République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab, Blida
USDB

Faculté des Sciences
Département Informatique

Mémoire pour l'obtention
d'un diplôme Master en informatique

Option : Ingénierie des Logiciels

Sujet :

Instanciation automatique de SPEMOntology avec les modèles procédés logiciels décrit sous Eclipse Process Famework (EPF)

Présenté par :
MAHIOUT Kamel
MOUZAI Boualem

Promoteur : Mme F.AOUSSAT.

Devant le jury composé de:

Président Benslili
Examinateur Oussalah
Examinateur M. Mustapha

2010- 2011

Remerciements

Avant tout, nous remercions Allah le tout-puissant qui nous a aidées à réaliser ce projet.

Nous tenons à adresser nos vifs remerciements les plus sincères et toute notre reconnaissance à notre promotrice Mme F. Aoussat, pour nous avoir confié ce projet, pour sa disponibilité, ses conseils et ses encouragements durant toute la période de réalisation de ce projet.

Nous remercions tous les membres du jury pour avoir accepté de juger ce travail et d'honorer notre soutenance par leur présence.

Enfin, nous tenons à remercier tous ceux qui ont contribué de près ou de loin à notre formation intellectuelle.

Dédicaces

Nous dédions ce travail :

A nos très chers parents qui nous toujours soutenu particulièrement durant toutes nos années d'études,

A tous les enseignants et étudiants du département Informatique de l'Université de Blida, en particulier ceux de la promotion 2010-2011.

ملخص

يهدف هذا المشروع إلى إعطاء المثائل تلقائيا للأونطولوجيا SPEMontology
المكتوبة في لغة الأونطولوجيا OWL انطلاقا من نموذج عملية تحضير البرامج
EPF(Eclispe Process Framework) / XML

RESUME

Modéliser des procédés logiciels de qualité, exige de l'expérience et un savoir faire confirmés, acquis par des années de raffinement et de réflexion. Réutiliser les procédés logiciels déjà développés, testés, utilisés et qui ont fait leur preuve est l'un de nos principaux objectifs.

Ce PFE s'inscrit dans la continuité du travail réalisé par les étudiants de l'année passée (2009/2010) ; pour l'implémentation d'une nouvelle approche de réutilisation de procédés logiciels, l'approche se base sur une ontologie de domaine (SPEMontology) qui regroupe les concepts de base et les instances des modèles de procédés logiciels à réutiliser.

Afin de capturer l'expérience de ce domaine, l'instanciation de cette ontologie doit se faire à partir de modèles de procédés logiciels existants et surtout éprouvés, La capture de ces connaissances passe nécessairement par une phase de réingénierie, des analyseurs de modèles de procédés logiciels doivent être développés à cet effet. Une instanciation pertinente est celle qui permet d'identifier de manière unique chaque instance de l'ontologie.

ABSTRACT

Modeling software quality processes requires experience and a confirmed know-how, acquired through years of refinement and reflection. Reusing software processes already developed, tested, used and proven is one of our main objectives.
The work is a continuation of the work done by students last year (2009/2010), for implementing a new approach for software processes reusing, the approach is based on domain ontology (SPEMontology) which includes the basic concepts and the instances of existing software processes models.

To capture the experience in this field, the instantiation of this ontology should be done from existing software process models and especially proven to capture such knowledge is necessary for a phase of reengineering, analyzers, process models software must be developed for this purpose. A perfect instantiation is that can uniquely identify each instance of the ontology.

Sommaire

INTRODUCTION GENERALE

INTRODUCTION GENERALE

Modéliser de nouveaux procédés logiciels en phase avec les nouvelles pratiques, méthodes, outils de développements est une nécessité pour la réussite des projets de développement logiciels.

Un modèle de procédé logiciel est la description de l'enchainement des activités, des outils, ressources humaines et matériels utilisé pour le développement et la maintenance d'un produit logiciels. Les modèles de procédés logiciels doivent refléter et s'adapter à la réalité du développement. Modéliser des procédés logiciels de qualité est alors une garantie pour la réussite des projets de développement logiciel.

Pour la modélisation de procédés logiciels de qualité une approche pour la modélisation des procédés logiciels par la réutilisation a été proposée [42]. L'objectif est de prendre en comptes les réflexions et les pratiques testées et adoptées par les modélisations et les exécutions précédentes.

L'approche proposée repose sur deux points essentiels :

1. Capitaliser les connaissances via une ontologie de domaine : l'ontologie instanciée permettra de réutiliser le savoir faire du domaine des procédés logiciels à travers l'inférence de nouvelles connaissances.
2. La manipulation de ces connaissances procédées logiciels en tant qu'architectures logicielles.

L'ontologie « SPEMontology » étant réalisée, l'objectif de ce projet de fin d'étude est la réalisation d'une partie de l'approche. Cette partie concerne l'instanciation de l'ontologie avec des modèles SPEM. L'objectif est de capitaliser les connaissances à partir de modèles SPEM. Ce travail s'inscrit dans la continuité d'un sujet de l'année précédente (instanciation de SPEMOntology avec des modèles PBOOL+). La difficulté du travail est de comprendre et maitriser les modèles SPEM qui sont plus complet et plus complexes que les modèles PBOOL+.

Pour réaliser ce travail, nous structurons notre mémoire en cinq chapitres :

Le chapitre 1 présente le standard SPEM, nous introduisant la notion de procédé logiciel ensuite, nous décrivons le standard SPEM en donnant plusieurs définitions et aussi son méta-modèle.

Le chapitre 2 présente les concepts base concernant les ontologies, et certaines ontologies développées dans le domaine des procédés logiciels.

Le chapitre 3 présente EPF (Eclipse Process Framework), nous commençons par définir EPF (Eclipse Process Framework) ainsi que les différents concepts qui le composent ensuite nous prendrons un exemple pour le dérouler avec EPF afin de nous permettre de présenter son fonctionnement. A la fin, nous parleront le fichier XML résultat qui représente le modèles de procédé logiciel SPEM.

Le chapitre 4 présente la conception de l'application que nous avons développée pour l'instanciation de l'ontologie SPEMOntology.

Enfin, **le chapitre 5** est l'implémentation de la solution proposée lors de la conception, nous présentons l'application développée, les outils et les langages de programmation utilisés.
Nous terminons par une conclusion générale et les perspectives de recherches.

Chapitre 1

Le standard SPEM

1. Introduction

L'OMG (**O**bject **M**anagement **G**roup) a défini le standard **SPEM** : **S**oftware **P**rocess **E**ngineering **M**etamodel, Dans le but d'unifier les langages de description de procédés logiciels, un procédé logiciel (Software Process) est l'ensemble des activités, des équipes, des outils et des techniques dont le but est d'assurer le développement et la maintenance des systèmes logiciels, SPEM est un langage de modélisation permettant la description des procédés logiciels en utilisant la notation UML (**U**nified **M**odeling **L**anguage).

Il offre un cadre conceptuel pour la modélisation des processus et méthodes de développement et permet la description des concepts d'un large éventail de procédés logiciels, sans se spécialiser dans un type précis, tout en compte la diversité des procédés logiciels.

La version de SPEM utilisée dans SPEMOntology est la version 2.0, qui est organisée en sept packages. SPEMOntology est générée à partir des packages : Core, Process Structure, Managed Content, Method Content et Process With Methods.

Ce chapitre est organisé en trois parties. La première partie , nous introduisant la notion de procédé logiciel, dans la deuxième partie, nous décrivons le standard SPEM en donnant des plusieurs définitions et aussi son méta-modèle. Dans la troisième partie nous avons présenté la version SPEM 2.0 utilisée, en décrivant les différents packages qu'il comporte et les dépendances entre ces packages.

2. Notion de procédé logiciel

Le but des **Procédés Logiciels** (Software Processes) est de formaliser et d'automatiser la répétitivité de certaines suites de tâches du développement des logiciels.

Les procédés sont utilisés dans différents domaines et de différentes manières : dans le domaine du travail coopératif assisté par ordinateur, dans les entreprises (sous le nom de workflows), dans le domaine de la gestion de projet, et dans le domaine de l'orchestration de services web … etc. [1]

On peut définir les procédés logiciels comme étant une suite d'étapes réalisées dans un but donné qui servent à gérer et assister le développement logiciel. [1]

On appelle procédé logiciel (Software Process) l'ensemble des activités, des équipes, des outils et des techniques dont le but est d'assurer le développement et la maintenance des systèmes logiciels. [3]

Un procédé logiciel définit les tâches à réaliser, les rôles participants et les produits manipulés pour élaborer ou maintenir un système logiciel. [4]

La qualité des systèmes logiciels dépend de la qualité des procédés par lesquels ils sont créés. [5]

Nous retenons la définition suivante : un procédé logiciel est l'ensemble des activités, des équipes, des outils et des techniques dont le but est d'assurer le développement et la maintenance des systèmes logiciels.

Un procède logiciel peut être représentée à l'aide d'un **Langage de Modélisation de Procédé** (PML). Une représentation du procède de logiciel dans un PML est appelé un **modèle de procède logiciel**. [6]

modèle de procède logiciel. [6]

2.1 Classification des Procédés Logiciels [1]

Les procédés logiciels servent à gérer et assister le développement logiciel. Il existe plusieurs sous-classes de procédés logiciels :

2.1.1. Les Procédés Logiciels Exécutables

Le Procédé Logiciel Exécutable définit la manière dont le développement logiciel est organisé, géré, mesuré, assisté, et amélioré (indépendamment du type de support technologique choisi pour le développement).

