*).

Ils sont utilisés pour affiner le méta-modèle de processus. Il en existe deux types : les patrons génériques et les patrons de domaine.

Les différents patrons génériques :

- Concept-Catégorie de Concepts : permet de découper une classe en plusieurs catégories.
- Composition – Agrégation réflexive : permet de définir une composition ou agrégation sur une classe.
- Association réflexive : permet de représenter une association réflexive sur une classe.

Les différents patrons de domaine :

- State Transition : permet de décrire les différents états et les différentes transitions entre les états d'un produit.
- NATURE : permet de préciser le méta-modèle d'un point de vue contexte.
- MAP : permet de préciser le méta-modèle d'un point de vue stratégie.
- Time-Unit : permet de représenter les différentes unités de temps d'un concept.

Les différents patrons existants sont fournis en Annexe page 41.

En se basant sur ces trois éléments que sont le graphe conceptuel, le PDMM et les patrons, la méthode permet de construire des méta-modèles de processus d'ingénierie de systèmes d'information multi-points de vue. Les méta-modèles de processus construits sont appelés Process Model Under Construction ou PMUC.

Ce résumé sur les recherches de ma tutrice permet de comprendre l'utilité du logiciel à concevoir durant le stage. Il permettra d'informatiser la méthode. L'objectif était d'arriver à une version du logiciel fonctionnelle, sans interface graphique élaborée mais comprenant toutes les fonctionnalités permettant l'exécution de la méthode. Nous allons maintenant passer à la description du cahier des charges qui m'a été fourni.

## 1.3 Cahier des charges

### 1.3.1 Vue globale du projet

Le logiciel à concevoir peut se décomposer en trois parties principales :

- Construction des méta-modèles de processus
- Instanciation des modèles de processus
- Exécution des processus

Mon objectif était de concevoir la partie « Construction des méta-modèles de processus ». L'idée est de créer, à partir du graphe conceptuel et du PDMM, un diagramme de classes UML (le PMUC) qui représente un méta-modèle de processus. Chaque concept du graphe conceptuel peut donc se traduire par une propriété UML au sein d'un diagramme de classes.

La correspondance entre un concept du graphe conceptuel et une classe du PMUC peut se faire de deux manières :

- en passant par le PDMM
- en passant par les patrons

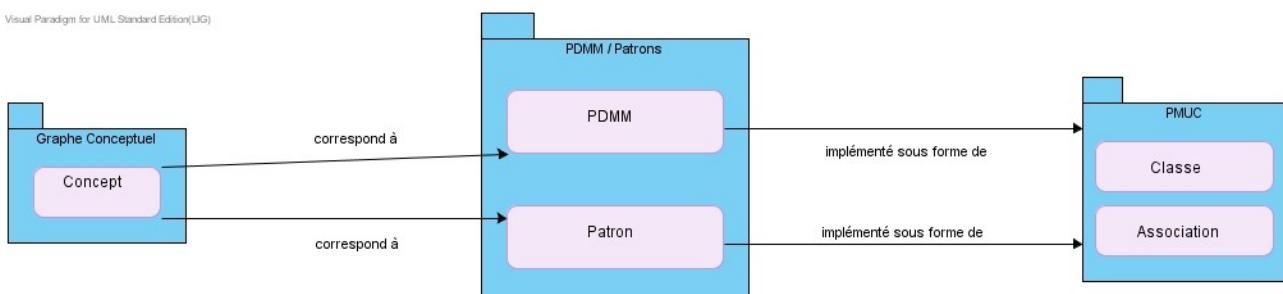
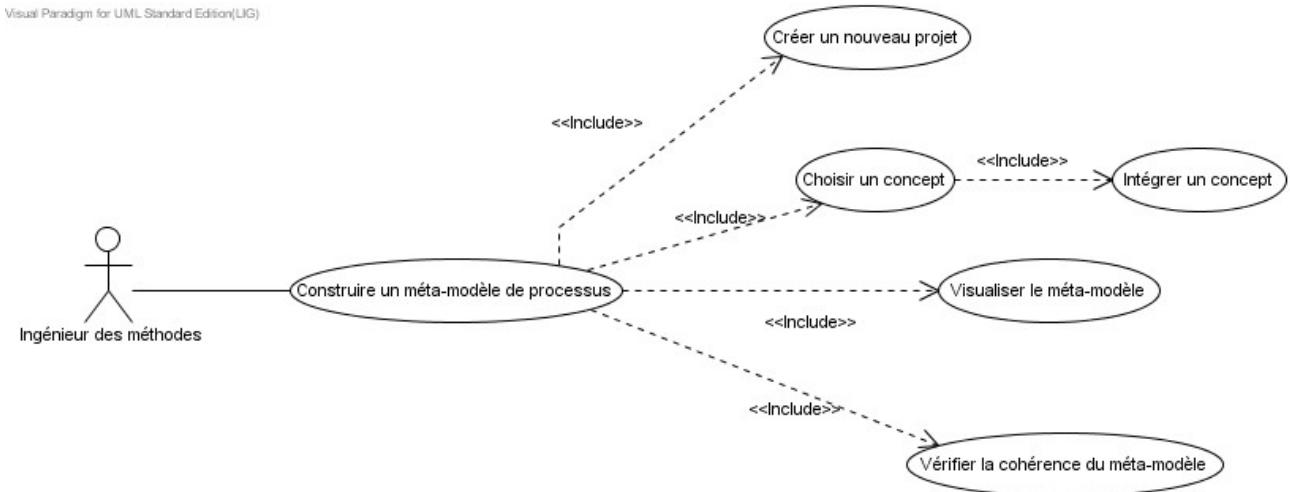


Illustration 2: Description des correspondances entre un concept du graphe conceptuel et une propriété UML du métamodèle à construire

Le logiciel doit proposer à l'utilisateur les fonctionnalités suivantes:

- Création d'un nouveau projet
- Intégration d'un concept au métamodèle (PMUC)
- Visualisation du métamodèle
- Vérification de la cohérence du métamodèle

Voici le diagramme de cas d'utilisation des fonctionnalités générales de l'outil :



*Illustration 3: Diagramme des cas d'utilisation*

### 1.3.2 Les différentes fonctionnalités

Nous décrivons ici les différentes fonctionnalités attendues.

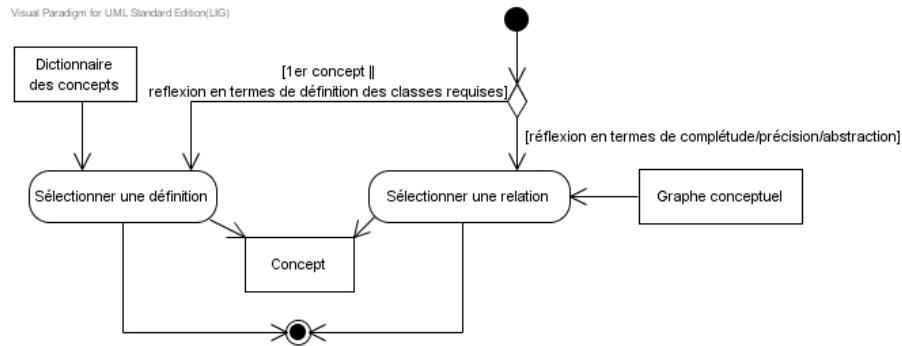
#### 1.3.2.1 *Créer un nouveau projet*

Cette fonction permet de créer un nouveau projet, c'est-à-dire un nouveau métamodèle. Elle doit mettre en place un nouvel environnement de travail pour l'utilisateur. Une fois cette opération réalisée, l'utilisateur peut créer son PMUC.

#### 1.3.2.2 *Choisir un concept*

Cette fonctionnalité demande à l'utilisateur de choisir un concept parmi une liste de concepts tirés du graphe conceptuel. Pour guider l'utilisateur, chaque concept a une définition, un ou plusieurs exemples d'utilisation et des synonymes (répertoriés dans un dictionnaire de concept). Pour choisir un concept, l'utilisateur peut réfléchir :

- en terme de définition, le choix se fait en fonction des définitions, synonymes, exemples de chaque concept.
- en terme de relation, le choix se fait en fonction des concepts déjà intégrés au PMUC. Par exemple, l'utilisateur peut compléter, préciser, abstraire ou concrétiser un concept déjà présent dans le PMUC.

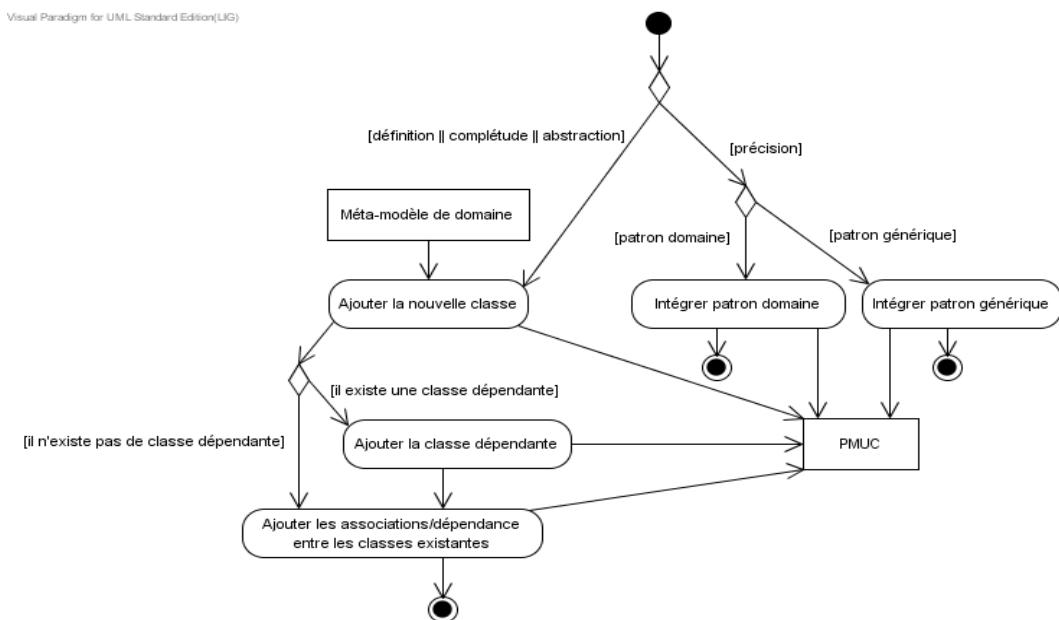


*Illustration 4: Diagramme d'activité "Choisir un concept"*

### 1.3.2.3 Intégrer un concept

Une fois le concept sélectionné, il faut l'intégrer au PMUC. Deux alternatives sont possibles :

- le concept a été sélectionné via une définition ou via une relation de complétude, d'abstraction ou de concrétisation. Le concept correspond donc forcément à une classe définie dans le méta-modèle de domaine (PDMM). Il faut ensuite vérifier les dépendances de ce concept, et ajouter au PMUC la ou les classes (s'il y a dépendance). Les associations définies dans le PDMM entre les concepts déjà présents dans le PMUC et les concepts qui viennent d'être intégrés au PMUC doivent être automatiquement ajoutées.
- Le concept a été sélectionné via une relation de précision, un patron de domaine ou générique doit être appliqué sur le concept concerné.



*Illustration 5: Diagramme d'activité "Intégrer un concept"*

#### **1.3.2.4 Appliquer un patron**

Les patrons sont obtenus par les relations de précision du graphe conceptuel. Ils correspondent à une transformation d'une partie du méta-modèle en cours de construction (PMUC).

Il existe deux types de patrons, les patrons de domaine et les patrons génériques.

- Un patron de domaine est un fragment de méta-modèle de processus existant. Il faut premièrement identifier les concepts communs entre le patron, et le méta-modèle en cours de construction. Il y a au moins un concept commun, celui qui est à la source de l'utilisation de la relation de précision. Il faut ensuite ajouter les associations définies dans le patron entre les associations du PMUC. Enfin, il faut ajouter les classes du patron qui ne sont pas définies dans le PMUC ainsi que les associations entre ces classes.
- Un patron générique correspond à une transformation du PMUC. Il met en œuvre au plus une classe.

#### **1.3.2.5 Visualiser le méta-modèle**

Cette fonctionnalité permet d'afficher le PMUC tel qu'il est à un instant T. Aucune modification ne peut être effectuée dessus. Il doit être affiché comme un diagramme de classes UML.

#### **1.3.2.6 Vérifier la cohérence du méta-modèle**

Cette fonction peut être appelée à n'importe quel moment. Elle doit vérifier les points suivants :

- Les dépendances entre concepts sont respectées. En effet, certains concepts sont dépendants d'autres concepts. Ils ne peuvent exister dans le PMUC sans que d'autres concepts y soient aussi.

Les dépendances entre concept sont stockées dans le tableau suivant.

Concept	Concept(s) dépendant(s) obligatoire(s)
Stratégie	Intention
Contexte	Situation et Alternative
Argument	Alternative
Alternative	Issue
Condition	Unité de Travail
Rôle	Alternative ou Unité de Travail ou Produit

Illustration 6: Tableau des dépendances entre concepts

- Aucune classe du PMUC ne doit être seule, c'est-à-dire non reliée par une association avec une autre classe du méta-modèle.

Si les deux conditions ci-dessus sont respectées, alors le PMUC est valide.

### 1.3.2.7 Pistes techniques

Le cahier des charges me fournissait des impératifs et des idées concernant les technologies à utiliser. Si certaines semblaient indiscutables, d'autres nécessitaient réflexion. Dans la prochaine partie, je vous présenterai les différentes technologies étudiées et/ou utilisées dans le cadre du projet.

## 1.4 Outils et logiciels étudiés

### 1.4.1 Java et Eclipse

Le langage utilisé pour la conception du logiciel devait être le langage de programmation Java. Ce langage permet de créer des logiciels très facilement portables sur plusieurs systèmes d'exploitation tels que UNIX, Microsoft Windows, Mac OS ou Linux avec peu ou pas de modification. De plus, c'est un langage orienté objet, très riche et complet. Il est très largement utilisé et donc beaucoup documenté.

Pour pouvoir développer en Java dans les meilleures conditions possibles, j'ai utilisé Eclipse. Eclipse est un environnement de développement intégré (Integrated Development Environment ou IDE) dont le but est de fournir une plate-forme modulaire pour permettre de réaliser des développements informatiques. Il est basé sur le concept de modules, plus communément appelés « plug-in », afin de s'adapter au maximum aux besoins de l'utilisateur. Conçu à la base pour réaliser des applications codées en Java, cet environnement permet au développeur de gagner énormément de temps en lui proposant des fonctionnalités telles que la complétion automatique, un accès direct à la [JavaDoc](#) sur les principaux objets Java, et bien d'autres. Eclipse est beaucoup critiqué pour sa lourdeur, mais je n'ai pas vraiment ressenti ce point faible ayant un très bon ordinateur à ma disposition. Le point fort de cet IDE est sans aucun doute le nombre important de plug-in qu'il peut intégrer.