2.1.2. Les Procédés Semi-Formels

Les procédés semi-formels sont des procédés logiciels non exécutables, qui ont pour but de décrire la façon dont le logiciel va être développé. Ils recommandent fortement l'utilisation de procédés pour formaliser et modéliser le développement logiciel.

2.1.3. Les Procédés Organisationnels

Les procédés organisationnels définissent les niveaux de maturité, et décrivent les éléments clés d'un procédé logiciel efficace. Ils ont prouvé l'importance de se baser sur un procédé lors du développement logiciel.

2.1.4. Les Procédés de Gestion de Configuration

La gestion de configuration est le contrôle de l'évolution des systèmes complexes.

2.2. Présentation de l'OMG

L'OMG (**O**bject **M**anagement **G**roup) est une organisation internationale de standardisation créée en avril 1989. Sa mission est de développer des spécifications standards pour l'industrie des logiciels, qui sont pérennes, fiables, offrant ainsi un cadre commun pour le développement et l'intégration des applications distribuées dans des environnements hétérogènes.

2.3. Modèles de procédés logiciels

L'objectif principal de la modélisation de procédés est d'expliciter les pratiques de développement pour pouvoir les étudier, les améliorer et les utiliser de manière répétitive, gérable et éventuellement automatisable.

Un modèle de procédé logiciel est la représentation explicite d'un procédé logiciel dans un langage de description des procédés logiciels. [4]

Le modèle de procédé logiciel comprend des activités de développement du logiciel et de maintenance, des activités de gestion de projet et d'assurance qualité, et des activités de méta-procédé. [1]

La modélisation d'un procédé logiciel est l'élaboration d'une description de ce procédé en utilisant un (ou plusieurs) modèle(s) de procédé. Dans cette définition, un modèle de procédé est une abstraction de procédé représentée par un Langage de Description de Procédés **PML** (**P**rocess **M**odeling **L**anguage). [5]

2.4. Langages des modèles des procédés logiciels (PML)

Un modèle de procédé doit être représenté dans un certain langage. Un langage de description de procédés est un formalisme de modélisation développé ou adapté pour décrire les procédés. Il définit les concepts dédiés aux procédés, et fournit une syntaxe et un système de notations pour représenter des modèles de procédé en utilisant ces concepts. [4]

Plusieurs langages de modélisation de procédés existent : Appele, PBOOL+,… Etc. Dans notre travail nous utilisons le PML EPF (Eclipse Process Framework).

3. Présentation du standard SPEM

SPEM est une norme de l'OMG dédiée à la description de procédés logiciels. SPEM propose un méta-modèle de langage de description de procédés basé sur UML. À ce jour, SPEM supporte la définition de modèles de procédé mais pas l'exécution de ces modèles. [17]

SPEM est un langage de modélisation adopté par l'OMG, utilisé pour la spécification des procédés logiciels. Il vise à offrir un cadre conceptuel pour modéliser, échanger, documenter, gérer et présenter les processus et méthodes de développement. [15]. Il définit les façons d'utiliser UML dans les projets logiciels et permet la création de modèles de processus. [13].

SPEM est une proposition de l'OMG qui définit un formalisme dédié à la description du processus de développement logiciel. Les outils basés sur SPEM sont des outils de rédaction et de personnalisation de processus. La planification et l'exécution d'un projet décrit avec SPEM ne rentrent pas dans le domaine de ce modèle. [10]

SPEM utilise la notation UML mais en la spécialisant pour le domaine des procédés logiciels. En UML, les mécanismes d'extension permettant d'étendre ce langage sont les

Profils qui réalisent le rapprochement et la spécialisation d'UML à un domaine spécifique. Ainsi, SPEM est un profil UML permettant de spécialiser UML pour décrire les concepts des procédés logiciels.

SPEM utilise la notation UML par conséquent, il possède une syntaxe graphique, de plus sa sémantique est exprimée en langage naturel c'est donc un langage semi formel, ce qui peut engendrer certaines ambiguïtés.

La syntaxe de SPEM permet de construire de nombreux modèles qui peuvent être en contradiction avec la sémantique exprimée en langage naturel dans sa spécification.

Dans [14], une solution a été proposée intégrant de nombreux ajouts syntaxiques, par la redéfinition de relations pour clarifier les concepts utilisés et pour compléter la formalisation de sa sémantique dans le but d'avoir un méta-modèle clair et facilement compréhensible. Ces redéfinitions ont été décrites sémantiquement à l'aide de contraintes OCL. Aussi, des contraintes OCL ont été ajoutées limitant les instances valides du méta modèle SPEM. Les propriétés exprimées s'appliquent sur tous les modèles de procédés.

3.1 Le modèle Conceptuel de SPEM

Conceptuellement, SPEM repose sur l'idée qu'un procédé de développement logiciel est une collaboration entre des entités actives et abstraites appelées **rôles** qui exécutent des **activités** sur des entités concrètes et réelles, **les produits.** [16] [17]

Chaque activité a besoin de *produits en entrée* et produit des *produits en sortie.* Ainsi les concepts de base de SPEM sont : Activité, Produit et Rôle ainsi que les relations entre ces concepts.

La figure 2.6 traduit ce modèle conceptuel fondamental en utilisant la notation UML. Cette figure ne fait pas partie du formalisme de SPEM, elle est donnée pour des raisons explicatives. [14][17].

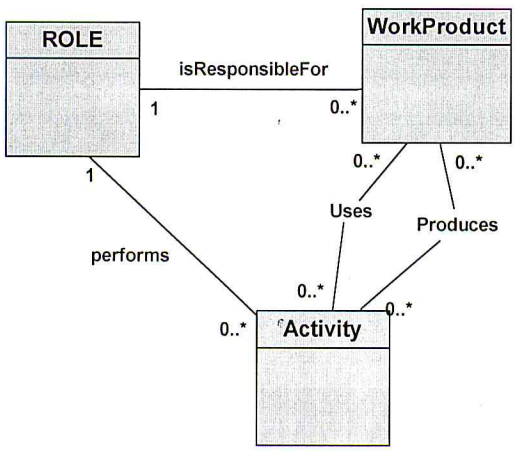


Figure 1.1 : Le modèle conceptuel de SPEM. [14].

Un **rôle** est une entité active et abstraite modélisant un ensemble de compétences d'un acteur concret d'un procédé qui n'est pas forcément un humain et en fonction de ses compétences il joue un ou plusieurs rôles, et réciproquement un rôle peut être peut être joué par un ou plusieurs acteurs.

Un rôle peut collaborer avec d'autres rôles, en échangeant des produits ou agir sur d'autres rôles, en déclenchant l'exécution de certaines activités. Il effectue des opérations : **activités** sur des **produits** qui sont des entités concrètes et réelles.

4. Présentation de SPEM 2.0

De [17] et [15] :

SPEM2.0 est défini selon une *vision attractive*. Celle-ci consiste à séparer les aspects *contenus* et *données* relatifs aux *méthodologies de développement*, de leurs possibles *instanciations* dans un *projet particulier*.

SPEM 2.0 sépare le contenu réutilisable d'une méthode de développement de son application dans les procédés. Le contenu d'un élément de méthode fournit des explications décrivant la réalisation des buts spécifiques du développement de façon indépendante d'un cycle de vie concret. Un procédé, représenté par une activité, prend ces éléments de contenu de méthode et les relient selon l'ordre d'exécution adapté aux besoins d'un type de projet spécifique. Cela permet à SPEM 2.0 de mieux supporter la réutilisation de procédés.

Ainsi, pour pleinement exploiter ce framework, la première étape devrait être de définir toutes les phases, activités, produits, rôles, guides, outils, etc., qui peuvent composer une méthode pour ensuite, dans une deuxième étape, choisir en fonction du contexte et du projet, le contenu de la méthode appropriée pour l'utiliser dans la définition du procédé.

SPEM2.0 est défini sous la forme d'un méta modèle MOF qui s'appuie sur les spécifications *UML2.0 Infrastructure* et *UML2.0 Diagram Interchange*. Il réutilise de UML2.0 infrastructure les concepts de base comme *Classifier* ou *Package*. Aucun concept d'UML2.0 Superstructure n'est réutilisé. Le standard est défini également sous la forme d'un profil UML où chaque élément du méta modèle de SPEM2.0 est défini comme un stéréotype d'UML2.0 Superstructure.

4.1 Structure du méta-modèle SPEM 2.0

De [15] et [18] :

Le méta modèle de SPEM2.0 est composé de sept packages liés avec le mécanisme ***merge*** (figure 2.7). Chaque package traite un aspect particulier et étend le package dont il dépend, lui fournissant une structure additionnelle et des possibilités de description des éléments en dessous.

SPEM est organisé en plusieurs packages afin, d'une part, de faciliter l'extension de SPEM, et d'autres part, d'augmenter la réutilisation des concepts décrit par SPEM que ce soit

structure des procédés logiciels, connaissances des méthodes de développement ou documentation et assistance.