Au-delà de ces nombreux avantages, l'IDE Eclipse était tout particulièrement intéressant dans le cadre du projet, car il propose un plug-in nommé ATL (ATLAS Transformation Language).

### 1.4.2 ATLAS Transformation Language (ATL)

ATL est un langage de transformation de modèles développé par l'Institut National de Recherche en Informatique et Automatique (INRIA). Le principe est de partir d'un modèle source, d'appliquer des transformations à ce modèle pour arriver à un modèle cible.

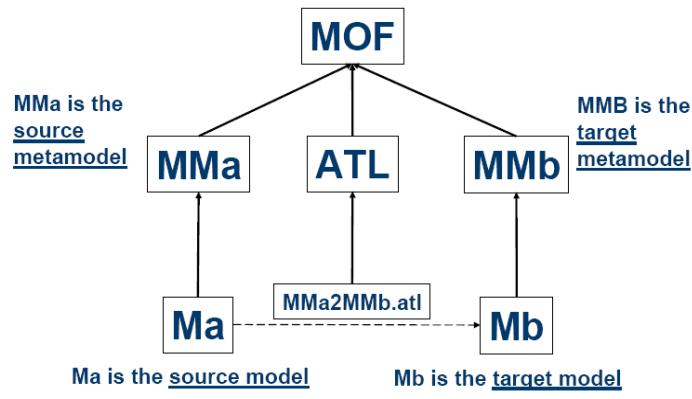


Illustration 7: Principe d'une transformation avec ATL (source : [wiki.eclipse.org](http://wiki.eclipse.org))

Pour le projet, la technologie ATL pouvait être utilisée pour l'application des patrons (génériques ou de domaine) au PMUC. Cette technologie était un point majeur du stage mais elle a révélé trop de points faibles :

- s'intègre très difficilement dans une application Java
- technologie pas encore suffisamment mature
- langage très spécifique, très peu répandu
- nécessite une importante période d'apprentissage qui devient vite pénalisante sur un stage de dix semaines
- manipule des modèles au format **ecore**, format peu intéressant dans le cadre du projet

Ayant assez longuement réfléchi sur la technologie à utiliser pour l'application des patrons, nous avons considéré que ATL présentait trop d'inconvénients. Je me suis alors penché sur deux autres manières de traiter les patrons : par le XSLT ou directement via des méthodes Java.

### 1.4.3 XSLT

XSLT est l'abréviation de eXtensible Stylesheet Language Transformations. C'est un langage de transformation de documents de type **XML** (il peut donc être utilisé sur des documents **XMI**, **HTML** etc). Il est très complet et offre beaucoup de possibilités à l'utilisateur. Il pouvait être une très bonne alternative à ATL, car le PMUC est stocké dans un fichier au format XMI (voir 1.4.4). Il permettait donc d'appliquer les transformations amenées par un patron directement sur le PMUC. L'étude de cette technologie a révélé ces différents points forts ainsi que ces faiblesses :

*Les plus :*

- très complet, beaucoup utilisé, bien documenté
- déjà rapidement étudié lors de la semaine anglaise de l'IUT
- l'appel d'une transformation se fait facilement dans une application Java

*Les moins :*

- nécessite d'appliquer les patrons directement sur le fichier XMI qui contient le PMUC, et donc de garder le fichier XMI toujours à jour (procédure assez lourde)

XSLT n'a pas non plus été gardé pour la gestion des patrons.

### 1.4.4 XMI

XMI signifie XML Metadata Interchange. C'est un standard de l'**OMG** créé pour rendre possible l'échange de modèles de données via le XML. Ce format est particulièrement utilisé pour échanger des diagrammes UML. Aujourd'hui la plupart des logiciels de modélisation UML prennent en charge le format XMI en proposant des fonctionnalités tels que l'import et l'export de diagrammes dans ce format.

Cependant, le XMI n'est pas encore totalement utilisé comme il le devrait. En effet, son but premier est de standardiser l'échange de modèles de données. Malheureusement, un fichier XMI généré par **VP-UML** ne peut pas être importé dans **Bouml** par exemple. Ce problème est dû au fait que le XMI est censé stocker le modèle UML et non le diagramme UML (Attention : il ne faut pas confondre modèle UML et diagramme UML. Le diagramme est la représentation graphique du modèle). L'exporteur XMI de VP-UML, lui, rajoute dans le fichier XMI toutes les informations concernant

l'affichage du diagramme (coordonnées graphiques, couleurs etc ...). En faisant ça, il corrompt le fichier XMI qui reste uniquement utilisable avec VP-UML.

Dans le projet, le XMI est quelque chose d'indispensable et de très important. Il sert à stocker le graphe conceptuel, le PDMM et le PMUC après export. L'idée est de pouvoir créer son méta-modèle à l'aide de l'application, de l'exporter au format XMI pour pouvoir ensuite l'ouvrir avec des logiciels de modélisation UML.

#### 1.4.5 L'API JDom

Un API (Application Programming Interface) est un ensemble de fonctions, procédures ou classes mises à disposition des programmes informatiques par une bibliothèque logicielle. JDom est donc un API Java, qui permet de lire, modifier et créer des documents au format XML ou dérivé. Il est assez simple d'utilisation tout en restant très puissant.

JDom s'avérait très utile pour notre projet afin de pouvoir lire les fichiers XMI contenant le graphe conceptuel et le PDMM. Il sera aussi utilisé pour créer le fichier XMI qui contient le PMUC.

#### 1.4.6 Synthèse

L'analyse de tous ces outils a été très enrichissante. Cependant, il a fallu faire un choix et notamment en ce qui concerne la technologie à utiliser pour gérer les patrons. Les trois possibilités étaient ATL, XSLT ou directement par du code Java. Ayant éliminé l'option ATL, je devais faire mon choix entre XSLT et Java. La décision finale s'est portée sur la deuxième solution, qui était sans aucun doute plus propre et plus simple à appliquer. Un comparatif plus précis des trois technologies ci-dessus est consultable en Annexe page 41.

Pour ce qui est de générer et d'importer les modèles XMI, nous avons choisi Bouml. Le schéma ci-dessous synthétise les différentes technologies utilisées et leur rôle pour le projet :

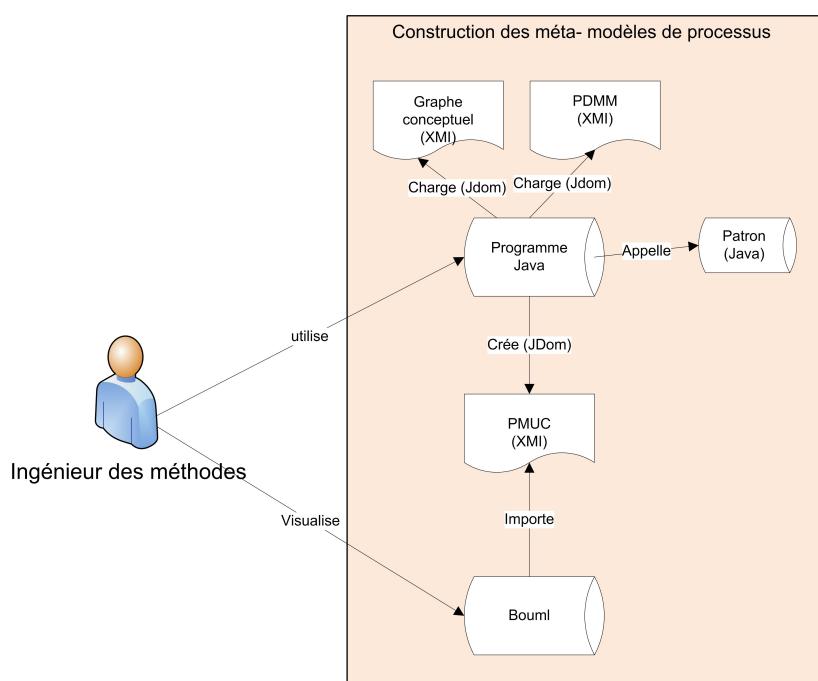


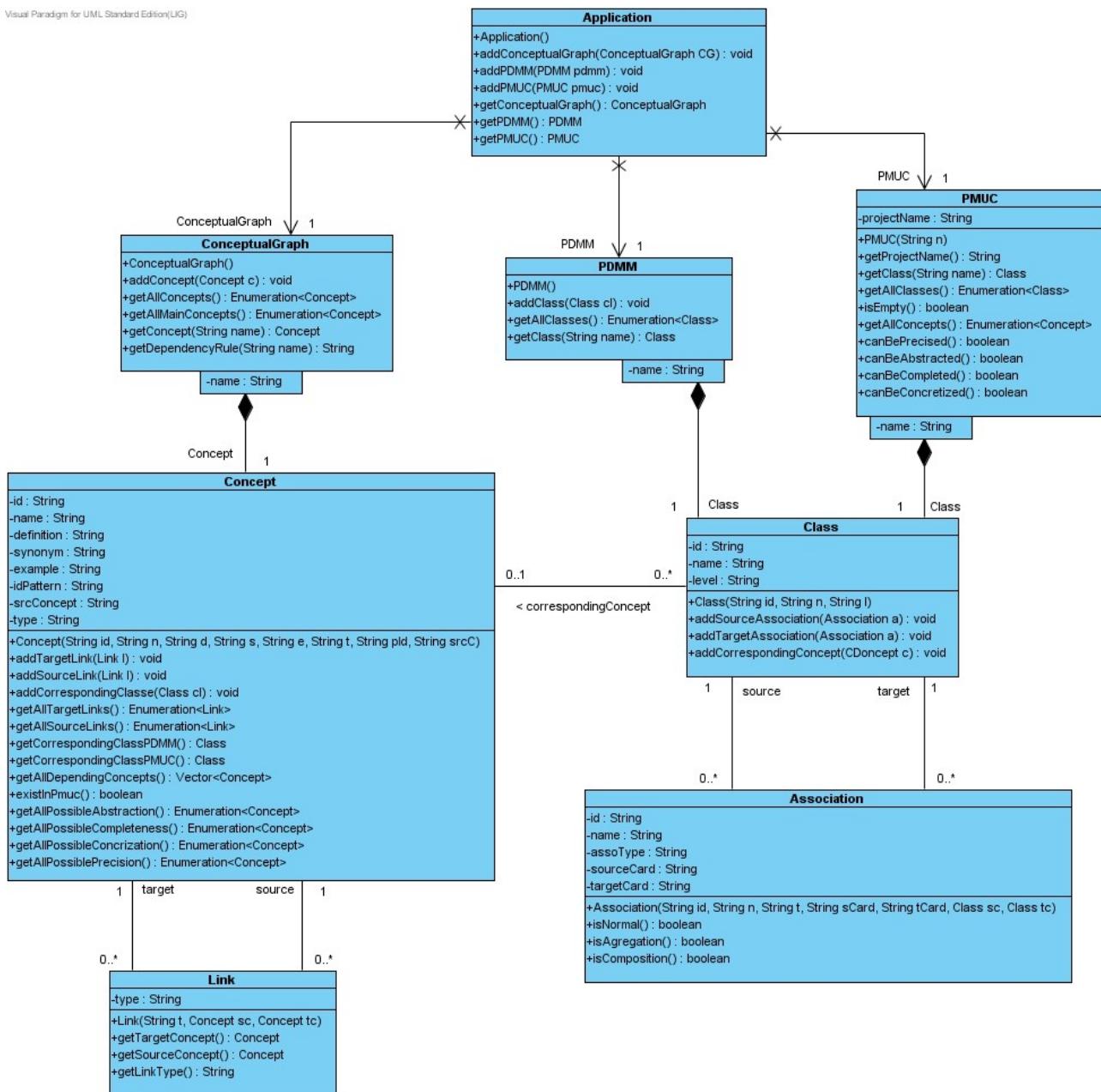
Illustration 8: Architecture des technologies du logiciel

Pour se plonger dans le projet, la période d'analyse était indispensable. Elle a commencé par la lecture de plusieurs articles écrits par ma tutrice afin d'avoir une idée sur ses recherches. J'ai ensuite étudié précisément le cahier des charges qui m'avait été fourni et, avant de passer aux spécifications, j'ai étudié différentes technologies afin de les comparer et de faire les bons choix pour le projet. A la fin de cette période d'analyse, il était temps de commencer les spécifications.

## 2 Spécifications

## 2.1 Le diagramme de classes

L'étude et l'analyse technique nous ont permis d'identifier les différentes classes indispensables à l'application. Après plusieurs propositions faites à ma tutrice, j'ai conçu le diagramme suivant :



*Illustration 9: Diagramme de classes*

## Description des différentes classes

### La classe Application :

Classe « racine » du logiciel, elle permet d'accéder à toutes les autres classes.

### La classe ConceptualGraph :

Représente le graphe conceptuel. Elle est composée d'objets de type Concept, liés à des objets de type Link.

### La classe Concept :

Représente les nœuds du graphe conceptuel :

- Les concepts secondaires sont les concepts qui correspondent à l'application d'un patron. Dans le graphe conceptuel, ce sont ceux que l'on peut atteindre uniquement par un lien de type précision.
- Les concepts principaux sont les concepts qui correspondent directement à une classe du PDMM. Ils sont atteints par des liens de type complétude, abstraction ou concrétisation.

### La classe Link :

Elle représente les liens permettant de relier chaque concept du graphe conceptuel. Un lien est composé d'un concept source et d'un concept cible. Il peut être de type « precision », « concretization », « abstraction » ou « completeness ».

### La classe PDMM et la Classe PMUC :

Ces deux classes représentent un diagramme de classes UML et sont, par conséquent, composées d'objets de la classe Class, liés à des objets de la classe Association. L'objet PDMM est construit au démarrage de l'application, en fonction des informations contenues dans le fichier XMI, qui contient le PDMM. L'objet PMUC est le méta-modèle de processus créé par l'ingénieur des méthodes.

### La classe Class :

Permet de représenter une classe UML. Une classe correspond à un concept du graphe conceptuel.

### La classe Association :

Utilisée pour représenter les différentes associations UML entre les classes du PDMM et du PMUC. Une association a un type (normal, composition, aggregation, concretisation), des cardinalités, un nom ainsi qu'une classe cible et une classe source.

Une fois le diagramme de classe créé, il s'agissait de choisir de quelle manière organiser le logiciel, autrement dit, quelle architecture mettre en place.

## 2.2 L'architecture du logiciel

Pour organiser et réaliser une application propre, j'ai décidé d'utiliser l'architecture MVC (Modèle Vue Contrôleur). Mise au point dans les laboratoires de recherche **Xerox PARC** en 1979, cette architecture permet de diviser l'application en un modèle (modèle de données), une vue (présentation, interface utilisateur) et un contrôleur (logique de contrôle, gestion des événements, synchronisation), chacun ayant un rôle restreint et précis.