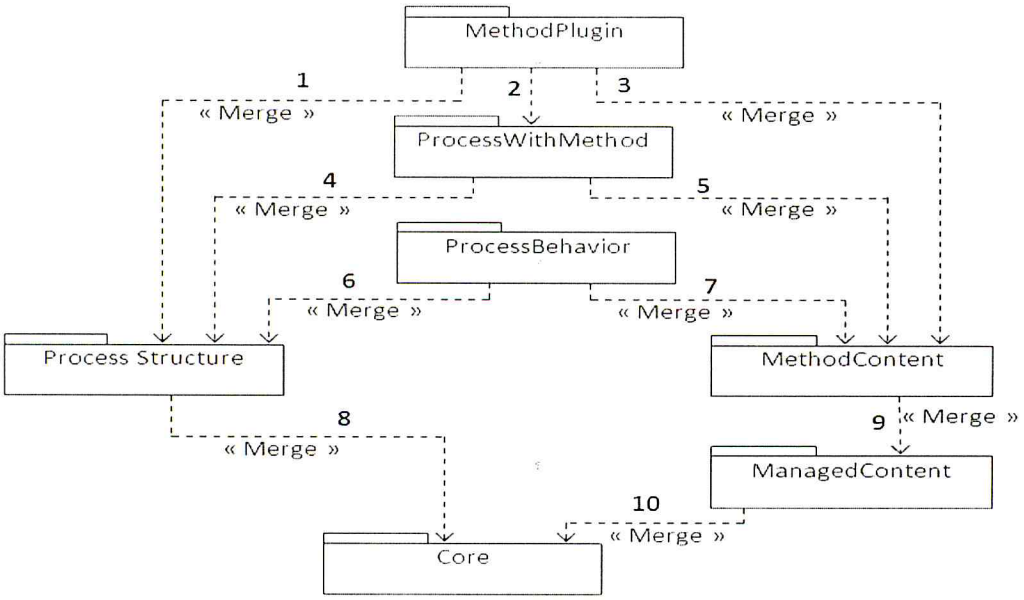


Figure 1.2 : Structure du méta-modèle SPEM 2.0. [14]

4.2 Le package Process Structure (Structure de procédé)

Le package **Process Structure** définit les éléments permettant de représenter les modèles de procédés en termes de flots d'activités (Activities) avec leurs utilisations de produit (WorkProductUses) et de rôle (RoleUses), il contient donc les éléments structuraux basiques pour décrire un procédé de développement qui n'est rien qu'un enchainement d'activités plus les éléments nécessaires pour leur accomplissement. Mais, ce package ne fournit pas les propriétés qui décrivent chaque élément.

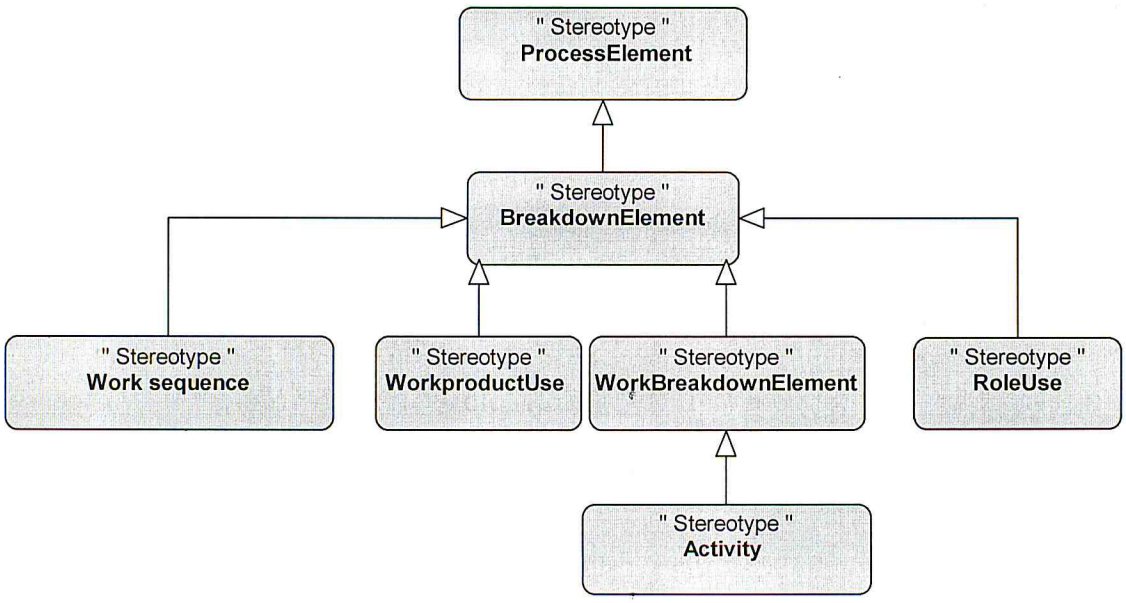


Figure 1.3 : Le noyau du package ProcessStructure [18]

La classe principale de ce package est la classe **« breakdownElement »** qui est la super classe des autres classes: **worksequence, WorkproductUse, WorkBreakdownElement, RoleUse.** Ces classes représentent les concepts activité, rôle et produit selon la vision « structure » d'où l'utilisation du mot « WorkBreakdownElement» pour activité, le suffixe « use » pour rôle et produit. L'enchainement des activités est important pour un fragment procédé logiciel, il est représenté par la classe **WorkSequence**.

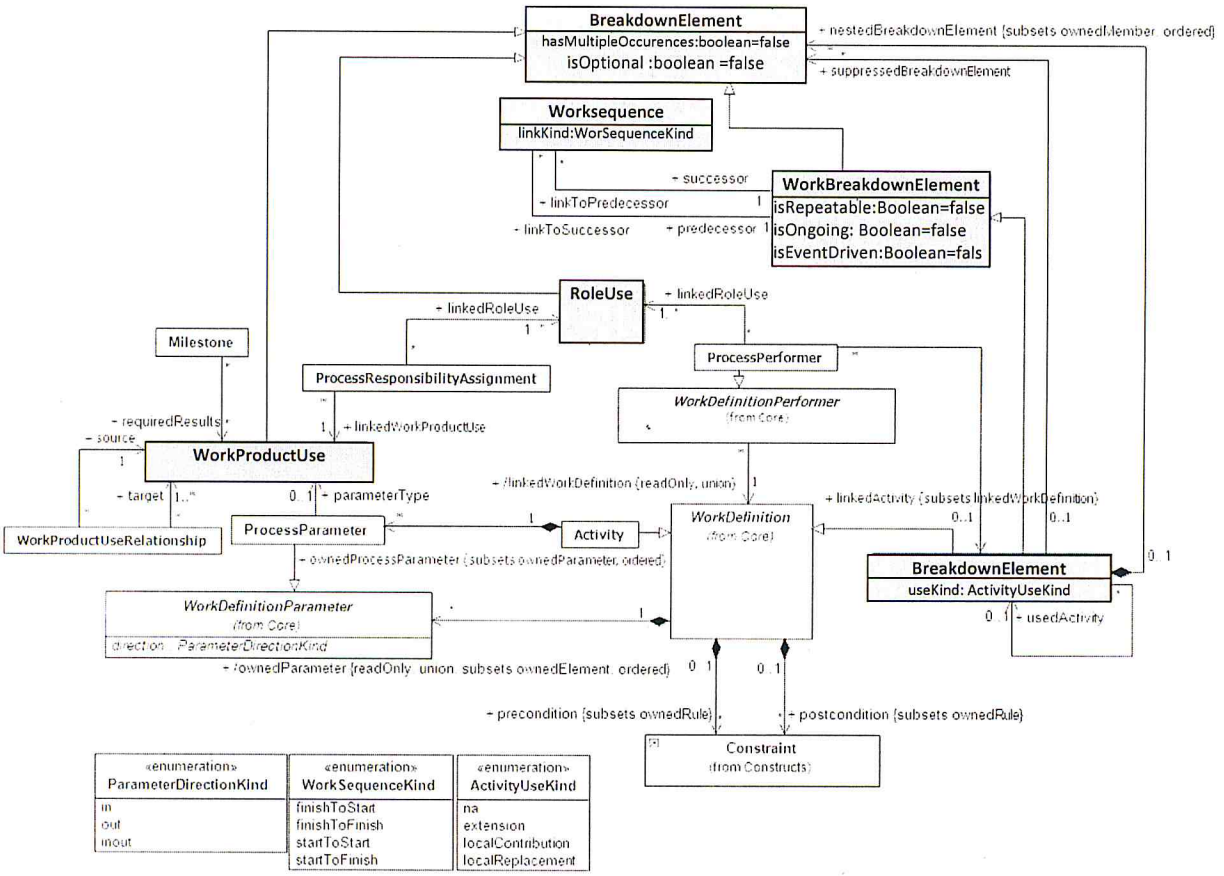


Figure 1.4 : Les principales classes et relations du package ProcessStructure [18].

4.3 Le package Managed Content (contenu géré)

Le package **Managed Content** fournit la possibilité de documenter textuellement les éléments définis dans le package Process Structure et les éléments du package Method Content, en fournissant les concepts de base pour gérer leurs descriptions textuelles.

Plusieurs méthodes de développements préfèrent décrire le procédé logiciel en langage naturel que de le décrire formellement. Elles considèrent que les procédés logiciels sont des procédés créatifs, et doivent être réévalués constamment. Aussi, elles considèrent que les meilleures pratiques de développement sont souvent communiquées à travers des descriptions en langage naturel sur le format papier, ce qui est le cas des méthodes agiles.

L'avantage de ce package est qu'il est possible d'utiliser ses concepts de manière autonome ou de les associer aux concepts du package Process Structure.

Ainsi, s'ils sont utilisés de manière autonome, un procédé logiciel basé SPEM est seulement un ensemble d'assistance décrivant les meilleures pratiques de développement logiciel indépendamment d'une structure définie. S'ils sont combinés avec des concepts du package Process Structure, le procédé logiciel décrit sera de plus, bien structuré.

La classe principale de ce package est la classe **DescribableElement**. Cette classe est une généralisation abstraite qui permet de représenter tout élément de procédés logiciels qui peut être documenté et décrit textuellement.

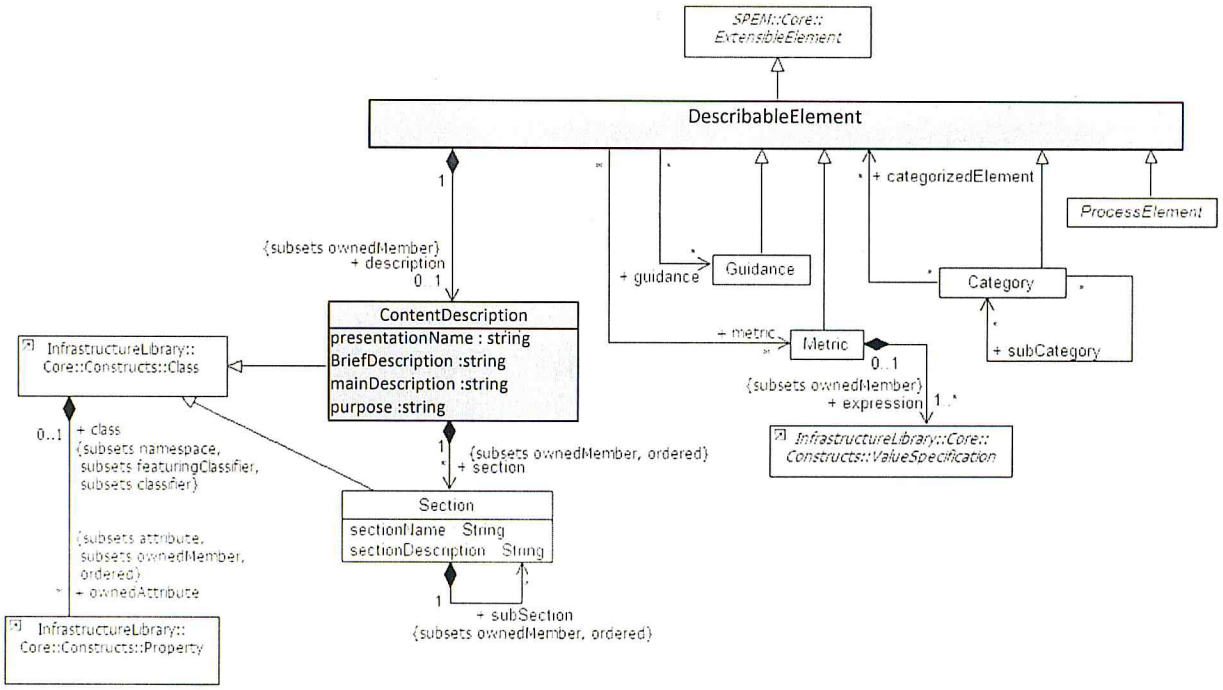


Figure 1.5 : Les principales classes du package ManageContent [18]

4.4 Le package Method Content (contenu méthode)

Le package **Method Content** permet de spécifier le contenu de méthodes basiques, comme les rôles (Roles), les tâches (Tasks) et les produits (WorkProducts) en définissant et en décrivant les éléments de base de chaque méthode.

Il fournit les concepts de base pour construire une base de connaissance indépendamment de procédés spécifiques et de projets de développement particuliers, car il intègre des concepts pour définir des cycles de vie et pour décrire des éléments réutilisables. Ces éléments sont réutilisables car ils fournissent la base pour décrire des méthodes, des techniques, de meilleures pratiques de développements logiciels sans prendre en compte un projet de développement spécifique et sont indépendants d'un procédé logiciel spécifique.

Ainsi, l'utilisateur peut réutiliser ces éléments en les organisant puis les personnaliser selon son projet. Des exemples de ces concepts sont les classes **ContentDescription** et **Guidance**.

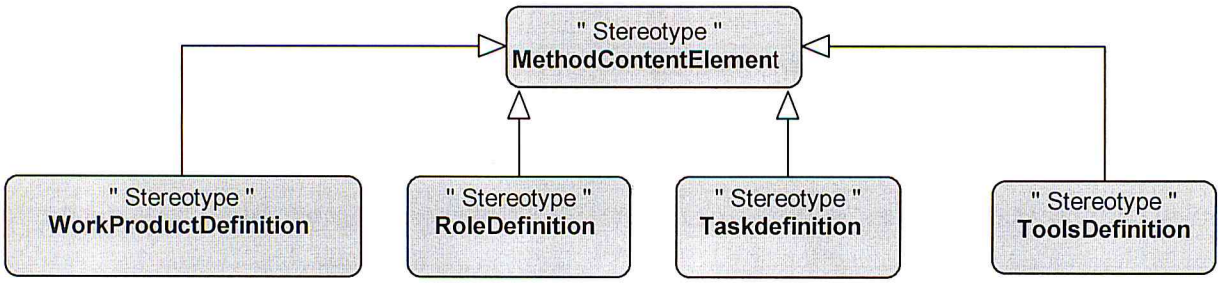


Figure 1.6: Le noyau du package MethodContent représentant les concepts de base du procédé logiciel.[18]

La classe de base de ce package est la classe **MethodContentElement**. Cette classe est la super classe des classes **WorkProductDefinition**, **RoleDefinition**, **TaskDefinition** et **ToolsDefinition** qui définissent les contenus des concepts activité, produits, rôle et outil d'où le suffixe *Definition*.

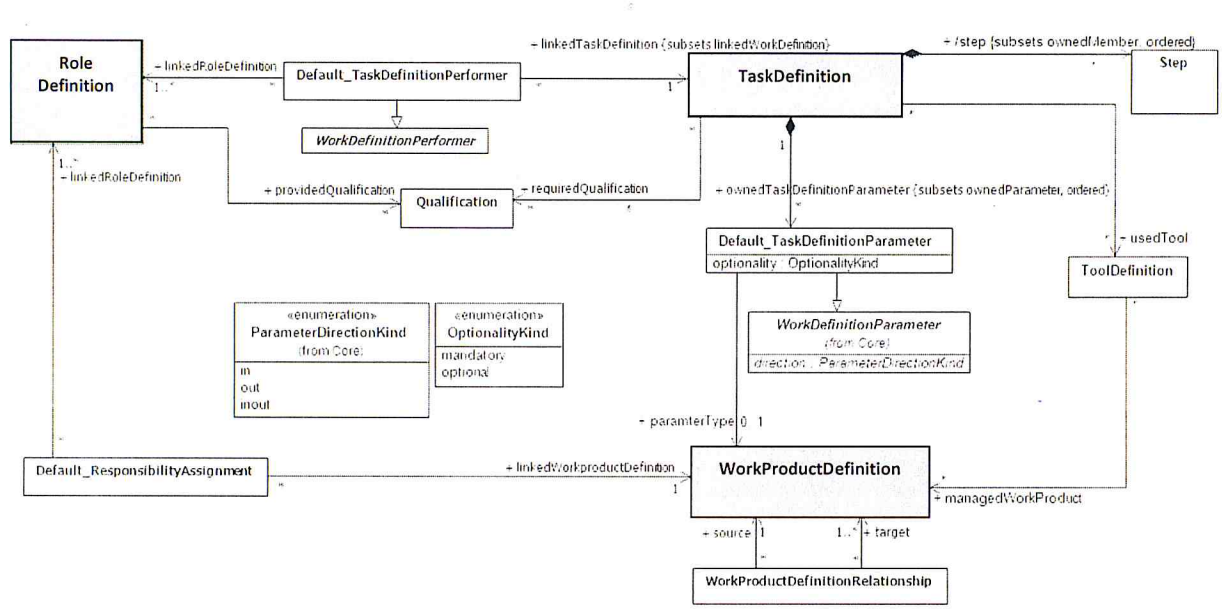


Figure 1.7: Les principales classes et relations du package MethodContent [18].

4.5 Le package Process With Methods (procédé avec méthode)

Le package **Process With Method** regroupe l'ensemble des éléments requis pour intégrer les processus définis avec les concepts du package Process Structure, avec les contenus définis avec les concepts du package MethodContent.

Ce package permet de décrire un procédé logiciel structuré en respectant les concepts du package Process Structure suivant une méthode de développement définie dans le package Content Method, plaçant le procédé logiciel dans un contexte particulier et en respectant un cycle de vie particulier.