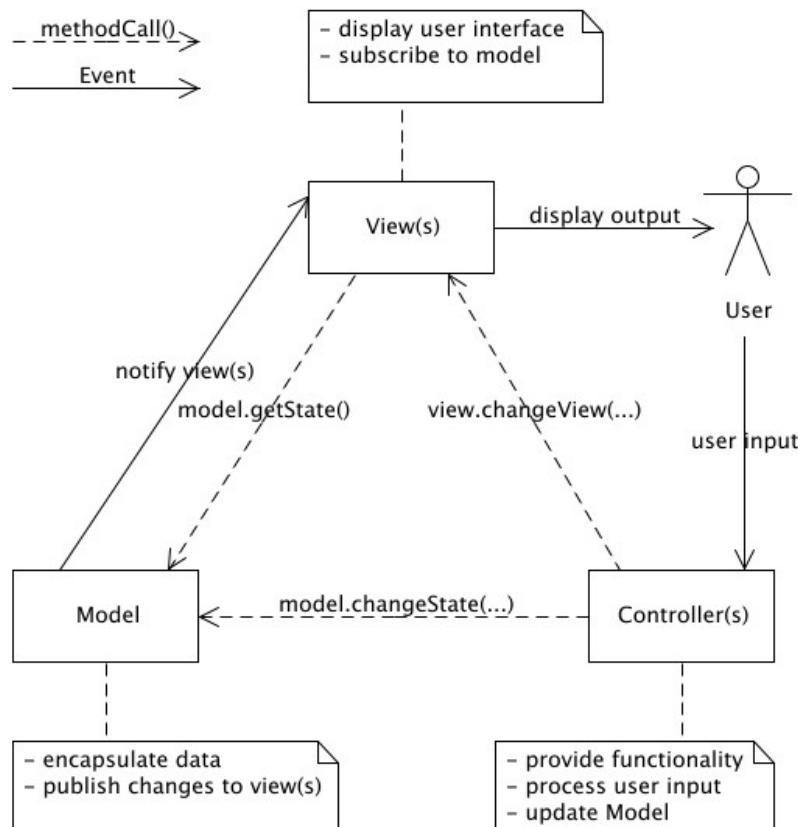


Illustration 10: Architecture Modèle-Vue-Contrôleur (source : <http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/mvc.html>)

Pour résumer, l'utilisateur agit sur le modèle (les différentes classes de l'application), en passant exclusivement par les contrôleurs. Les contrôleurs appellent ensuite les vues adaptées et les mettent à jour si le modèle change. Les vues sont une représentation fidèle du modèle à un instant T.

Basée sur le MVC, voici l'architecture générale de l'application :

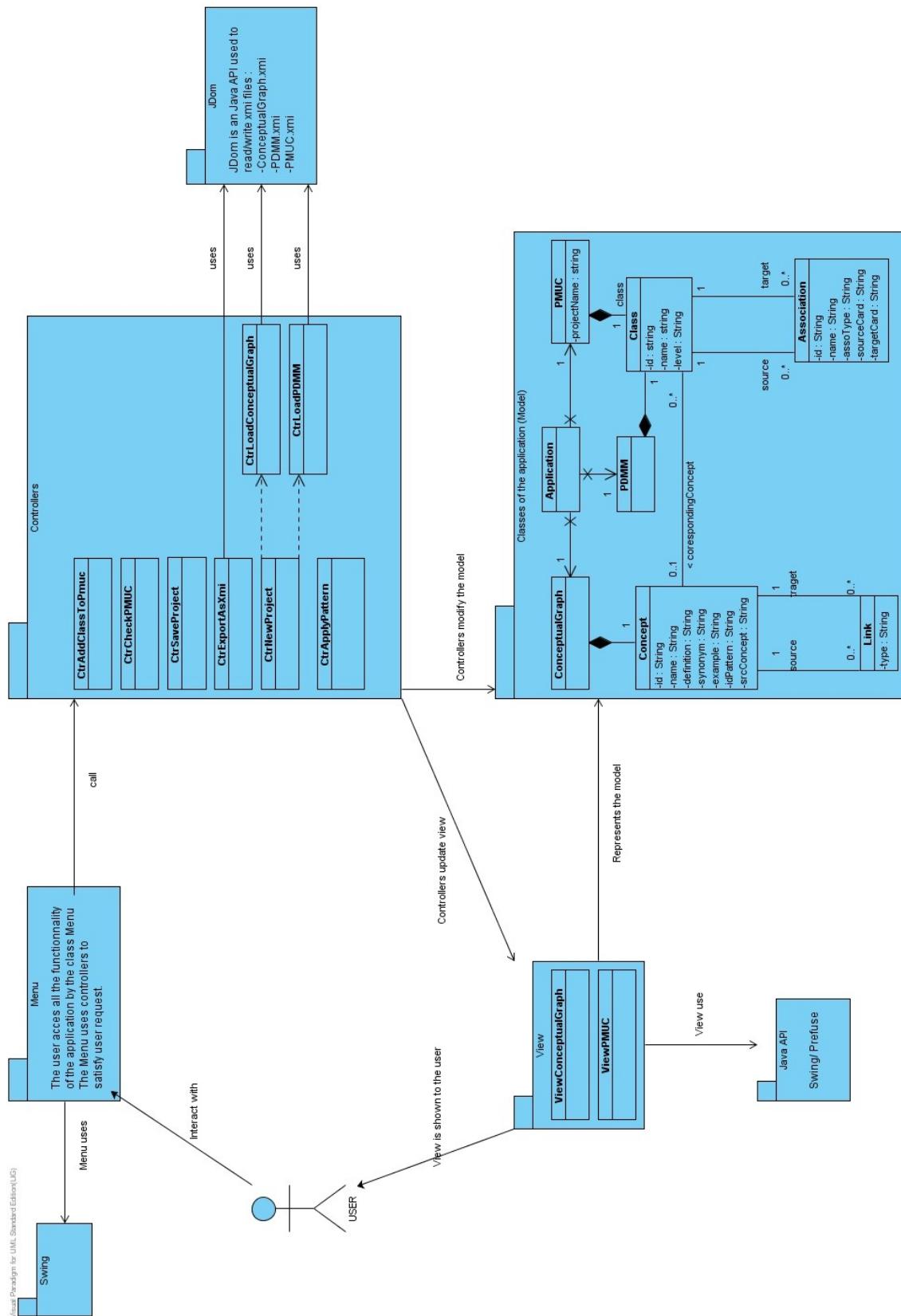


Illustration 11: Architecture de l'application

Le choix de cette architecture nécessitait l'ajout de certaines classes à l'application. Premièrement, les classes de type contrôleur. Les contrôleurs agissent sur le modèle, il fallait donc ajouter un ou plusieurs contrôleurs pour chaque fonctionnalité du logiciel. Il fallait aussi ajouter deux vues : ViewPMUC et ViewConceptualGraph. Ces vues représentent respectivement le PMUC et le graphe conceptuel, qui sont les deux seuls éléments que l'utilisateur a besoin de voir.

Enfin, la dernière classe à ajouter est la classe Menu. Elle a un rôle très important dans le logiciel, car elle indique à l'utilisateur les actions qu'il peut réaliser, et, en fonction des choix de celui-ci, appelle les contrôleurs adéquats.

## 2.3 Fonctionnalités de l'application

La liste des fonctionnalités attendues était donnée dans le cahier des charges. Il indiquait quelles fonctionnalités étaient indispensables au logiciel. La période de spécification a amené d'autres fonctionnalités découlant des choix effectués.

### 2.3.1 Crédit d'un nouveau projet

Cette fonctionnalité doit être proposée à l'utilisateur au démarrage de l'application. Il doit avoir le choix entre ouvrir un projet existant ou bien créer un nouveau projet. S'il décide de créer un nouveau projet, le nom du projet lui est demandé. L'application crée à ce moment là un nouveau répertoire du même nom que le projet. Ce répertoire peut contenir deux fichiers différents. L'un dont l'extension est .data, qui contient une sauvegarde des objets de l'application au format binaire, et l'autre dont l'extension est .xmi, qui est le fichier généré lors de l'export du PMUC.

Ensuite, l'application se charge, c'est-à-dire qu'elle crée les objets ConceptualGraph et PDMM à partir des informations lues (à l'aide de l'API JDom) dans les fichiers ConceptualGraph.xmi et PDMM.xmi. Une fois ces deux objets créés, il faut établir les correspondances entre les concepts principaux du graphe, et les classes du PDMM. Pour des raisons de lisibilité et simplification du code, trois contrôleurs sont utilisés pour réaliser cette opération :

- CtrNewProject, qui se charge de demander le nom du projet, appelle le contrôleur CtrLoadConceptualGraph, appelle le contrôleur CtrLoadPDMM et établit les correspondances entre les concepts principaux et les classes du PDMM.
- CtrLoadConceptualGraph permet de charger le graphe conceptuel, en créant tous les objets Concept et Link à partir des informations contenues dans le fichier ConceptualGraph.xmi.
- CtrLoadPDMM, qui, de la même manière que CtrLoadConceptualGraph crée l'objet PDMM en créant les objets Class et Association à partir des informations contenues dans le fichier PDMM.xmi.

Le diagramme de séquences UML ci-dessous représente l'algorithme de la création d'un nouveau projet.

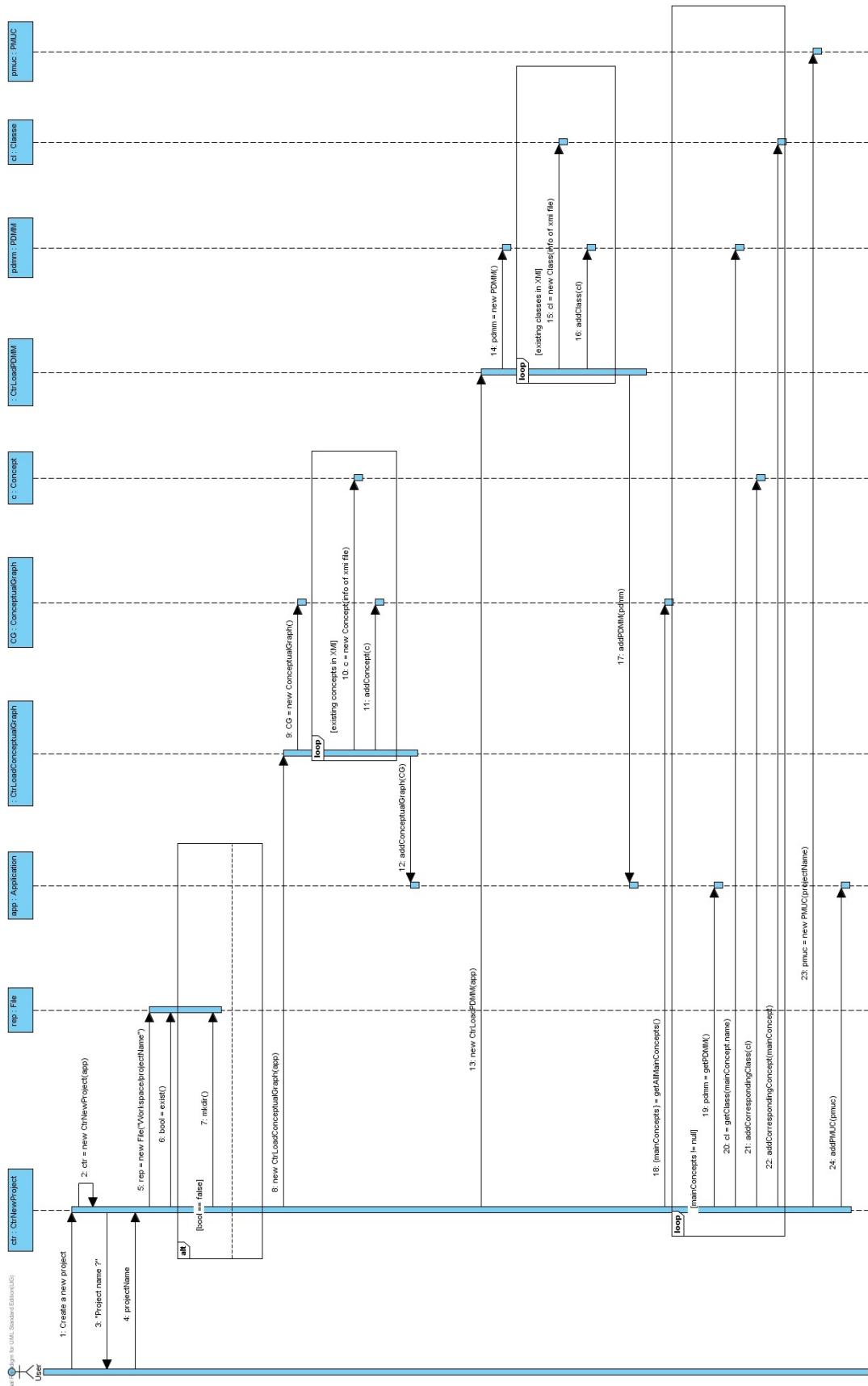


Illustration 12: Diagramme de séquences « Crée un nouveau projet »

### 2.3.2 Sauvegarder un projet en cours

Lors de l'utilisation du logiciel, la sauvegarde du projet doit être possible à tout moment. Après étude des différentes solutions possibles, la sauvegarde des classes de l'application au format binaire semblait la plus appropriée. Ainsi, si l'utilisateur sauvegarde un projet, il peut quitter l'application et reprendre exactement là où il en était. La fonctionnalité « ouvrir un projet en cours » vient donc s'ajouter à la fonctionnalité « créer un nouveau projet ». Le choix entre ces deux options doit être laissé à l'utilisateur lors de l'exécution du logiciel.

### 2.3.3 Ouvrir un projet existant

Cette fonctionnalité est proposée en parallèle de « créer un nouveau projet ». Elle doit montrer à l'utilisateur une liste contenant tous les projets qu'il a sauvegardés afin que celui-ci puisse facilement faire son choix. Une fois le projet à ouvrir choisi, l'initialisation du logiciel se fera par le processus de **sérialisation Java**.

### 2.3.4 Choisir un concept

C'est la classe Menu qui se charge de cette fonctionnalité. N'oublions pas que cette fonctionnalité est là pour guider l'utilisateur, l'un des objectifs principaux de la méthode. Pour ce faire, la classe Menu propose deux niveaux différents :

Le premier niveau :

What do you want to do ?
1 : Modify my PMUC
2 : Check my PMUC
3 : Save my project
4 : See my PMUC
5 : Export my PMUC to xmi
6 : Exit application

Il permet d'accéder à toutes les fonctionnalités du logiciel. Dans le cas où l'utilisateur choisit l'option 1 (Modify my PMUC) on arrive dans le deuxième niveau de menu.

Actions you can do on your PMUC
1 : Add a concept by definition
2 : Complete my PMUC
3 : Precise my PMUC
4 : Abstract my PMUC
5 : Concretize my PMUC
6 : Return to main menu

Ce menu là n'est pas statique. Il évolue en fonction de l'état actuel du PMUC. Par exemple, si le PMUC est vide, seul l'option 1 (Add a concept by definition) est disponible. Autre exemple, les options 2, 3, 4, 5 sont disponibles si et seulement si un des concepts du PMUC peut être complété, précisé, abstrait ou concrétisé.