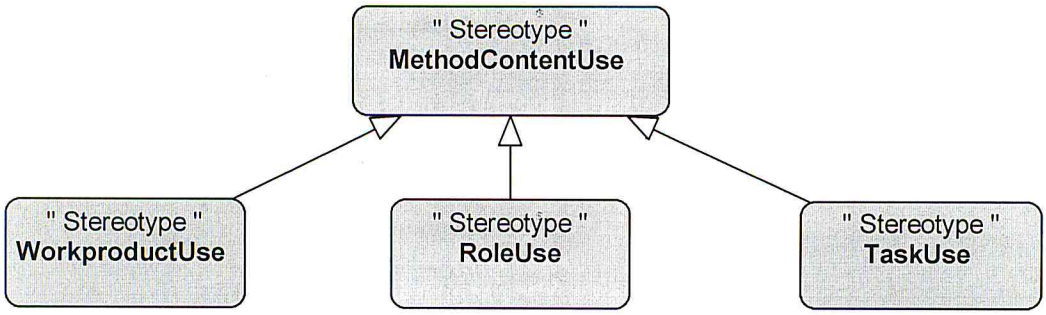


Figure 1.8 : Le noyau du package ProcessWithMethod représentant les concepts de base du procédé logiciel. [18]

La classe principale de ce package est la classe **MethodContetUse**. Cette classe est la super classe des classes **TaskUse**, **RoleUse** et **WorkproductUse**. Ces deux dernières héritent aussi de la classe BrekdownElement du package Process Structure.

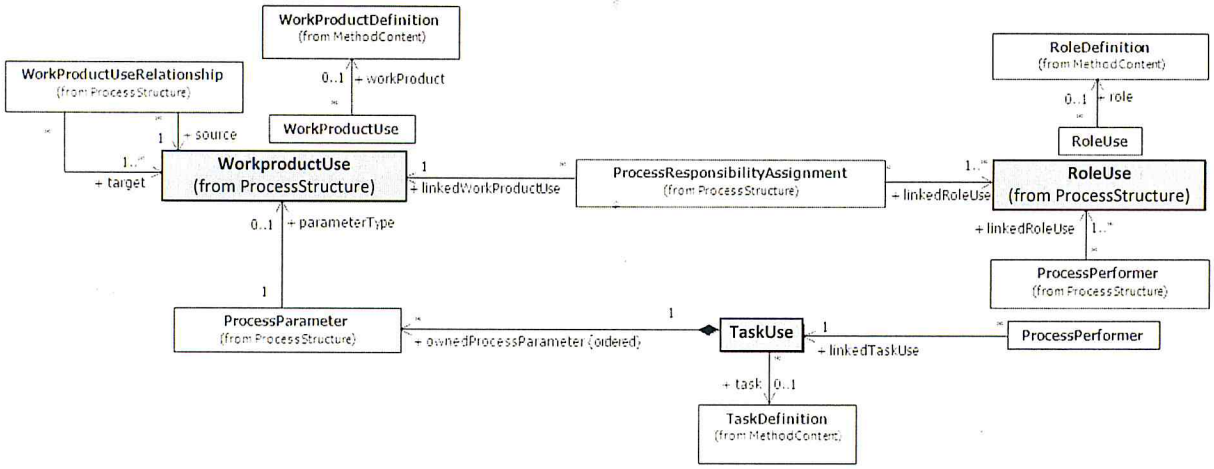


Figure 1.9: Les principales classes et relations du package ProcessWithMethod [18]

4.6 Dépendances entre les packages de SPEM 2.0 [13]

Après avoir décrit en quoi consiste chaque package, nous reprenons le digramme de la figure 2.7 pour expliquer les dépendances entre ces différents packages.

Dépendance 1 : Le package **Method Plugin** dépend du package **Process Structure** car pour gérer et réutiliser le contenu d'un procédé (fragment procédé), il faut connaitre sa structure qui est définie dans le package Process Structure.

Dépendance 2 : Le package **Method Plugin** dépend du package **Process With Method** car pour gérer et réutiliser le contenu d'un procédé, il faut connaitre le contenu de chacun de ses éléments. Pour chaque élément, son contenu lui est intégré par le package Process With Method.

Dépendance 3 : Le package **Method Plugin** dépend du package **Method Content** car il permet de gérer et réutiliser le contenu d'une méthode (rôle, tâche…) et ce contenu est spécifié par le package Method Content.

Dépendance 4 : Le package **Process With Method** dépend du package **Process Structure** car le package Process With Method consiste à intégrer aux éléments d'un procédé (ou fragment de procédé) définis dans le package Process Structure leurs contenus.

Dépendance 5 : Le package **Process With Method** dépend du package **Methode Content** car le package Process With Method consiste à intégrer aux éléments d'un procédé (ou fragment de procédé) définis dans le package Process Structure leurs contenus qui est définis dans le package Method Content.

Dépendance 6 : Le package **Process Behavior** dépend du package **Process Structure** car il permet de lier un élément de procédé avec un comportement externe, il faudra donc connaitre sa structure qui est définie dans le package Process Structure.

Dépendance 7 : Le package **Process Behavior** dépend du package **Method Content** car il permet de lier un élément de procédé avec un comportement externe, il faudra donc connaitre son contenu qui est défini dans le package Method Content.

Dépendance 8 : Le package **Process Structure** dépend du package **Core** car ce dernier définit tous les concepts qui permettent l'abstraction des classes du package Process Structure.

Dépendance 9 : Le package **Method Content** dépend du package **Managed Content** car le package Method Content spécifie le contenu d'une méthode et cela nécessite la description de cette dernière et celle ci est fournie par le package de Managed Content.

Dépendance 10 : Le package **Managed Content** dépend du package **Core** car ce dernier définit tous les concepts qui permettent l'abstraction des classes du package **Managed Content.**

5. Conclusion

Nous avons introduit dans ce chapitre la notion des procédés logiciels. Par la suite nous avons présenté le profil UML SPEM défini par l'OMG, qui spécialise le méta-modèle UML pour décrire les concepts des procédés logiciels permettant ainsi de modéliser les procédés logiciels en utilisant la notation UML.

Ensuite nous avons présenté la version actuelle du standard SPEM : **SPEM 2.0** que nous utilisons dans notre travail, son organisation et les relations entre les différents packages qu'il comporte.

Notre travail consiste à instancier automatiquement l'ontologie SPEMOntology qui regroupe les concepts du méta-modèle SPEM. Dans le chapitre qui suit nous allons décrire le concept d'ontologie et présenter l'ontologie OWL SPEMOntology.

Chapitre 2
Ontologie et SPEMOntology

1. Introduction :

Nous présentons par ce présent chapitre le concept d'ontologie, nous expliquerons la nécessité d'introduire les ontologies dans le domaine informatique et présentons les définitions proposées, nous procéderons par la suite à démontrer la différence entre le concept d'ontologie et les bases de connaissances, les diagrammes de classes.

En outre, nous énumérons les éléments qui constituent une ontologie tout en montrant les objectifs visés par cette dernière dans les différents modèles de l'informatique.

Puis, nous expliquerons les ontologies OWL (Ontology Web Language) tout en mettant l'accent sur le langage utilisé (OWL).

Enfin, nous procéderons à la description de l'ontologie SPEMontology déduite de la transformation du méta-model SPEM à une ontologie OWL en utilisant le langage de transformation de modèle ATL (ATLAS Transformation Language), d'où nous présentons les différents paquets (packages) de l'ontologie SPEMontology, et les concepts que nous allons instancier.

2. Notion d'ontologie :

2.1 L'apparition des ontologies en informatique :

Alors que les systèmes experts résolvaient automatiquement les problèmes , apparurent par la suite les systèmes à base de connaissances qui permettaient le stockage , la consultation, et la modification des connaissances , le raisonnement automatique sur les connaissances stockées ; Et avec le développement considérable des réseaux qui permettent le partage des connaissances entre les systèmes informatiques , l' expérience de leur développement a mis en évidence la complexité et la longue durée du processus de la construction d'une base de connaissance . [19]

De ce fait, le besoin de la réutilisation et le partage des bases de connaissances ou des parties de ces dernières se fait impérativement sentir, trois scénarios ont été proposés :

- Réutiliser une base de connaissances pour étendre la classe de problèmes pouvant être résolus au sein d'un SBC (Système de base de connaissances) .
- Concevoir un nouveau système réalisant des tâches différentes mais dans le même domaine.
- Faire appel à des compétences d'autres systèmes au cours d'un raisonnement.

Mais, ce propositions trouvent plusieurs difficultés car les systèmes de base de connaissances sont présentés dans des langages différents, par leurs structures de données (langage de règles, langage à objet) et par leurs mécanismes d'inférence.

Aussi , la diversité des termes est considérable , même dans les systèmes de base de connaissances réalisant des activités similaires , et parfois employés dans des sens différents d'où la difficulté de la réutilisation et le partage des bases de connaissances , c'est alors qu'au début des années 90 des chercheurs réunis au sein du projet Américain KNOWLEDGE

SHARING EFFORT ont mis en évidence leur représentation explicite du sens qu'ils nomment ontologie .

Comme la représentation des connaissances sous forme de règles logiques utilisées dans les systèmes experts s'avèrent insuffisante, de nombreux formalismes ont été introduits pour modéliser la richesse sémantique des connaissances au niveau conceptuel tel que : les langages à base de frames, les logiques de descriptions …etc., ces langages permettent la représentation des concepts sous-jacents à un domaine de connaissances, les relations qui les lient et les sémantiques de ces dernières indépendamment de l'usage que l'on souhaite établir sur ces connaissances. [19]

2.2 Définition d'une ontologie :

D'après la philosophie grecque, le terme ontologie signifie : **explication systématique de l'existence**, l'étude de ce qui existe dans le monde [20].

En informatique et en science de l'information l'ontologie est une représentation des connaissances du monde au niveau du mode conceptuel qui par la suite permettra le raisonnement automatique sur les objets du domaine concerné.

Les ontologies permettent la compréhension des connaissances dans un domaine d'une façon générale et de fournir par la suite une représentation communément acceptée qui pourra être utilisée et partagée par diverses applications [20].

Réutilisable : cette représentation est faite de façon déclarative, cela veut dire qu'elle est indépendante de la manière de l'usage de ces connaissances.

Partageable : cette représentation est établie sur la base des concepts qui caractérisent un domaine ainsi que sur des concepts fondamentaux communs, qui sont utilisables à travers divers domaines, ce qui permet la communication entre les systèmes d'information qui doivent partager des informations basées sur des concepts communs [21].

3. Composant d'une ontologie :

Une ontologie est un ensemble de concepts dans un domaine donné réel ou imaginaire avec les relations établies entre ces derniers qui sont organisés dans un graphe dont les relations peuvent être **sémantiques** ou de **subsomption**. [21].

3.1 Concept :

Un concept est une entité qui peut représenter un objet, une notion ou une idée elle est composée de trois éléments :

1. **Un terme** représentant le concept en langue.
2. **Une notion (intension)** comprenant la sémantique du concept représenté en termes de propriétés et d'attributs, de contraintes et de règles.

Les règles qui décrivent les inférences possibles sont des affirmations représentées par : antécédent →conséquent

3. **L'extension** du concept regroupant tous les objets manipulés à travers le concept, ces objets sont appelés instances du concept.

Exemple : Soit le concept ordinateur.

Terme : Ordinateur

Intension : Un ordinateur est une machine dotée d'une unité de traitement lui permettant d'exécuter des programmes enregistrés, c'est un ensemble de circuits électroniques permettant de manipuler des données sous forme binaire ou bits.

Règle : Un ordinateur avec des performances élevées est cher

Extension : liste de tous les ordinateurs du monde.

Remarque : On peut désigner deux concepts partageant la même extension sans avoir la même intension, c'est-à-dire avoir des points de vue différents sur un même objet

Deux extensions peuvent ne pas être disjointes alors que deux intensions peuvent être différentes au moins par une propriété.

Exemple : Les mouches peuvent être considérées comme des insectes nuisibles ou comme appât de pêche.

Un concept peut être désigné par des termes différents comme un terme peut désigner plusieurs concepts ce qui crée une ambigüité.

Exemple : Le terme bureau pour un meuble et bureau pour une salle comme bureau du directeur.

L'ambigüité résultante d'un concept ayant plusieurs termes ne peut être gérée par la machine sauf si cette dernière est soumise à une restriction à un domaine de connaissances précis permettant ainsi d'éviter d'avoir des concepts différents désignés par un même terme, par contre la désignation d'un concept par plusieurs termes facilite l'utilisation de l'ontologie.

Un concept peut être défini à partir d'autres concepts par exemple le concept **voiture** se définit par les concepts **habitacle, moteur** et **roues** …etc.

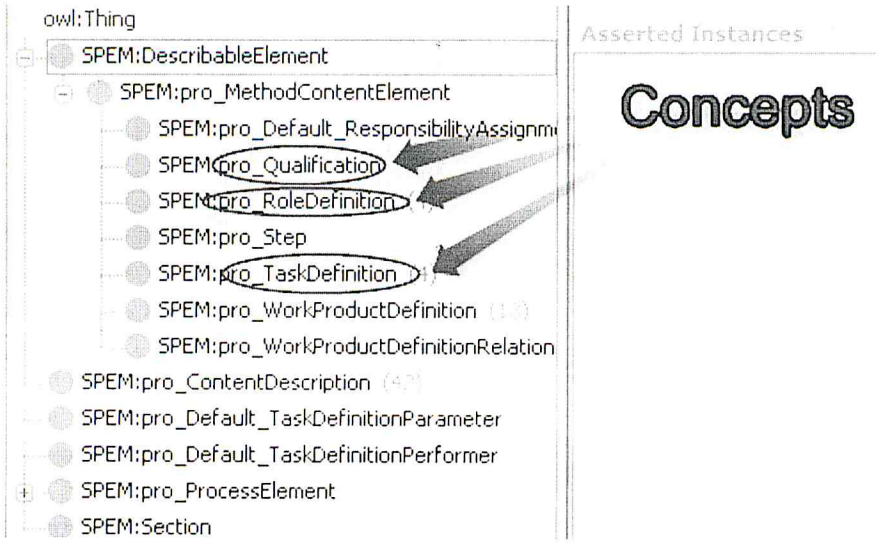


Figure 2.1 : Les Concepts(Classes).

3.2 Relations: [19]

On peut présenter des liens conceptuels existants entre les concepts suivant les propriétés de ces derniers, d'autres doivent être présentés par des relations autonomes.

Les relations peuvent être spécifiées par des propriétés tout comme pour les concepts, elles respectent une organisation hiérarchique imposée par la propriété de **subsomption.**

3.2.1 Relation de subsomption

La subsomption est une structure hiérarchique dotée de propriétés conceptuelles et qui lient deux concepts.

Un concept C1 subsume un concept C2 si toute propriété sémantique de C1 est aussi une propriété sémantique de C2, c'est-à-dire C2 est plus spécifique que C1 et l'extension C2 est forcément plus réduite que celle de C1, par contre son intension est plus riche.

3.2.2 Liens conceptuels

Les relations permettent la liaison des instances de concepts ou les concepts génériques, elles sont caractérisées par un **terme** ou plusieurs, et une **signature** qui précise le nombre d'instances de concepts liés par ces relations, leurs types et la façon de lire ces relations.

Une relation a des propriétés qui sont ou bien des cardinalités représentant le nombre des relations entre les même concepts ou instances de concepts, ou des propriétés algébriques telles que la symétrie, la réflexivité et la transitivité.

Nous distinguons deux types de propriétés :

- Propriétés qui lient deux relations : incompatibilité et inverse.
- Propriétés qui lient deux concepts : équivalence, disjonction, et dépendance.

3.3 Fonctions :

Elles sont des cas particuliers de relations dans lesquelles le $n^{ième}$ élément de la relation est défini à partir des n-1 premiers [20].

3.4 Axiomes

On utilise les axiomes pour formuler des phrases qui ont la valeur de vérité toujours à vrai, leur utilisation permet représenter les intensions de concepts et des relations dans un domaine, et spécifier comment utiliser les concepts et les relations qui les lient pour une meilleure expression des connaissances dans le domaine.

Certains sont particuliers, on les trouve dans plusieurs ontologies par exemple la relation **carré de** est l'inverse de la relation **racine de .**D'autres sont propres à un seul domaine, ils définissent la sémantique du domaine, prenant comme exemple dans la géométrie, on l'axiome : il n'existe pas plus d'un triangle couvrant trois points. [20][27].

3.5 Instances : De [19]

On utilise les instances pour représenter des éléments dans un domaine. Une classe peut avoir plusieurs instances partageant les mêmes propriétés mais ils doivent être nommés différemment.

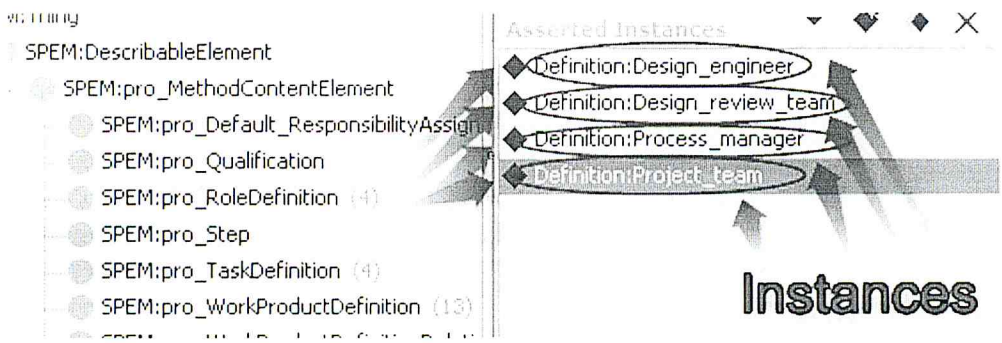


Figure 2.2 : Les instances des concepts.

4. Rôle des ontologies

De [21], [28], [29] et [30] :

Une ontologie est principalement développée afin :

- De permettre la réutilisation du savoir sur un domaine, c'est l'une des principales raisons qui ont poussé la recherche sur les ontologies ces dernières années.
- D'acquérir et de représenter les connaissances, puisqu'une ontologie permet de rassembler toutes les définitions des termes d'un domaine donné.