A ce moment là, l'utilisateur fait son choix. Deux scénarios sont alors possibles :

- Si l'utilisateur choisit la première option, tous les concepts principaux, ainsi que tous les concepts pouvant être directement atteints (par précision, complétude, abstraction ou concrétisation) par ceux présents dans le PMUC sont proposés (s'ils n'ont pas déjà été ajoutés précédemment).
- Sinon, si l'utilisateur choisit l'option 2, 3, 4 ou 5, seuls les concepts pouvant être atteints par la relation choisie sont proposés. Supposons qu'il souhaite compléter son PMUC, le programme va alors lui proposer la liste des concepts présents dans le PMUC qui peuvent être complétés, autrement dit, tous ceux ayant un lien de type complétude qui mène à un concept qui n'existe pas dans le PMUC. L'utilisateur choisit ensuite le concept qu'il souhaite compléter. Le programme lui liste enfin tous les concepts qui peuvent compléter le concept précédemment choisi, qui ne sont pas dans le PMUC.

### 2.3.5 Intégrer un concept

L'intégration d'un concept peut se faire de deux manières :

- Si le concept à intégrer est un concept secondaire (le concept correspond à l'application d'un patron) alors c'est le contrôleur CtrApplyPattern qui va se charger de l'ajouter. Ce contrôleur contient une méthode par patron. Il va appeler la méthode du patron correspondant au concept. Cette méthode va agir directement sur le PMUC en lui appliquant les transformations définies par le patron.
- Si le concept en question est un concept principal (le concept correspond à une classe du PDMM), c'est le contrôleur CtrAddClassToPMUC qui s'en charge. Il va premièrement récupérer (s'il en existe) tous les concepts dépendants au concept choisi. Ensuite, pour chaque concept à ajouter, il va récupérer la classe du PDMM correspondant à ce concept, en faire une copie et ajouter cette copie au PMUC. Il va ensuite créer toutes les associations apportées par ces ajouts de classes, toujours en se basant sur le PDMM.

L'algorithme de l'ajout d'un concept principal est présenté dans le diagramme de séquences ci-dessous :

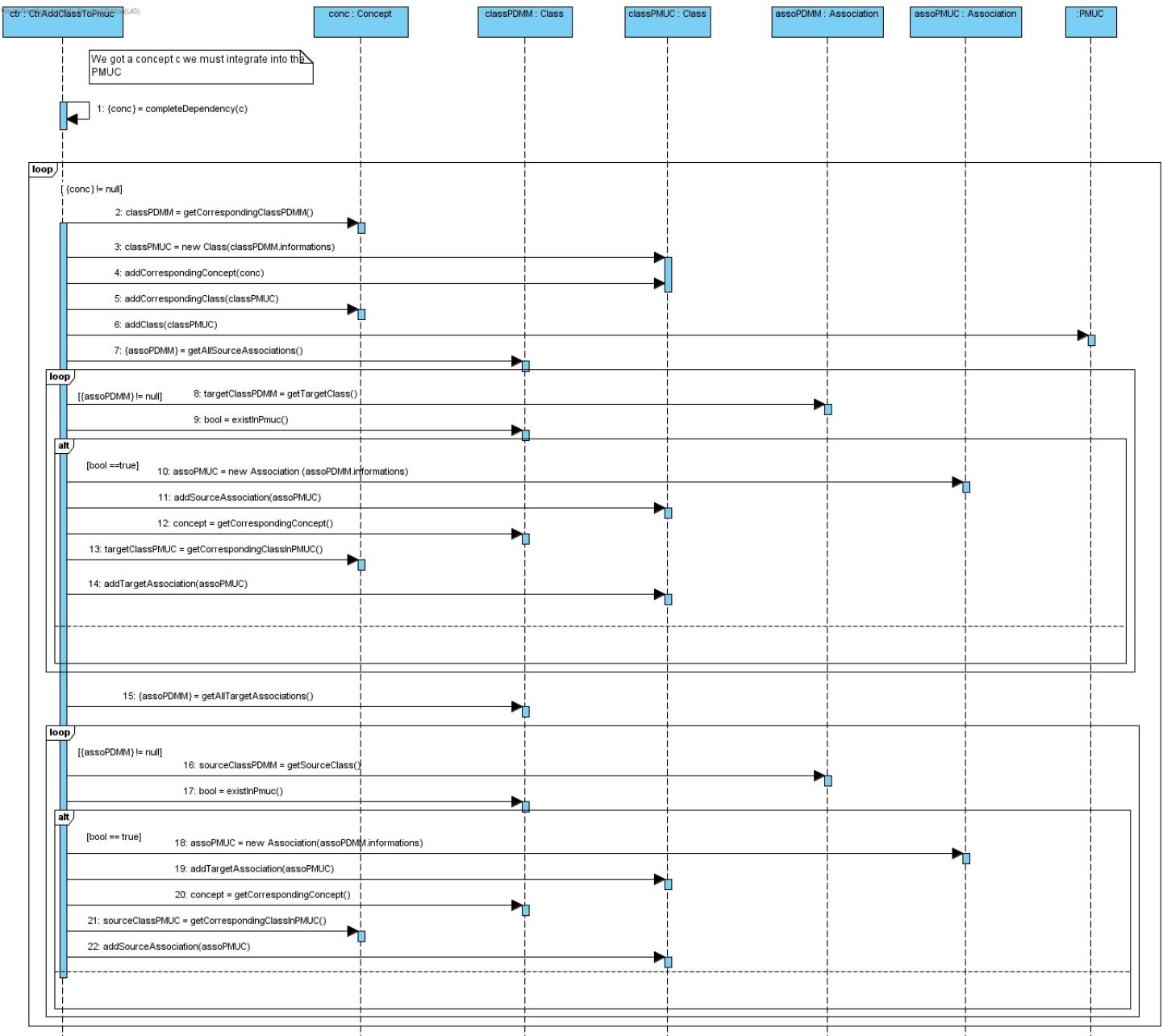


Illustration 13: Diagramme de séquences «Intégrer un concept»

### 2.3.6 Vérifier la cohérence du PMUC

Comme indiqué dans le cahier des charges, cette fonctionnalité doit vérifier les points suivants :

- aucune classe du PMUC ne doit être esseulée (elles doivent toutes être liées au minimum à une autre classe par une association).
- Les dépendances doivent être vérifiées.

L'algorithme de vérification du PMUC est présenté dans ce diagramme de séquences :

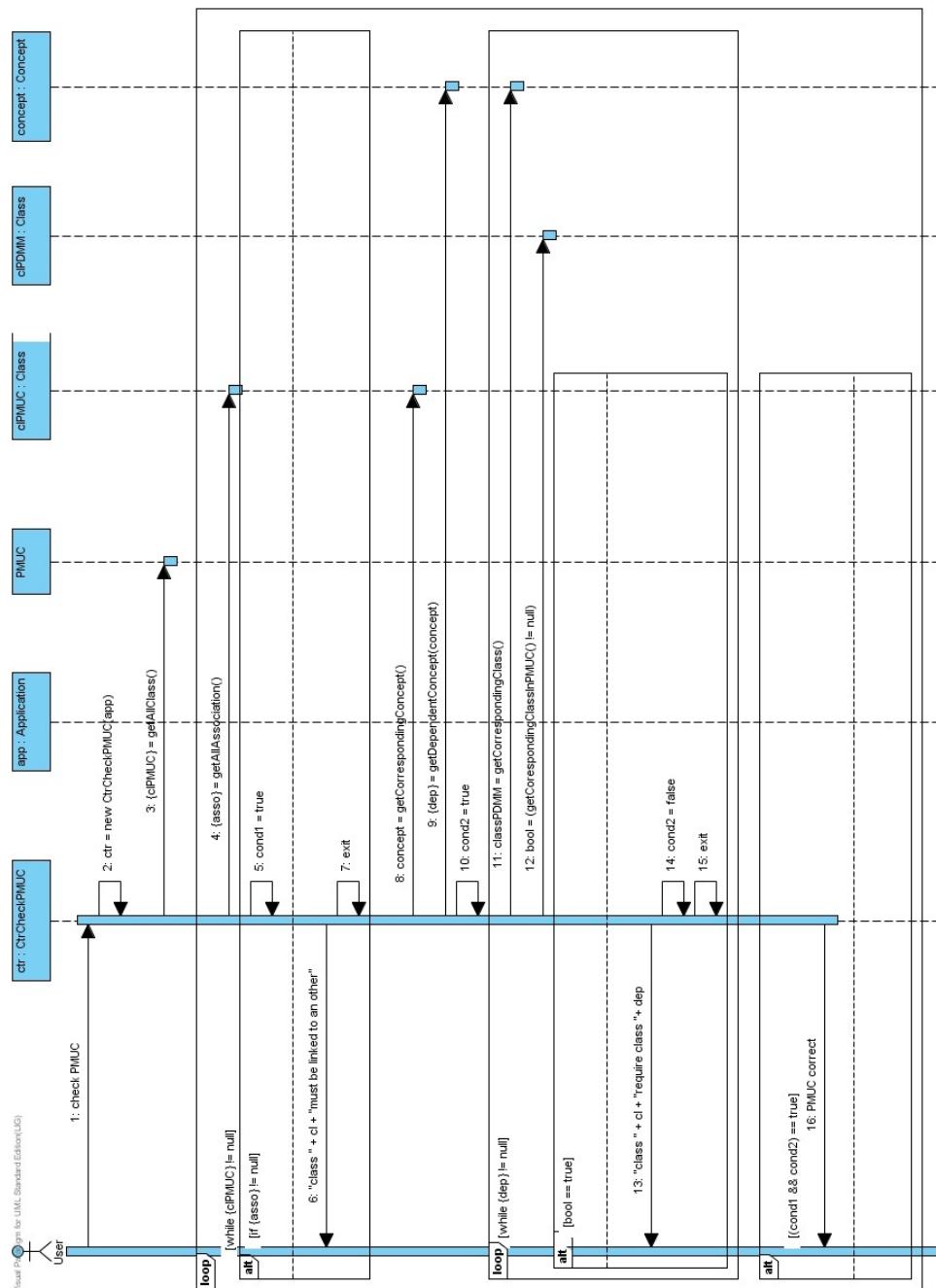


Illustration 14: Diagramme de séquences « Vérifier la cohérence du PMUC »

### 2.3.7 Exporter le PMUC dans un fichier XMI

L'export du PMUC dans un fichier XMI est une fonctionnalité très intéressante pour l'utilisateur. En effet, elle permet à l'utilisateur de reprendre son méta-modèle de processus, créé avec le logiciel, dans un atelier de génie logiciel.

Pour créer l'exporteur, l'utilisation de l'API Java JDom est indispensable. Cette opération est gérée par le contrôleur CtrExportAsXmi, qui crée un fichier XMI valide et l'enregistre dans le dossier consacré au projet.

La phase de spécification a été une phase cruciale du stage. Je faisais un rapport, contenant des diagrammes UML, des scénarios, ... , que je présentais ensuite à ma tutrice. Elle regardait mes idées, posait des questions, exposait son opinion. Les plusieurs petits brain-stormings nous ont permis de nous mettre d'accord sur les spécifications finales du logiciel à développer. C'est pourquoi, quatre semaines après le début de mon stage, je rentrai dans la partie réalisation du programme.

## 3 Réalisation

### 3.1 Les différentes étapes de la réalisation

Les périodes d'analyse et de spécification ayant été faites avec rigueur, la réalisation en terme de programmation n'a jamais été perturbée par des problèmes majeurs.

#### 3.1.1 Crédit du squelette des classes principales

La première étape fut de coder le squelette des principales classes de l'application en se servant du diagramme de classes réalisé durant la période de spécification. Pour cela il fallait penser à indiquer dans la déclaration des classes qu'elles seront utilisées au sein d'une architecture MVC et qu'elles seront sauvegardées grâce au mécanisme de **sérialisation**.

Exemple de déclaration de la classe Concept :

```
public class Concept extends Observable implements Serializable
```

#### 3.1.2 Créer un nouveau projet

La deuxième étape consistait à programmer la fonction « créer un nouveau projet ». Il fallait donc construire les objets ConceptualGraph et PDMM à partir des fichiers XMI. Pour générer les fichiers XMI, j'ai décidé d'utiliser le logiciel Bouml car il permettait, contrairement à la plupart de ses concurrents, d'exporter seulement le modèle, sans ajouter d'informations secondaires telle que la représentation graphique du diagramme. Il a donc fallu traduire le graphe conceptuel et le PDMM dans Bouml afin de pouvoir générer les fichiers XMI correspondants et ainsi avoir une base de travail.

Le fichier XMI contenant le graphe conceptuel a dû subir quelques modifications manuelles. En effet, un objet de la classe Concept doit contenir des informations tels que la définition du concept, des synonymes, des exemples et, si le concept est un concept secondaire, un identifiant permettant de savoir à quel patron correspond le concept et sur quel concept principal le patron est appliqué. C'est pourquoi, des attributs « definition », « example », « synonym », « patternId » et « srcConcept » ont été ajoutés pour chaque élément du fichier XMI représentant un concept.

Cette partie du code nécessitait l'utilisation de l'API JDom. Comme dit précédemment, cet API est très utilisé et beaucoup documenté. J'ai donc facilement trouvé de bons tutoriaux sur internet et ai rapidement assimilé les principes de son fonctionnement.

Principe de fonctionnement de JDom :

Il faut dans un premier temps comprendre les différents objets qu'utilise JDom :

- « Document », qui sert à stocker la racine du document XML ou dérivé.
- « Element », qui peut représenter n'importe quel élément d'un fichier XML. Un élément commence par « < » et se finit par « > ».

- « Namespace », qui sert à déclarer les différents espaces de nom (**namespace**) qui interviennent dans le document.

Dans notre cas, il a fallu ajouter deux namespaces « UML 2.1 » et « XMI 2.1 ». La déclaration d'un namespace se fait de cette manière :

```
Namespace nsUML;
```

```
nsUML = Namespace.getNamespace("uml", "http://schema.omg.org/spec/UML/2.1");
```

Il faut ensuite créer un objet SAXBuilder. Il a pour fonction de retrouver ou de créer la racine d'un document XML.

```
SAXBuilder builder = new SAXBuilder();
```

On se sert de **builder** pour créer une variable de type Document afin de se positionner à la racine du document :

```
Document doc = builder.build("ConceptualGraph.xmi");
```

Une fois la racine créée, on peut naviguer, lire et modifier de différentes manières le fichier, avec des fonctions prédéfinies telles que getChild(), getAttribute(), setAttribute().

C'est donc en utilisant cette technologie que je récupérais toutes les informations dont j'avais besoin afin de créer les objets de type Class, Concept, Association, Link, ConceptualGraph et PDMM.

Pour vérifier que le chargement de ces deux objets se faisait correctement, j'utilisais deux vues qui me permettaient de valider le bon fonctionnement de mon code.

### 3.1.3 Cration de la classe Menu

Il tait possible,  ce stade du dveloppement d'integrer l'interface avec laquelle l'utilisateur agit, autrement dit, la classe Menu, classe qui permet de filtrer les actions que l'utilisateur peut effectuer en fonction de l'tat de son PMUC  un instant T. Le menu tait au dbut simple, je le faisais voluer  chaque ajout de fonctionnalit.

### 3.1.4 Integrer un concept

Une fois que la fonction « crer un nouveau projet » fonctionnait comme elle le devait et que le Menu tait disponible, il tait possible de commencer  programmer les differentes mthodes permettant de crer un PMUC (« integrer un concept »). J'ai donc cre le contrôleur CtrAddClassToPmuc. L'algorithme que celui-ci devait effectuer avait dj t reprsent prcisment dans un diagramme de squences. Celui-ci a donc t ralis de manire efficace, en suivant  la lettre le diagramme de squence.

J'ai ensuite enrichi le Menu fin que celui-ci permette  l'utilisateur de complter son PMUC en rflchissant en terme de relation ou de dfinition (« choisir un concept »). La fonctionnalit « integrer un concept » tant dj fonctionnelle, il me restait juste  slectionner dans l'objet ConceptualGraph quel concept proposer  l'ingnieur des mthodes en fonction de son choix.

### 3.1.5 Sauvegarder/Ouvrir un projet

Comme dit précédemment, la sauvegarde de l'application se fait au format binaire. C'est le contrôleur CtrSaveProject qui est en charge de cette fonctionnalité. Pour la sauvegarde, on utilise le mécanisme de sérialisation, qui permet d'enregistrer ou de récupérer des objets dans un flux.

La sérialisation d'un objet est effectuée lors de l'appel de la méthode writeObject() sur un objet implémentant l'interface ObjectOutputStream (par exemple un objet de la classe ObjectOutputStream).

La désérialisation d'un objet est effectuée lors de l'appel de la méthode readObject() sur un objet implantant l'interface ObjectInputStream (par exemple un objet de la classe ObjectInputStream).

Lorsqu'un objet est sauvé, tous les objets qui peuvent être atteints depuis cet objet sont également sauvés. De plus, tout objet n'est sauvé qu'une fois grâce à un mécanisme de cache.

Pour qu'un objet puisse être sauvé ou récupéré sans lever une exception il doit implémenter l'interface Serializable.

Sauvegarder un projet en cours :

```
FileOutputStream f = new FileOutputStream(projectName+".data");
ObjectOutputStream out = new ObjectOutputStream(f);
out.writeObject(app);
out.flush();
out.close();
```

Ouvrir un projet en cours :

```
FileInputStream f = new FileInputStream(projectName+".data");
ObjectInputStream in = new ObjectInputStream(f);
app = (Application) in.readObject();
```

### 3.1.6 Vérifier la cohérence du PMUC

« Vérifier la cohérence du PMUC » est effectuée par le contrôleur CtrCheckPMUC. L'algorithme réalisé par celui-ci avait déjà été détaillé lors de la période de spécification. Il a donc été créé rapidement, en respectant les spécifications.

### 3.1.7 Appliquer un patron

L'application d'un patron au PMUC se fait directement par des méthodes Java (les solutions ATL et XSLT ayant été abandonnées). Pour une gestion simple et propre, c'est le contrôleur CtrApplyPattern qui se charge de cette fonctionnalité. Il contient une méthode par patron, chacune agissant directement sur le PMUC. La création d'une transformation se fait de cette manière :

Implémenter une nouvelle transformation :

La première étape consiste à récupérer la classe du PMUC correspondante au concept source, celui sur lequel on va appliquer le patron.

Il faut ensuite construire, si c'est nécessaire, la ou les nouvelles classes définies dans le patron.

Rappel: une classe se construit en quatre étapes:

- 1: Construction de l'objet Class cl = new Class()
- 2: Ajout d'un concept correspondant à la classe cl.addCorrespondingConcept(Concept c)
- 3: Ajout au concept correspondant la classe cl précédemment créée c.addCorrespondingClass(cl)
- 4: Ajout de la classe au PMUC pmuc.addClass(cl)

Une fois la ou les classes créées, il faut créer les associations entre cette ou ces classes et les classes déjà existantes.

Rappel: une association se construit en trois étapes:

- 1: Construction de l'objet Association a = new Association()
- 2: Ajout de l'association à la classe source sourceClass.addSourceAssociation(a)
- 3: Ajout de l'association à la classe cible targetClass.addTargetAssociation(a)

Attention : lors de la création des classes, il faut bien faire la distinction entre les nouvelles classes (celle qui n'existent pas dans le PDMM) et les autres. Exemple : si un patron est constitué de la classe Work\_Unit, il faut vérifier que cette classe n'existe pas déjà dans le PMUC. Si elle existe déjà, on la récupère et on ne crée que les associations. Si elle n'existe pas, on l'ajoute au PMUC en passant par le contrôleur CtrAddClassToPmuc.

*Extrait de la documentation du projet*

### 3.1.8 L'export XMI

La fonctionnalité « exporter mon PMUC dans un fichier XMI » n'a pas posé de réels problèmes, mais elle s'est avérée plus longue et compliquée que prévu. En effet, pour avoir un fichier XMI valide et cohérent, il faut que la fonction permettant l'export gère parfaitement les différents identifiants qui composent un document XMI. Pour cela, il fallait faire attention qu'à chaque création de classe ou d'association, celles-ci aient un identifiant unique. Pour mieux comprendre ce problème, nous allons maintenant étudier plus dans le détail la structure d'un fichier XMI.

## 3.2 Structure d'un fichier XMI

### 3.2.1 L'en-tête

Elle indique la version du standard XMI utilisé ainsi que les différents espaces de nommage (namespace).

### 3.2.2 Les classes

Classe et concept sont deux choses différentes, mais elles sont représentées de la même manière dans un fichier XMI :

---

```
<packagedElement xmi:type="uml:Class name="Work_Unit" xmi:id="classeId">
    <ownedAttribute xmi:type="uml:Property" name="level">
        ...
    </ownedAttribute>
</packagedElement>
```

---

L'élément `<packagedElement>` contient son nom, son identifiant, etc. Pour chaque association (entrante ou sortante) et pour chaque attribut, il y a un élément `<ownedAttribute>`. Il contient le nom de la propriété UML (cette propriété pouvant être l'attribut d'une classe ou une association). Dans le cas d'une association, il contient le nom du rôle.

### 3.2.3 Propriétés des classes

Comme dit précédemment, les classes ont des propriétés. Celles-ci peuvent représenter une association (entrante ou sortante) ou bien l'attribut d'une classe.

Association :

---

```
<ownedAttribute name="target" xmi:id="idAssoUnique" association="idAssoCommun">
    <type xmi:type="uml:Class" xmi:idref=" id de l'autre classe de l'association"/>
```

---

---

```

<lowerValue xmi:type="uml:LiteralString" xmi:id="idCard" value="1"/>
<upperValue xmi:type="uml:LiteralString" xmi:id="idCard" value="*"/>
</ownedAttribute>

```

---

La compréhension du rôle de chaque identifiant est primordiale. idAssoUnique, est un identifiant unique à chaque élément <ownedAttribute> de type association. Il est réutilisé dans la description des associations (voir un peu plus bas).

idAssoCommun, est l'identifiant de l'association, c'est-à-dire qu'il est le même pour l'élément <ownedAttribute> représentant l'association "sortante" et l'association "rentrante".

Rappelons qu'une association est définie deux fois: une fois dans sa classe source et une fois dans sa classe cible.

Attribut :

---

```

<ownedAttribute name="level" xmi:id="MIPSI_000xO47" >
    <type xmi:type="uml:Class" xmi:idref="identifiant du type de l'attribut"/>
</ownedAttribute>

```

---

L'identifiant du type de l'attribut permet de connaître le type de l'attribut (dans notre cas, si level est « intentional » ou « operational »). Cette information est stockée une seule fois, à l'extérieur des classes :

---

```

<packagedElement xmi:type="uml:DataType" xmi:id="datatype_1" name="intentional"/>
<packagedElement xmi:type="uml:DataType" xmi:id="datatype_2" name="operational"/>

```

---

### 3.2.4 Description des associations

En plus d'être définies dans chaque classe, les associations sont redéfinies à l'extérieur :

---

```

<packagedElement xmi:type="uml:Association" xmi:id="idAssoCommun" visibility="private">
    <memberEnd xmi:idref="idAssoUnique"/>
    <memberEnd xmi:idref="idAssoUnique"/>
<packagedElement/>

```

---

### 3.3 Tests effectués

Les tests sur le bon fonctionnement du logiciel se sont faits tout au long du développement. Au début, il s'agissait de vérifier la trace laissée par la vue du PMUC, afin de voir si les modifications de celui-ci étaient bien prises en compte et correctement faites. Actuellement, la représentation du PMUC se fait d'une manière textuelle :

```
-----
-                                     PMUC
-----

CLASS : Work_Product
-LEVEL : operational
*outgoing association
*incoming association
    Work_Unit( *.* ) ----in(normal)----> ( *.* )Work_Product
    Work_Unit( 1.* ) ----out(normal)----> ( *.* )Work_Product
    Role( 1.* ) ----is_responsible_for(normal)----> ( *.* )Work_Product
-----
CLASS : Role
-LEVEL : operational
*outgoing association
    Role( 1.* ) ----carries_out(normal)----> ( *.* )Work_Unit
    Role( 1.* ) ----is_responsible_for(normal)----> ( *.* )Work_Product
*incoming association
-----
CLASS : Work_Unit
-LEVEL : operational
*outgoing association
    Work_Unit( *.* ) ----in(normal)----> ( *.* )Work_Product
    Work_Unit( 1.* ) ----out(normal)----> ( *.* )Work_Product
*incoming association
    Role( 1.* ) ----carries_out(normal)----> ( *.* )Work_Unit
-----
```

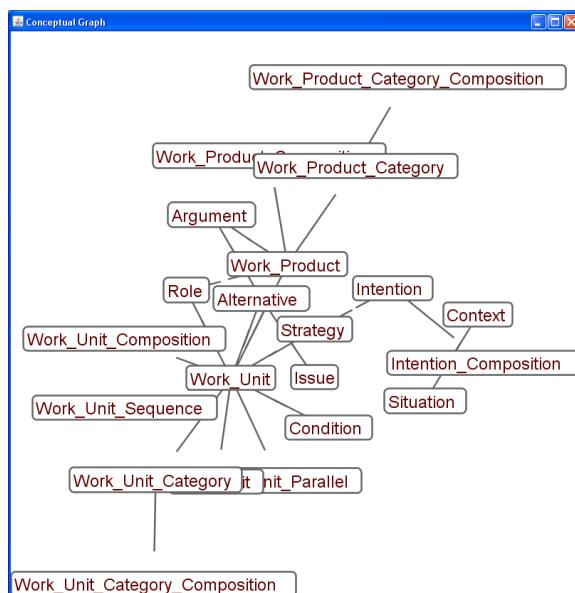
Une fois l'exporteur XMI réalisé, beaucoup de tests ont été fait en créant des PMUC, en les exportant dans un fichier XMI et en important ce fichier XMI dans Bouml. Une fois l'import réalisé, il est possible de visualiser le diagramme de classes. Ainsi, je pouvais voir si le diagramme représentait bien le PMUC que j'avais construit.

### 3.4 Documentation

Comme dit au début du mémoire, ma mission était de concevoir la première partie du logiciel. La suite sera reprise par d'autres stagiaires. Afin que ceux-ci ne perdent pas de temps pour comprendre ce qui a déjà été réalisé et de quelle manière ceci a été fait, la documentation prenait une dimension très importante.

Pour cela, le code a été documenté par une [JavaDoc](#). J'ai aussi décidé de regrouper tous les points importants concernant différentes parties du projet dans un site internet afin que les futurs stagiaires puissent accéder rapidement à tous mes conseils et mes explications.

La suite du projet consiste notamment à créer la vue du graphe conceptuel en utilisant l'API Prefuse. Prefuse est un outil qui permet de visualiser des informations sous différentes formes graphiques. Il est très intéressant car il permet de créer des graphes interactifs. L'idée est que, plus tard, les ingénier des méthodes choisissent des concepts en cliquant simplement sur les nœuds du graphe qui les intéressent. Étant très intrigué par les possibilités de cet API, je l'ai rapidement étudié. On peut trouver sur internet quelques vidéos montrant les capacités impressionnantes de Prefuse. Malheureusement, son utilisation n'est vraiment pas facile et la documentation très faible. J'ai pu tout de même créer un programme permettant la visualisation du graphe conceptuel :



*Illustration 15: Vue du graphe conceptuel avec Prefuse*

Ce travail n'a pas été inutile, car il a permis de confirmer le choix technique qu'est Prefuse pour la visualisation du graphe conceptuel et il permettra au stagiaire suivant de s'inspirer de mon code.

## Conclusion

Le projet à réaliser durant mon stage de fin de Diplôme Universitaire Technologique se situe dans le vaste contexte des Systèmes d'Information. Il a pour but de soutenir une nouvelle méthode développée par Charlotte Hug, doctorante au Laboratoire d'Informatique de Grenoble. Cette méthode doit permettre aux ingénieurs des méthodes de construire des méta-modèles de processus afin de modéliser les différents concepts et leurs interactions. Elle est basée sur trois éléments que sont le Process Domain Meta Model, dit PDMM, le graphe conceptuel et les patrons.

Mon stage s'est décomposé en quatre grandes parties. La première consistait à comprendre les recherches de ma tutrice, et d'analyser le cahier des charges qu'elle m'avait fourni. Ensuite, après avoir compris les objectifs et les principales fonctionnalités du logiciel à réaliser, une étude des technologies pouvant être utilisées ainsi que les spécifications techniques de l'application ont été réalisées. Une fois les spécifications validées par ma tutrice, la conception du logiciel a commencé. A la fin du stage, après avoir obtenu un code fonctionnel répondant positivement aux tests, j'ai porté une grande attention à la documentation, en soulignant les points importants du logiciel. Cela représentait quelque chose d'important étant donné que la suite du projet allait être confiée à un autre stagiaire.

Je pense avoir plutôt bien géré mon temps tout au long de la réalisation du logiciel. En effet, j'ai atteint les objectifs qui m'étaient fixés avant la fin du stage. Cela m'a permis de rédiger une documentation solide et de commencer le maquettage de l'interface graphique. La période de programmation n'a pas pris de retard et ce notamment grâce à des spécifications poussées, qui m'ont permis d'anticiper et d'envisager rapidement des solutions face aux éventuels problèmes (planning en Annexe page 41).

Ce stage m'a permis de mettre en pratique les connaissances qu'elles soient techniques ou tactiques, en terme de méthode de raisonnement, acquises au cours de ma formation à l'IUT. J'ai aussi découvert le monde de la recherche, qui m'était jusque là inconnu. La recherche représente quelque chose d'essentiel, totalement lié au progrès technologique. J'ai été impressionné par la diversité des secteurs, rien que dans le domaine de l'informatique.