- De distinguer le savoir sur un domaine du savoir opérationnel, vu qu'une ontologie permet la représentation des connaissances d'une façon déclarative indépendamment de la manière dans ces dernières vont être utilisées.
- D'expliciter ce qui est implicite sur un domaine ; les ontologies offrent des spécifications explicites du savoir sur un domaine pour les nouveaux utilisateurs qui doivent apprendre la signification des termes du domaine.
- D'analyser le savoir sur un domaine, ceci est fait une fois que la spécification des termes du domaine est effectuée. L'analyse est importante pour la réutilisation ou l'extension des ontologies.
- Partager la compréhension commune de la structure des informations entre les personnes et les fabricants de logiciels, Les ontologies permettent le partage de la compréhension et la communication dans des contextes particuliers, elle peut être utilisée pour créer un réseau de relations, qui définit les connexions entre les composants d'un système. La communication est assurée grâce à la **non-ambiguïté** des termes utilisés et définis par l'ontologie dans les systèmes

5. Les ontologies OWL

5.1. Présentation du langage OWL [13]

Le **W3C** (**W**orld **W**ide **W**eb **C**onsortium) propose une hiérarchie des langages pour le Web sémantique. (figure 2.3).

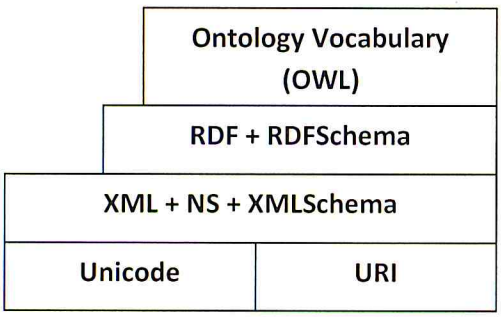


Figure 2.3 : Une partie de l'Architecture du Web sémantique.

Cette architecture repose sur **URI**, (**U**niform **R**esource **I**dentifier) qui permet l'attribution d'un identifiant unique à des ressources.

XML (**E**xtensible **M**arkup **L**anguage) est le langage de base qui procure une syntaxe aux documents structurés, mais il ne dispose d'aucune sémantique permettant de décrire la signification de ces documents.

Le langage **RDF** (**R**esource **D**escription **F**ramework) est un langage formel ayant une sémantique simplifiée permettant d'exprimer des relations entre les ressources du Web en utilisant des triplets de la forme sujet-prédicat-objet.

Au RDF s'ajoute **RDFS** (RDFschema) qui permet de déclarer des classes de ressources RDF avec une sémantique pour définir des hiérarchies de généralisation de classes.

RDFS permet aussi de déclarer des propriétés définies comme des relations binaires entre les classes, en précisant leur domaine d'applicabilité et leur domaine de valeurs ainsi que leur niveau de généralisation dans la hiérarchie de propriétés.

Cependant, l'extension à RDFS ne fournit que des mécanismes primitifs pour spécifier les classes et les propriétés et n'intègre pas des capacités de raisonnement. Pour munir les langages du Web d'une sémantique formelle permettant la représentation des ontologies, le RDF et RDFS ont été enrichis par l'apport du langage ontologique **OWL** (**O**ntology **W**eb **L**anguage).

OWL utilise les concepts de classes, de ressources, de littéral et de propriétés des sous-classes, de sous-propriétés, de champs de valeurs et de domaines d'application définis dans RDFS et ajoute les concepts de classes équivalentes, de propriété équivalente, d'égalité de deux ressources, de leurs différences, du contraire, de symétrie et de cardinalité.

5.2 Structure et composants d'une ontologie OWL :

De [31] et [32]

Une ontologie owl est un ensemble d'informations qui inclue des définitions de classes, des propriétés et des instances,

On reconnait une ontologie OWL par les documents ayant "owl" ou "RDF" comme extension. Ce document contient deux sections principales qui sont Les espaces de nommage et leur entête, et les définitions de trois types d'objets définis pour décrire une ontologie: classes, instances et propriétés.

5.2.1 Un espace de nommage (Namespace) :

Pour employer des termes dans une ontologie OWL, il est indispensable d'indiquer la source du vocabulaire de ces termes; L'espace de nommage d'OWL http://www.w3.org/2002/07/owl# est la source intégrale du vocabulaire d'OWL, et c'est pour cette raison que l'ontologie OWL commence par une déclaration d'un namespace contenu dans une balise rdf:RDF tout comme un document XML.

La balise rdf :RDF contient :

- L'espace de nommage propre à l'ontologie.
- Les espaces de nommage des ontologies importées.
- L'espace de nommage du vocabulaire OWL utilisé.
- Les objets définis dans l'espace de nommage de RDF et RDFS.
- Les types de données définis dans l'espace de nommage de XML schéma.

5.2.2 Un entête :

La majorité des documents OWL contiennent un entête placé juste après les balises du namespace, son rôle est de donner une description aux informations sur le contenu de l'ontologie, il est présenté avec la balise **owl :Ontology** .

5.2.3 Classes :

En général, une classe est la définition de plusieurs individus qui partagent certaines propriétés, toutes les classes owl sont des sous classes de la classe **owl :Thing** et au même temps des super classes de la classe **noThing** qui n'a aucune instance.

On peut organiser les classes OWL avec un ordre hiérarchique ; et pour cela on doit utiliser la propriété SubClassOf. Il y a une diversité de façons pour la déclaration d'une classe :

- En nommant la classe.
- En énumérant toutes les instances de la classe.
- Par définition d'une classe anonyme qui se compose de toutes les instances de la classe owl :Thing qui satisfait une ou plusieurs contraintes.
- Par union, intersection ou complémentaire d'autres classes définies.

5.2.4 Propriétés :

Une propriété permet la description des faits ou des relations entre classes. Dans OWL on distingue deux types de propriétés définis par les deux classes **owl :ObjectProperty** et **owl :DataTypeProperty,** ces deux classes sont des sous-classes de la classe **rdf :Property.**

5.2.4.1 Propriétés d'Objets :

La classe owl : objectProperty donne une définition d'une propriété entre deux concepts d'une ou plusieurs classes, autrement dit une relation.

5.2.4.2 Propriété de type de données(DataTypeProperty) :

La classe owl : dataTypeProperty permet de relier une donnée ou une valeur à un individu d'une classe.

5.2.4.3 Image et domaine d'une propriété (Range and Domain) d'une propriété :

Domain et Range signifient respectivement la classe source et la classe cible d'un objectProperty. Le domaine d'un ObjectProperty et d'un DataTypeProperty est une classe OWL et leurs images sont une classe OWL et un type de données OWL.

5.2.5 Instance de classe :

L'instance d'une classe de l'ontologie peut avoir plusieurs noms ce qui peut créer des ambigüités. La solution proposée par OWL est d'utiliser des propriétés telle que **owl : same As, owl :differentForm** et **owl : allDifferent.**

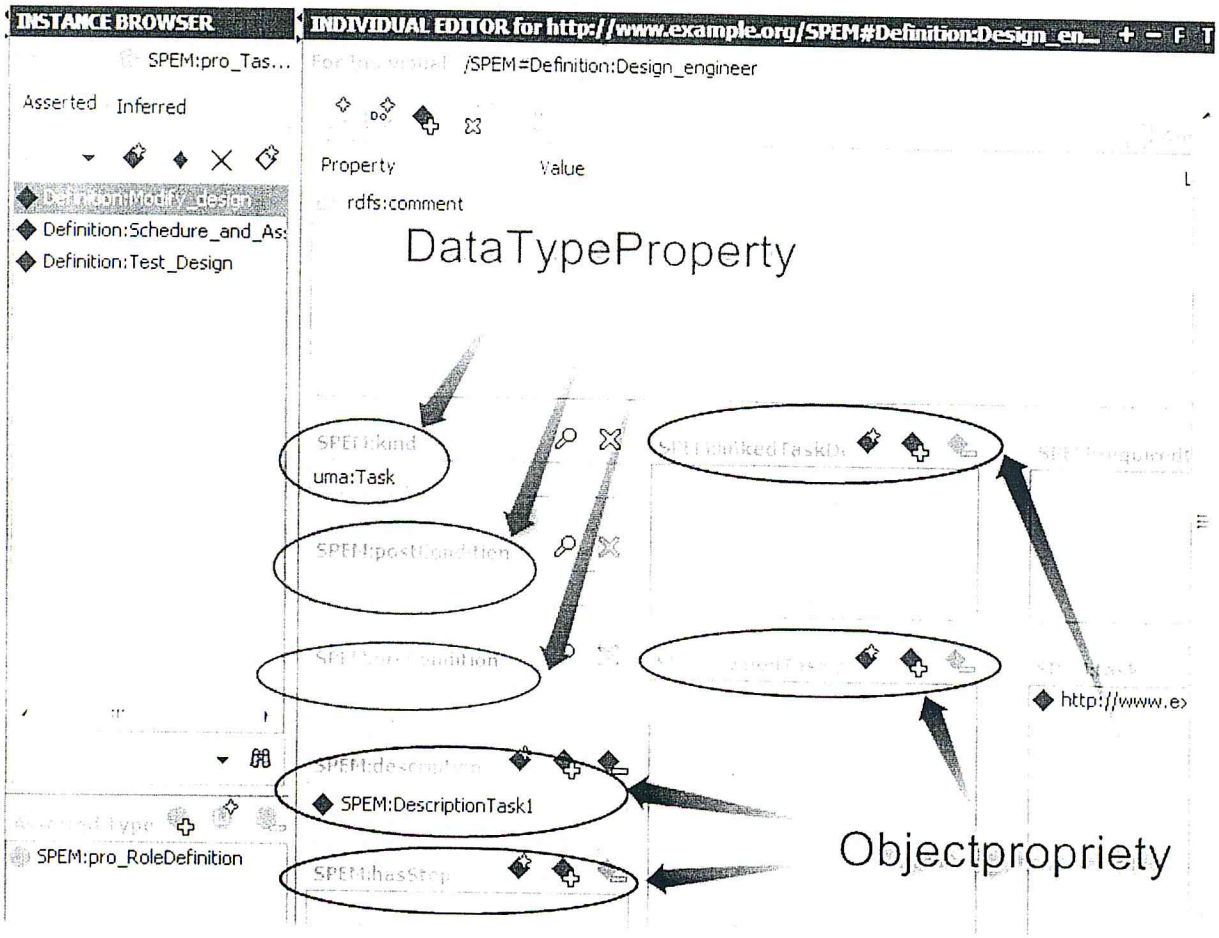


Figure 2.4 : Les propriétés (DataTypePropriety,ObjectPropriety) .