D'un point de vue plus personnel, je suis plutôt content de mon passage au laboratoire. En effet, j'ai su travailler efficacement et de manière autonome. J'ai réussi à m'adapter assez rapidement au changement que fut le passage de l'IUT au monde du travail. Le plus dur pour moi a été de comprendre le sujet sur lequel travaillait ma tutrice. En effet, je ne connaissais pas du tout le domaine de la métamodélisation et de l'ingénierie des processus de systèmes d'information, notions qui se sont précisées au cours du temps. En revanche, le côté technique ne m'a pas posé de problèmes.

Finalement, je suis satisfait de l'ensemble de mon stage qui selon moi m'a permis de faire un point sur ce que j'avais acquis durant ces deux dernières années. Il m'a prouvé que bien d'autres facettes de l'informatique restent à étudier et m'a conforté dans mon choix de continuer mes études dans le domaine de l'informatique. Cette première expérience aura été très enrichissante et restera un bon souvenir.

## Glossaire

- Bouml : Logiciel de modélisation UML qui a l'avantage de générer des fichiers XMI propres qui respectent parfaitement la norme.
- Ecore : extension du format qui permet de stocker des méta-modèles construit avec EMF.
- EMF : Eclipse Modeling Framework est un module de l'IDE Eclipse qui permet de construire des modèles.
- HTML : l'Hypertext Markup Language, est le format de données conçu pour représenter les pages web.
- IDM : Ingénierie Dirigée par les Modèles, domaine de l'informatique mettant à disposition des outils, concepts et langages pour créer et transformer des modèles informatiques.
- IHM : Interface Homme Machine.
- Ingénieur des méthodes : personne qui va définir les différents processus d'ingénierie de système d'information dans le cadre d'un projet ou d'une organisation.
- JavaDoc : la Javadoc est un outil développé par Sun Microsystems permettant de créer la documentation d'un logiciel en format HTML depuis les commentaires présents dans un code source en Java.
- Méta-modèle : un méta-modèle peut être défini comme un langage de modélisation. Il sert ainsi à exprimer les concepts communs à l'ensemble des modèles d'un même domaine.
- namespace : les espaces de nommage XML offrent une méthode simple pour qualifier les noms des éléments et des attributs utilisés dans des documents XML, en associant ceux-ci avec des espaces de nommage désignés par des références d'URI.
- OMG : Object Management Group est un consortium qui a pour but de fournir une standardisation des modèles.
- Sérialisation : dans le cadre d'un développement Java, elle permet de stocker l'état des objets, ainsi que la manière de les recréer plus tard. Un objet peut ainsi exister entre deux exécutions d'un programme, ou entre deux programmes : c'est la persistance objet.
- Sun Microsystems : Sun Microsystems est un constructeur d'ordinateurs et un éditeur de logiciels américain.
- UML : Unified Modeling Language est un langage graphique de modélisation des données et des traitements. C'est une formalisation très aboutie et non-propriétaire de la modélisation objet utilisée en génie logiciel.
- URI : un URI, de l'anglais Uniform Resource Identifier, soit littéralement identifiant uniforme de ressource, est une courte chaîne de caractères identifiant une ressource sur un réseau.
- VP-UML : Visual Paradigm for UML est un logiciel de modélisation UML.
- xerox PARC : Xerox PARC (pour Xerox Palo Alto Research Center) est un centre de recherches en informatique de la firme Xerox, situé à Palo Alto en Californie.

- XMI : XML Metadata Interchange est un standard permettant l'échange de modèles entre différents logiciels de modélisation.
- XML : l'eXtensible Markup Language est un langage informatique de balisage générique. Il est essentiellement utilisé pour stocker des données organisées.

## Bibliographie / Webographie

### Bibliographie

- Pascal Roques, « UML 2 par la pratique », Eyrolles, 2006
- Grady Booch, James Rumbaugh, Ivar Jacobson, « The Unified Modeling Language User Guide », Addison-Wesley, 2000
- Charlotte Hug, Agnès Front, Dominique Rieu, «Ingénierie des processus : une approche à base de patrons», in /Modèles, Formalismes et outils pour les systèmes d'information/, RSTI série Ingénierie des systèmes d'Information, Vol.13, n°4
- Charlotte Hug, Agnès Front, Dominique Rieu, « A Method based on a Conceptual Graph for Information Systems Engineering Processes », soumis à 28th International Conference on Conceptual Modeling.

### Webographie

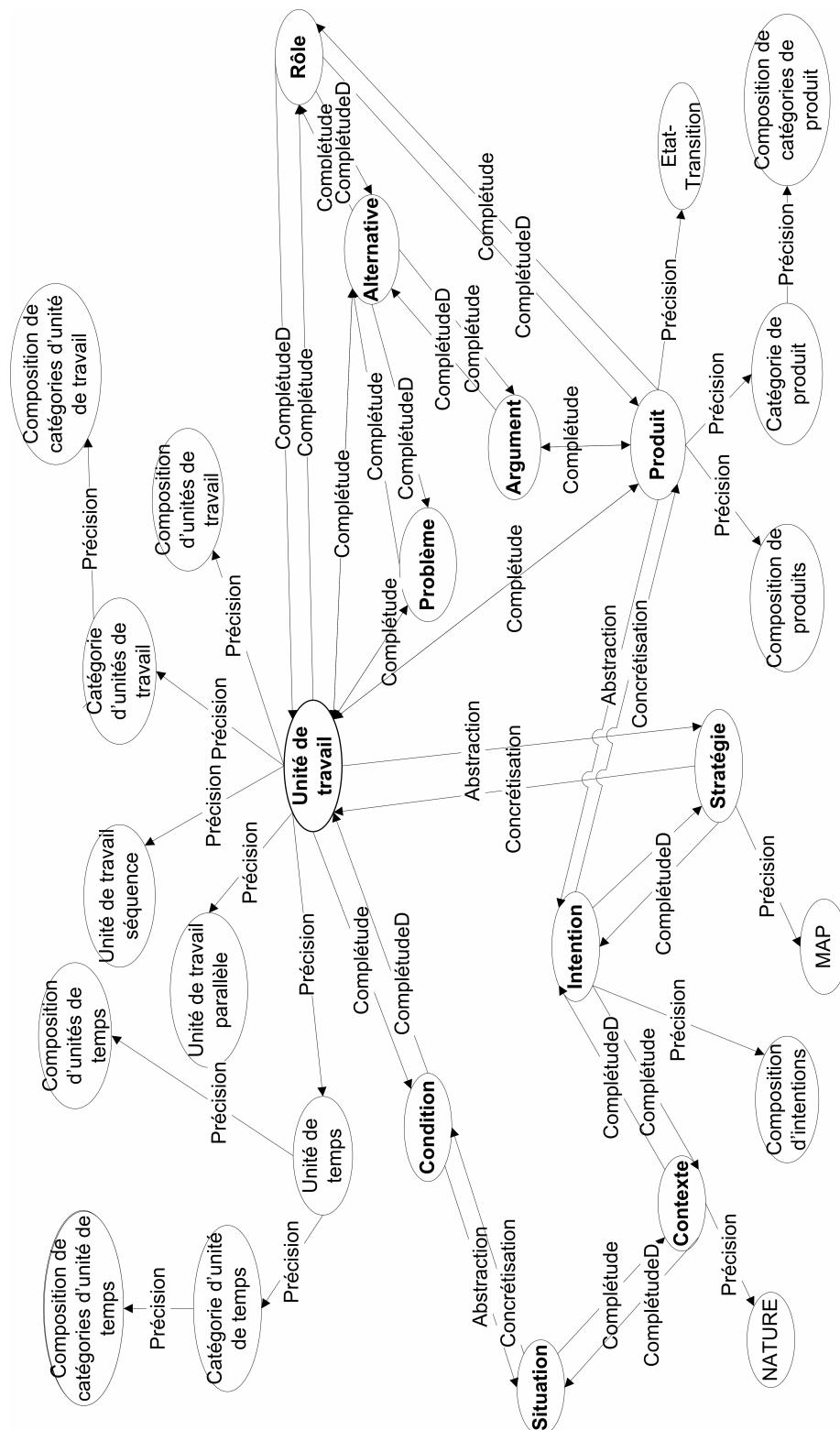
- Site de l'OMG : <http://www.omg.org/>
- Support Java : <http://java.sun.com/>
- Support UML : <http://www.uml.org/>
- Support Eclipse : <http://www.eclipse.org/>
- Support JDom : <http://www.jdom.org/>
- Support Bouml : <http://bouml.free.fr/>
- Tutoriaux en tous genres : <http://www.developpez.com/>
- Forum d'aide : <http://www.developpez.net/forums/>
- Encyclopédie : <http://wikipedia.org/>
- Moteur de recherche : <http://www.google.com/>

## Annexes

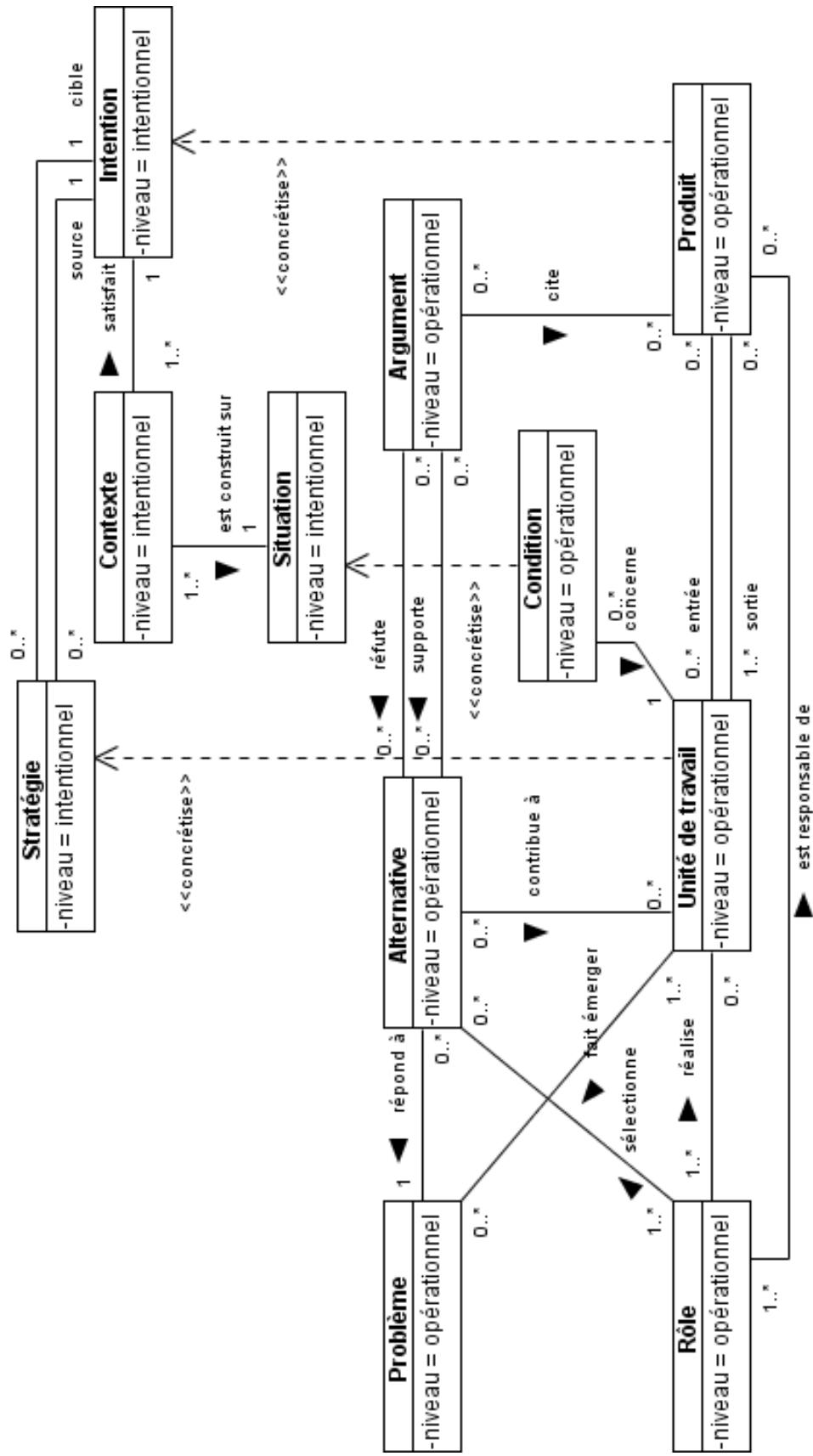
### Index des Annexes

Le Graphe Conceptuel.....	42
Le Process Domain Meta Model (PDMM).....	43
Les Patrons.....	44
Patrons génériques.....	44
Patrons de domaine.....	45
Synthèse sur la technologie à utiliser pour la gestion des patrons.....	47
Captures d'écran du site web de documentation.....	49
Plannings.....	50
Planning prévisionnel.....	50
Planning réel.....	51
Explication des différences entre le planning réel et le planning prévisionnel.....	52

## Le Graphe Conceptuel



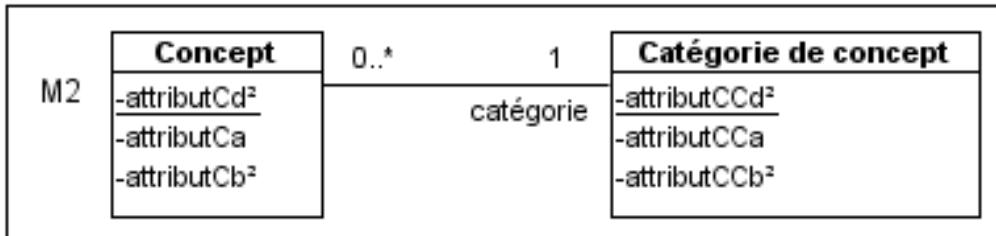
## Le Process Domain Meta Model (PDMM)



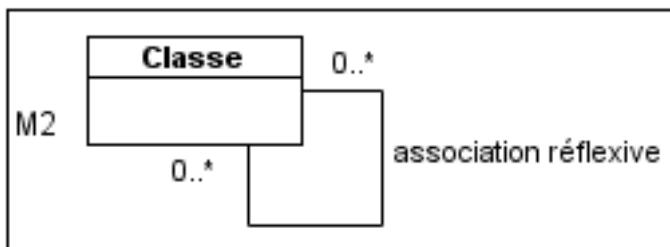
## Les patrons

### Patrons génériques

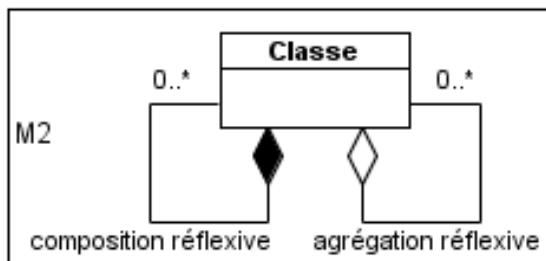
- Concept Catégorie de Concept



- Association réflexive

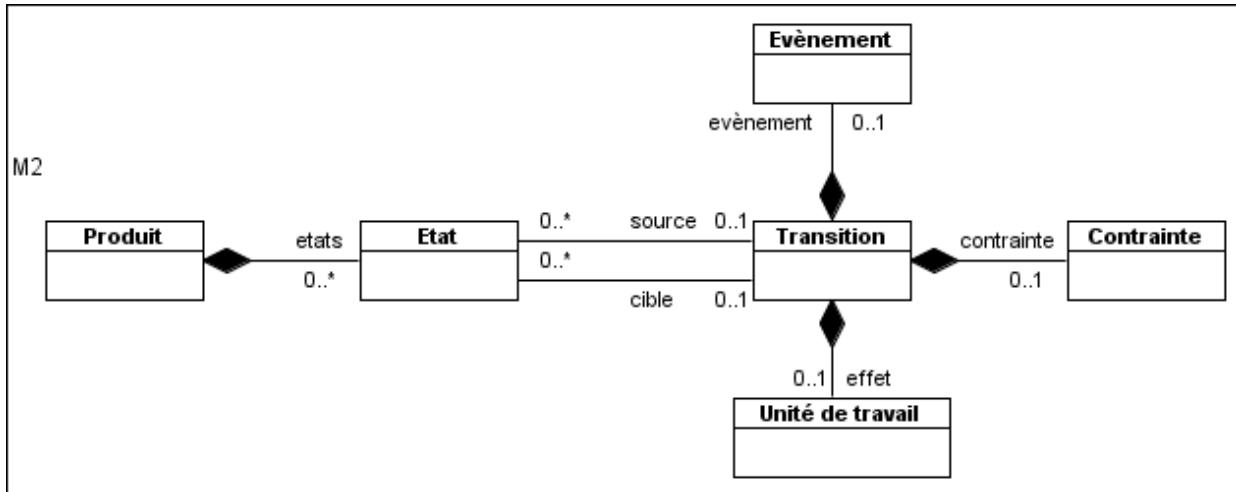


- Composition/agrégation réflexive

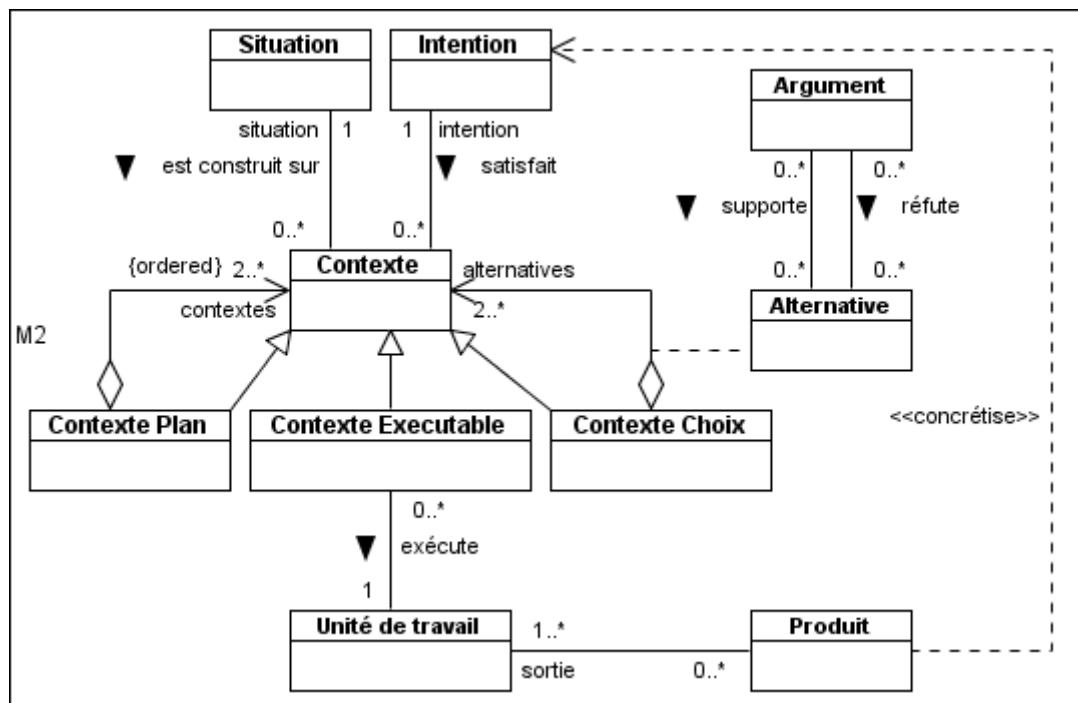


## Patrons de domaine

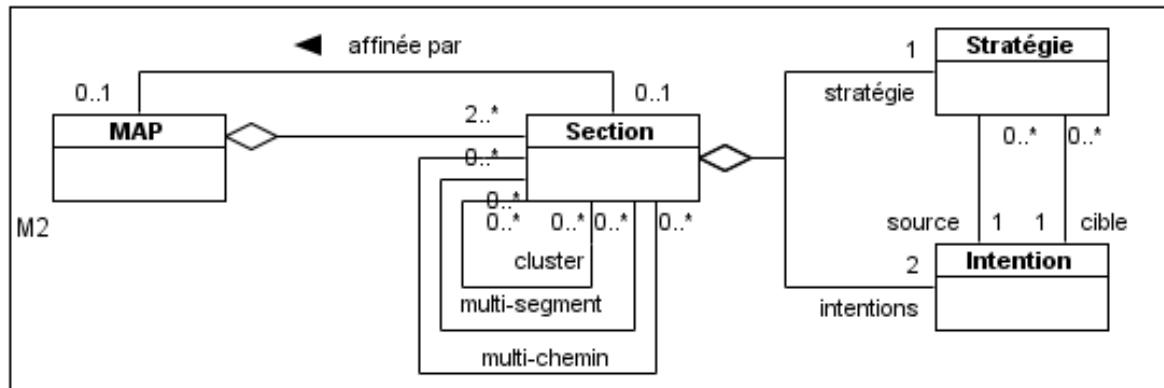
- État Transition (State Transition)



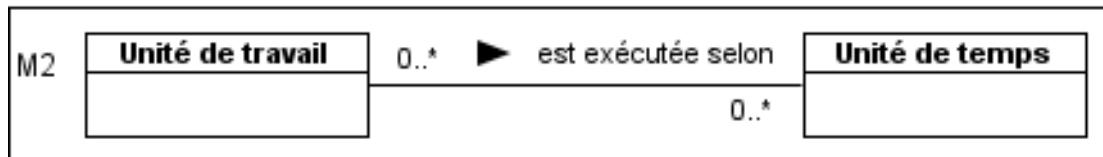
- NATURE



- MAP



- Unité de temps (Time Unit)



## Synthèse sur la technologie à utiliser pour la gestion des patrons

Cette synthèse est un rapport fait à ma tutrice pour lui donner mon avis sur le meilleur choix quant à la technologie à utiliser pour l'intégration des patrons dans le logiciel :

---

### **Quelle technologie utiliser pour l'application de patrons ?**

#### Problématique :

Nous avons des objets (PMUC, Class, Association) dans un programme Java qui représentent un diagramme de classe UML. A certaines parties de ce modèle, on veut pouvoir appliquer des patrons qui résulteraient en une transformation du modèle.

#### **ATL**

Nécessite de garder tout le temps à jour le modèle UML au format ecore et de définir une transformation différente par patron.

Les moins:

- Assez lourd à réaliser (chaque transformation nécessite un méta-modèle source et un méta-modèle cible).
- Chaque appel de patron nécessite la transformation de l'objet PMUC en fichier ecore.
- S'intègre difficilement dans Java.
- Problème d'évolution ATL/Java.
- Nécessite une longue période d'apprentissage.

#### **XSLT**

Nécessite de sauvegarder tout le temps le PMUC dans un fichier xmi mis à jour à chaque modification. Il y aurait un fichier xslt par patron, et l'application du patron se ferait directement dans le fichier xmi.

Scénario:

- 1: Sauvegarde du PMUC en son état actuel dans le fichier xmi.
- 2: Choix du patron à appliquer.
- 3: Sélection du fichier xslt.
- 4: Application du patron directement sur le fichier xmi.
- 5: Recharger les classes PMUC, Class, Association à partir du fichier xmi modifié par le patron.

Les plus:

- Facile à appeler dans un programme Java.

Les moins:

- Assez lourd à mettre en place (voir scénario).
- Technologie plus utilisée pour sélectionner des éléments dans un fichier au format xml (et non en rajouter).

- Pas très dynamique.

Cependant, garder à l'œil xslt pour la transformation d'un fichier xmi « brut » en un fichier xmi « affichable ».

### Code Java

L'application d'un patron se ferait directement sur les objets PMUC, Class, Association.

Peut être fait de deux manières :

Une classe par patron, qui se chargerait d'appliquer la transformation au PMUC.

Une classe pour tous les patrons, qui en fonction du ou des paramètres donnés appliquerait tel ou tel patron.

Les plus:

- Dynamique.
  - Facile à implémenter (comprendre, modifier).
-

## Captures d'écran du site web de documentation

# Documentation projet MIPSI

## Menu

- [JavaDoc](#)
- [Architecture](#)
- [Le XMI](#)
- [ConceptualGraph](#)
- [PDMM](#)
- [PMUC](#)
- [Association](#)
- [Concept](#)
- [Menu](#)
- [Transformations](#)
- [Diagramme de classes](#)
- [Accès document](#)
- [Liens utiles](#)
- [Prefuse](#)

## Documentation

Ce petit site permet de regrouper toute la documentation concernant le projet.

Une bonne idée serait de l'enrichir au fur et à mesure que l'application évolue afin de faciliter la maintenance et l'évolution de l'application.

Dernière modification : 15/05/2009

# Documentation projet MIPS

**Menu**

- [Accueil](#)
- [JavaDoc](#)
- [Architecture](#)
- [Le XMI](#)
- [ConceptualGraph](#)
- [PDMM](#)
- [PMUC](#)
- [Association](#)
- [Concept](#)
- [Menu](#)
- [Transformations](#)
- [Diagramme de classes](#)
- [Accès document](#)
- [Liens utiles](#)
- [Prefuse](#)

## Le graphe conceptuel

Le graphe conceptuel est un des piliers de la méthode. Pour pouvoir le charger (représenter les concepts et les liens sous forme d'objets) dans notre application, il est stocké dans un fichier xmi. Ce fichier xmi est lu à l'aide de l'API java JDom, et les objets Link et Concept sont construits à partir des informations lues.

```

    graph TD
        UOUnitOfWork((Unité de travail))
        COTCategoryOfTime((Catégorie d'unité de temps))
        CUDTCategoryOfUnitOfWorkTime((Composition de catégories d'unité de temps))
        CUDTCategoryOfTime((Composition d'unités de temps))
        UTSSequence((Unité de travail séquence))
        UTParallel((Unité de travail parallèle))
        CUDTCategoryOfWork((Composition d'unités de travail))
        CUDTCategoryOfWorkTime((Composition de catégories d'unité de travail))
        CUDTCategoryOfProduct((Composition de catégories de produit))
        CUDTCategoryOfProductCategory((Composition de catégories de travail))
        Rôle((Rôle))
        Alternative((Alternative))
        Condition((Condition))
        NATURE((NATURE))
        MAP((MAP))
        Stratégie((Stratégie))
        Argument((Argument))
        Problème((Problème))
        Intention((Intention))
        Contexte((Contexte))
        Situation((Situation))
        EtatTransition((Etat-Transition))
        CUDTCategoryOfIntention((Composition d'intentions))
        CUDTCategoryOfProduit((Composition de produits))
        CUDTCategoryOfProduitCategory((Catégorie de produit))

        UOUnitOfWork -- "Précision" --> COTCategoryOfTime
        UOUnitOfWork -- "Précision" --> CUDTCategoryOfTime
        UOUnitOfWork -- "Précision" --> CUDTCategoryOfWork
        UOUnitOfWork -- "Précision" --> CUDTCategoryOfWorkTime
        UOUnitOfWork -- "Précision" --> CUDTCategoryOfProduct
        UOUnitOfWork -- "Précision" --> CUDTCategoryOfProductCategory
        UOUnitOfWork -- "Précision" --> Rôle
        UOUnitOfWork -- "Précision" --> Alternative
        UOUnitOfWork -- "Précision" --> Condition
        UOUnitOfWork -- "Précision" --> NATURE
        UOUnitOfWork -- "Précision" --> MAP
        UOUnitOfWork -- "Précision" --> Stratégie
        UOUnitOfWork -- "Précision" --> Argument
        UOUnitOfWork -- "Précision" --> Problème
        UOUnitOfWork -- "Précision" --> Intention
        UOUnitOfWork -- "Précision" --> Contexte
        UOUnitOfWork -- "Précision" --> Situation
        UOUnitOfWork -- "Précision" --> EtatTransition

        COTCategoryOfTime -- "Précision" --> CUDTCategoryOfTime
        CUDTCategoryOfTime -- "Précision" --> UTSSequence
        CUDTCategoryOfTime -- "Précision" --> UTParallel
        CUDTCategoryOfTime -- "Précision" --> CUDTCategoryOfWork
        CUDTCategoryOfTime -- "Précision" --> CUDTCategoryOfWorkTime
        CUDTCategoryOfTime -- "Précision" --> CUDTCategoryOfProduct
        CUDTCategoryOfTime -- "Précision" --> CUDTCategoryOfProductCategory
        CUDTCategoryOfTime -- "Précision" --> Rôle
        CUDTCategoryOfTime -- "Précision" --> Alternative
        CUDTCategoryOfTime -- "Précision" --> Condition
        CUDTCategoryOfTime -- "Précision" --> NATURE
        CUDTCategoryOfTime -- "Précision" --> MAP
        CUDTCategoryOfTime -- "Précision" --> Stratégie
        CUDTCategoryOfTime -- "Précision" --> Argument
        CUDTCategoryOfTime -- "Précision" --> Problème
        CUDTCategoryOfTime -- "Précision" --> Intention
        CUDTCategoryOfTime -- "Précision" --> Contexte
        CUDTCategoryOfTime -- "Précision" --> Situation
        CUDTCategoryOfTime -- "Précision" --> EtatTransition

        CUDTCategoryOfWork -- "Précision" --> CUDTCategoryOfWorkTime
        CUDTCategoryOfWork -- "Précision" --> CUDTCategoryOfProduct
        CUDTCategoryOfWork -- "Précision" --> CUDTCategoryOfProductCategory
        CUDTCategoryOfWork -- "Précision" --> Rôle
        CUDTCategoryOfWork -- "Précision" --> Alternative
        CUDTCategoryOfWork -- "Précision" --> Condition
        CUDTCategoryOfWork -- "Précision" --> NATURE
        CUDTCategoryOfWork -- "Précision" --> MAP
        CUDTCategoryOfWork -- "Précision" --> Stratégie
        CUDTCategoryOfWork -- "Précision" --> Argument
        CUDTCategoryOfWork -- "Précision" --> Problème
        CUDTCategoryOfWork -- "Précision" --> Intention
        CUDTCategoryOfWork -- "Précision" --> Contexte
        CUDTCategoryOfWork -- "Précision" --> Situation
        CUDTCategoryOfWork -- "Précision" --> EtatTransition

        CUDTCategoryOfProduct -- "Précision" --> CUDTCategoryOfProductCategory
        CUDTCategoryOfProduct -- "Précision" --> Rôle
        CUDTCategoryOfProduct -- "Précision" --> Alternative
        CUDTCategoryOfProduct -- "Précision" --> Condition
        CUDTCategoryOfProduct -- "Précision" --> NATURE
        CUDTCategoryOfProduct -- "Précision" --> MAP
        CUDTCategoryOfProduct -- "Précision" --> Stratégie
        CUDTCategoryOfProduct -- "Précision" --> Argument
        CUDTCategoryOfProduct -- "Précision" --> Problème
        CUDTCategoryOfProduct -- "Précision" --> Intention
        CUDTCategoryOfProduct -- "Précision" --> Contexte
        CUDTCategoryOfProduct -- "Précision" --> Situation
        CUDTCategoryOfProduct -- "Précision" --> EtatTransition

        CUDTCategoryOfProductCategory -- "Précision" --> Rôle
        CUDTCategoryOfProductCategory -- "Précision" --> Alternative
        CUDTCategoryOfProductCategory -- "Précision" --> Condition
        CUDTCategoryOfProductCategory -- "Précision" --> NATURE
        CUDTCategoryOfProductCategory -- "Précision" --> MAP
        CUDTCategoryOfProductCategory -- "Précision" --> Stratégie
        CUDTCategoryOfProductCategory -- "Précision" --> Argument
        CUDTCategoryOfProductCategory -- "Précision" --> Problème
        CUDTCategoryOfProductCategory -- "Précision" --> Intention
        CUDTCategoryOfProductCategory -- "Précision" --> Contexte
        CUDTCategoryOfProductCategory -- "Précision" --> Situation
        CUDTCategoryOfProductCategory -- "Précision" --> EtatTransition

        Rôle -- "Complétude" --> Alternative
        Rôle -- "Complétude" --> Condition
        Rôle -- "Complétude" --> NATURE
        Rôle -- "Complétude" --> MAP
        Rôle -- "Complétude" --> Stratégie
        Rôle -- "Complétude" --> Argument
        Rôle -- "Complétude" --> Problème
        Rôle -- "Complétude" --> Intention
        Rôle -- "Complétude" --> Contexte
        Rôle -- "Complétude" --> Situation
        Rôle -- "Complétude" --> EtatTransition

        Alternative -- "Complétude" --> Condition
        Alternative -- "Complétude" --> NATURE
        Alternative -- "Complétude" --> MAP
        Alternative -- "Complétude" --> Stratégie
        Alternative -- "Complétude" --> Argument
        Alternative -- "Complétude" --> Problème
        Alternative -- "Complétude" --> Intention
        Alternative -- "Complétude" --> Contexte
        Alternative -- "Complétude" --> Situation
        Alternative -- "Complétude" --> EtatTransition

        Condition -- "Complétude" --> NATURE
        Condition -- "Complétude" --> MAP
        Condition -- "Complétude" --> Stratégie
        Condition -- "Complétude" --> Argument
        Condition -- "Complétude" --> Problème
        Condition -- "Complétude" --> Intention
        Condition -- "Complétude" --> Contexte
        Condition -- "Complétude" --> Situation
        Condition -- "Complétude" --> EtatTransition

        NATURE -- "Complétude" --> MAP
        NATURE -- "Complétude" --> Stratégie
        NATURE -- "Complétude" --> Argument
        NATURE -- "Complétude" --> Problème
        NATURE -- "Complétude" --> Intention
        NATURE -- "Complétude" --> Contexte
        NATURE -- "Complétude" --> Situation
        NATURE -- "Complétude" --> EtatTransition

        MAP -- "Complétude" --> Stratégie
        MAP -- "Complétude" --> Argument
        MAP -- "Complétude" --> Problème
        MAP -- "Complétude" --> Intention
        MAP -- "Complétude" --> Contexte
        MAP -- "Complétude" --> Situation
        MAP -- "Complétude" --> EtatTransition

        Stratégie -- "Complétude" --> Argument
        Stratégie -- "Complétude" --> Problème
        Stratégie -- "Complétude" --> Intention
        Stratégie -- "Complétude" --> Contexte
        Stratégie -- "Complétude" --> Situation
        Stratégie -- "Complétude" --> EtatTransition

        Argument -- "Complétude" --> Problème
        Argument -- "Complétude" --> Intention
        Argument -- "Complétude" --> Contexte
        Argument -- "Complétude" --> Situation
        Argument -- "Complétude" --> EtatTransition

        Problème -- "Complétude" --> Intention
        Problème -- "Complétude" --> Contexte
        Problème -- "Complétude" --> Situation
        Problème -- "Complétude" --> EtatTransition

        Intention -- "Complétude" --> Contexte
        Intention -- "Complétude" --> Situation
        Intention -- "Complétude" --> EtatTransition

        Contexte -- "Complétude" --> Situation
        Contexte -- "Complétude" --> EtatTransition

        Situation -- "Abstraction" --> Condition
        Situation -- "Concrétisation" --> NATURE
        Condition -- "Abstraction" --> COTCategoryOfTime
        Condition -- "Concrétisation" --> CUDTCategoryOfTime
        Condition -- "Abstraction" --> CUDTCategoryOfWork
        Condition -- "Concrétisation" --> CUDTCategoryOfWorkTime
        Condition -- "Abstraction" --> CUDTCategoryOfProduct
        Condition -- "Concrétisation" --> CUDTCategoryOfProductCategory
        Condition -- "Abstraction" --> Rôle
        Condition -- "Concrétisation" --> Alternative
        Condition -- "Abstraction" --> Condition
        Condition -- "Concrétisation" --> NATURE
        Condition -- "Abstraction" --> MAP
        Condition -- "Concrétisation" --> Stratégie
        Condition -- "Abstraction" --> Argument
        Condition -- "Concrétisation" --> Problème
        Condition -- "Abstraction" --> Intention
        Condition -- "Concrétisation" --> Contexte
        Condition -- "Abstraction" --> Situation
        Condition -- "Concrétisation" --> EtatTransition

        NATURE -- "Abstraction" --> MAP
        NATURE -- "Concrétisation" --> Stratégie
        NATURE -- "Abstraction" --> Argument
        NATURE -- "Concrétisation" --> Problème
        NATURE -- "Abstraction" --> Intention
        NATURE -- "Concrétisation" --> Contexte
        NATURE -- "Abstraction" --> Situation
        NATURE -- "Concrétisation" --> EtatTransition

        MAP -- "Abstraction" --> Stratégie
        MAP -- "Concrétisation" --> Argument
        MAP -- "Abstraction" --> Problème
        MAP -- "Concrétisation" --> Intention
        MAP -- "Abstraction" --> Contexte
        MAP -- "Concrétisation" --> Situation
        MAP -- "Abstraction" --> EtatTransition

        Stratégie -- "Abstraction" --> Argument
        Stratégie -- "Concrétisation" --> Problème
        Stratégie -- "Abstraction" --> Intention
        Stratégie -- "Concrétisation" --> Contexte
        Stratégie -- "Abstraction" --> Situation
        Stratégie -- "Concrétisation" --> EtatTransition

        Argument -- "Abstraction" --> Problème
        Argument -- "Concrétisation" --> Intention
        Argument -- "Abstraction" --> Contexte
        Argument -- "Concrétisation" --> Situation
        Argument -- "Abstraction" --> EtatTransition

        Problème -- "Abstraction" --> Intention
        Problème -- "Concrétisation" --> Contexte
        Problème -- "Abstraction" --> Situation
        Problème -- "Concrétisation" --> EtatTransition

        Intention -- "Abstraction" --> Contexte
        Intention -- "Concrétisation" --> Situation
        Intention -- "Abstraction" --> EtatTransition

        Contexte -- "Abstraction" --> Situation
        Contexte -- "Concrétisation" --> EtatTransition
    
```

Le chargement du graphe est effectué par le contrôleur CtrLoadConceptualGraph. Pour comprendre la démarche adoptée par ce contrôleur, se rendre dans la section [xmi](#).

## Plannings

### Planning prévisionnel

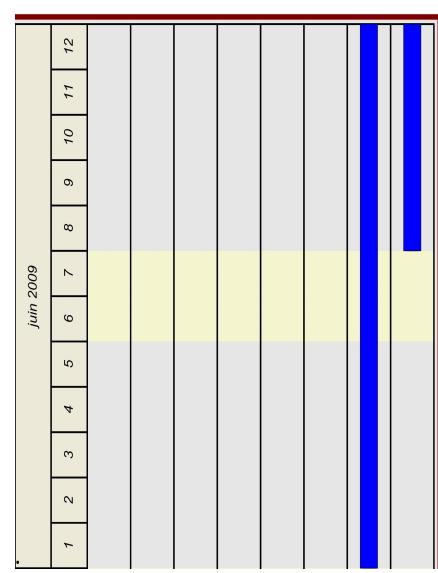
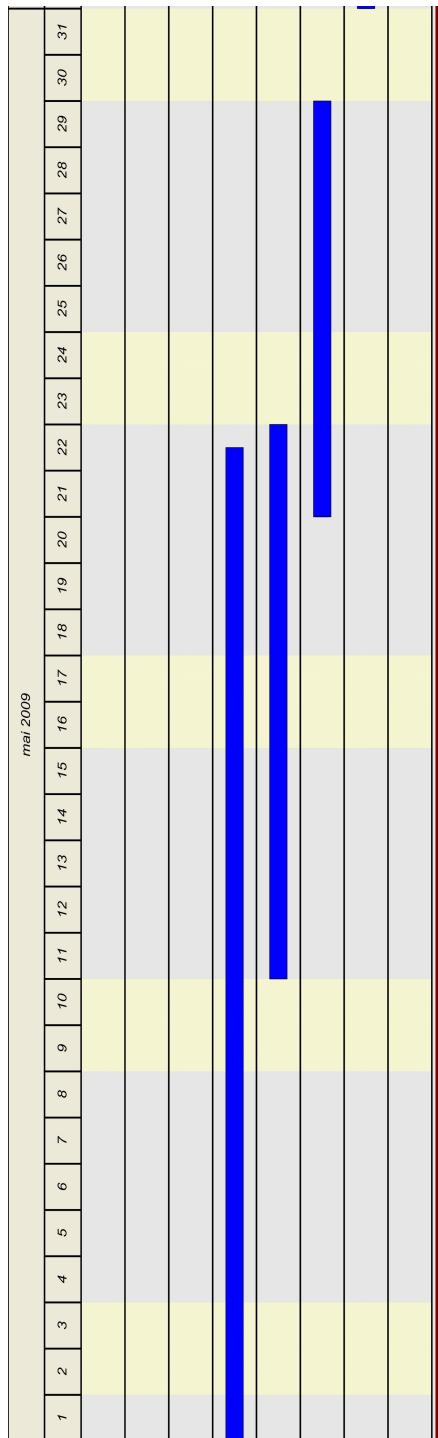
ID	Nom de tâche	Début	Terminer	Durée	avr. 2009																											
					6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30			
1	Lecture CDC - Articles	06/04/2009	10/04/2009	5j																												
2	Etude technique et analyse	14/04/2009	28/04/2009	11j																												
3	Rédaction dossiers technique et spécifications	14/04/2009	12/06/2009	44j																												
4	Programmation	29/04/2009	05/06/2009	28j																												
5	Rédaction Rapport de stage	06/04/2009	12/06/2009	50j																												

ID	Nom de tâche	Début	Terminer	Durée	mai 2009																											
					4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
1	Rédaction dossiers technique et spécifications	14/04/2009	12/06/2009	44j																												
2	Programmation	29/04/2009	05/06/2009	28j																												
3	Rédaction Rapport de stage	06/04/2009	12/06/2009	50j																												

ID	Nom de tâche	Début	Terminer	Durée	juin 2009																												
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	
1	Rédaction dossiers technique et spécifications	14/04/2009	12/06/2009	44j																													
2	Programmation	29/04/2009	05/06/2009	28j																													
3	Rédaction Rapport de stage	06/04/2009	12/06/2009	50j																													

Planning réel

ID	Nom de tâche	Début	Terminer	Durée	avr. 2009																											
					6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30			
1	Lecture CDC - Articles	06/04/2009	17/04/2009	10j																												
2	Etude technique et analyse	08/04/2009	22/04/2009	10j 4h																												
3	Rédaction dossier technique et spécifications	10/04/2009	24/04/2009	11j																												
4	Programmation	27/04/2009	22/05/2009	19j																												
5	Rédaction documentation	11/05/2009	22/05/2009	10j																												
6	Rédaction rapport de stage	21/05/2009	29/05/2009	7j																												
7	Début maquettage IHM	01/06/2009	12/06/2009	10j																												
8	Transfert de compétences	08/06/2009	12/06/2009	5j																												



## Explication des différences entre le planning prévisionnel et le planning réel

### Tâche n°1 : Lecture cahier des charges et articles

Le planning prévisionnel prévoyait une semaine, j'en ai en réalité passé deux. En effet, le sujet était difficile à cerner et j'ai décidé de commencer l'étude technique et l'analyse en parallèle.

### Tâche n°2 : Étude technique et analyse

J'ai décidé de commencer cette tâche plus tôt que prévu car elle m'a aidait dans la compréhension du sujet. Afin de ne pas perdre de temps, j'ai aussi commencé les spécifications en parallèle. Cela me permettait de bien voir dans quel contexte telle ou telle technologie serait utilisée.

### Tâche n°3 : Rédaction dossier technique et spécifications

Cette tâche a elle aussi commencé plus tôt que prévu. En effet, je me forçais à régulièrement faire des diagrammes, même si ceux-ci n'étaient que des brouillons, ils me permettaient de synthétiser mes idées et d'en discuter avec ma tutrice. Cette tâche s'est finit bien avant la date prévue, car, pour des raisons de clarté, la suite de cette tâche a été renommée en « Rédaction de la documentation ».

### Tâche n°4 : Programmation

La programmation a débuté légèrement plus tôt que prévu (deux jours) car les spécifications étaient finies et validées par ma tutrice.

### Tâche n°5 : Rédaction documentation

La rédaction de la documentation est la suite de la tâche nommée « Rédaction dossier technique et spécifications ». Elle a commencé une fois que la programmation du logiciel était bien avancée et a duré environ deux semaines.

### Tâche n°6 : Rédaction rapport de stage

Bien que toutes mes spécifications et les rapports fait durant le stage m'ont été très utiles pour la rédaction de mon mémoire, j'ai réellement entamé cette tâche bien plus tard que prévu.

### Tâche n°7 et 8 : Maquettage IHM et transfert de compétence

Ayant atteint les objectifs qui m'avaient été fixés plus tôt que prévu, je devrai commencer le maquettage de l'IHM dès la semaine du 1 Juin. Enfin, lors de la dernière semaine de mon stage, un autre stagiaire viendra prendre le relais et je lui transmettrai mes compétences.

.