6. Description de l'ontologie SPEMONTOLOGY :

Dans cette partie nous allons présenter l'ontologie SPEMONTOLOGY qui est une ontologie de procédés logiciels plus précisément c'est une ontologie du méta-modèle SPEM.

SPEMOntology est une ontologie de procédés logiciels décrite en langage d'ontologie OWL, elle représente quelques packages du méta-modèle SPEM.

Cette ontologie regroupe tous les concepts manipulés dans le domaine des procédés logiciels indépendamment des techniques et langages de modélisation, elle a été obtenue automatiquement en appliquant des transformations ATL (Atlas Transformation Language) sur le méta-modèle SPEM.

6.1 Les package de SPEMOntology :

SPEMOntology se concentre essentiellement sur le package ProcessWithMethods (Voir chapitre SPEM), elle comporte aussi des classes des packages ProcessStructure, MethodContent, ManagedContent et bien sure des classes du package Core.

Dans SPEMOntology nous retrouvons assez distinctement les éléments du modèle conceptuel de SPEM. Ces éléments sont représentés par des concepts et des relations tels que : le concept TaskUse qui décrit une tâche, le concept RoleUse qui décrit un rôle et le concept WorkProductUse qui décrit un produit.

6.2 Création de SPEMOntology :

Comme dit plus haut, SPEMontology a été créée par le biais de plusieurs transformations ATL(Atlas Transformation Language), ce dernier est un langage de transformation de modèles, il a permis le passage -dans notre cas- du modèle SPEM qui est conforme au méta-modèle SPEM à l'ontologie SPEMontology qui est conforme au méta-modèle OWL.

Finalement, SPEMontology est une ontologie décrite en langage d'ontologie OWL, elle permet la modélisation des procédés logiciels conformes au méta-modèle SPEM. Elle représente les éléments du modèle conceptuel de SPEM, et regroupe tous les concepts utilisés dans le domaine des procédés logiciels.

6.2.1 Structure de SPEMOntologie

L'ontologie SPEMOntologie est structurée en classes selon les packages du standard SPEM 2.0, les deux classes concernées par notre instanciation sont : MethodeContentElement du package MethodeContent et BreakDonwElement du package ProcessStructure.

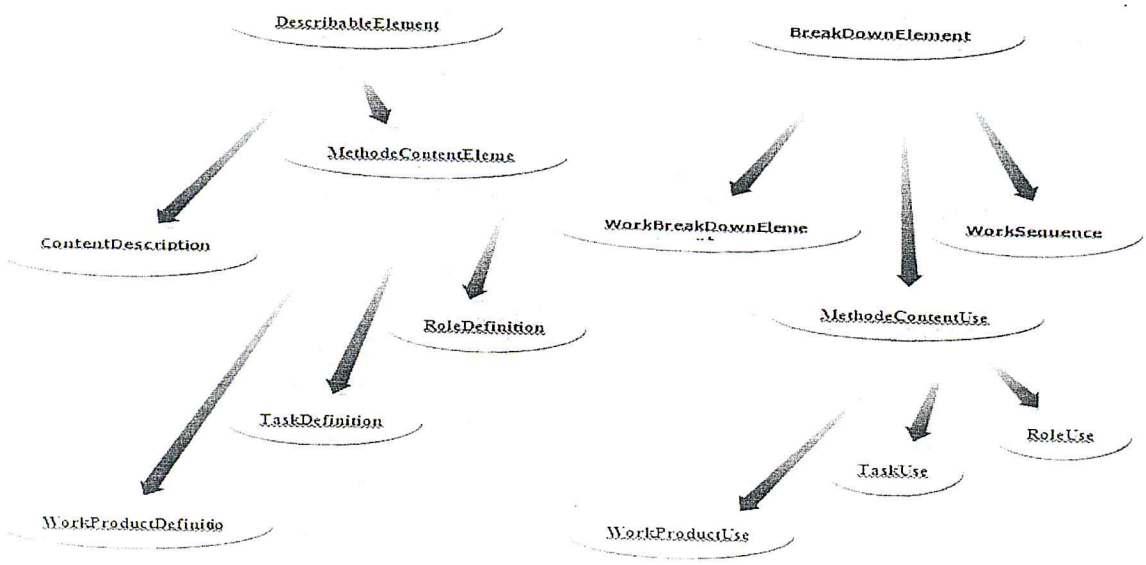


Figure 2.5 : Structure de notre partie de SPEMOntologie.

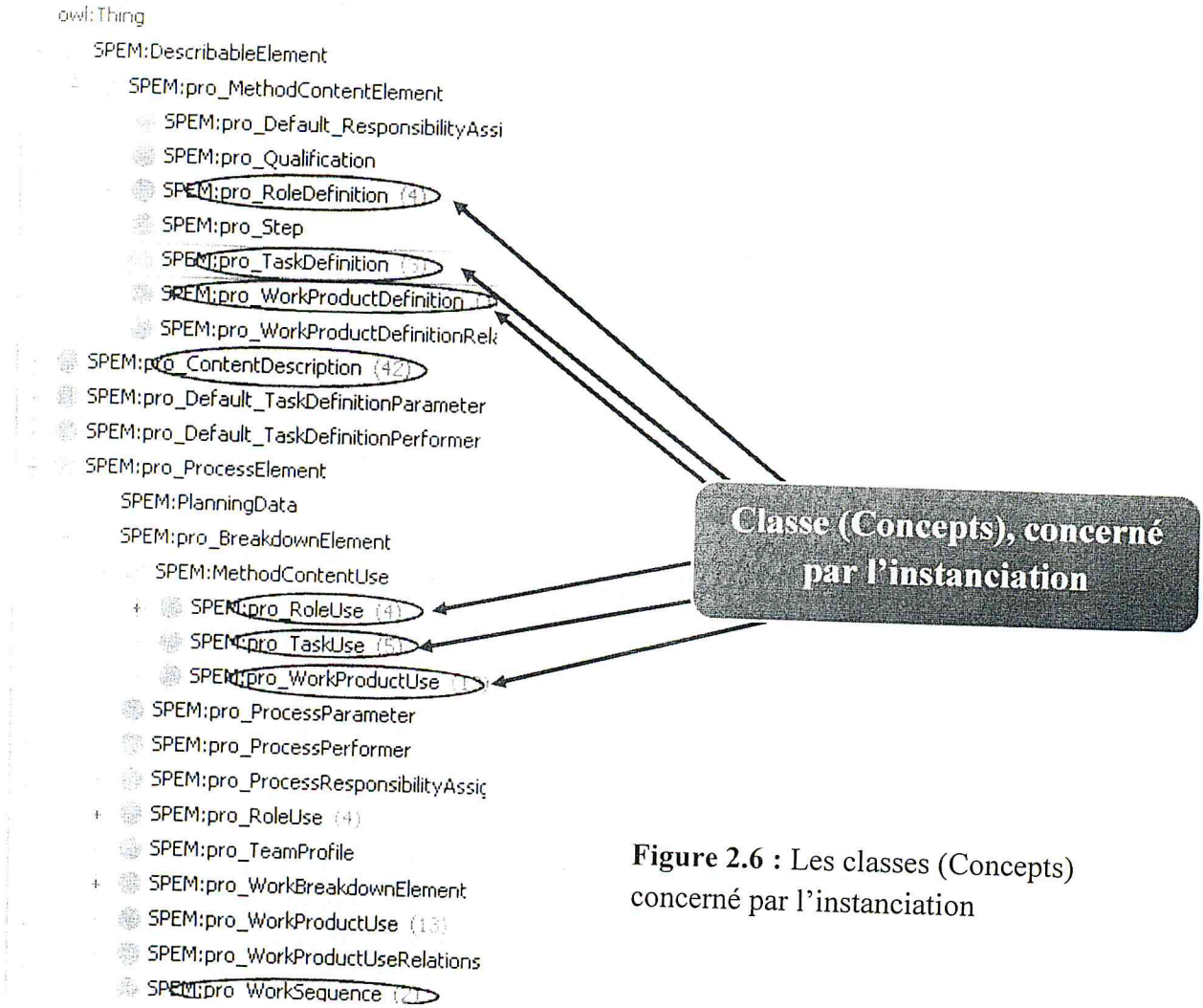
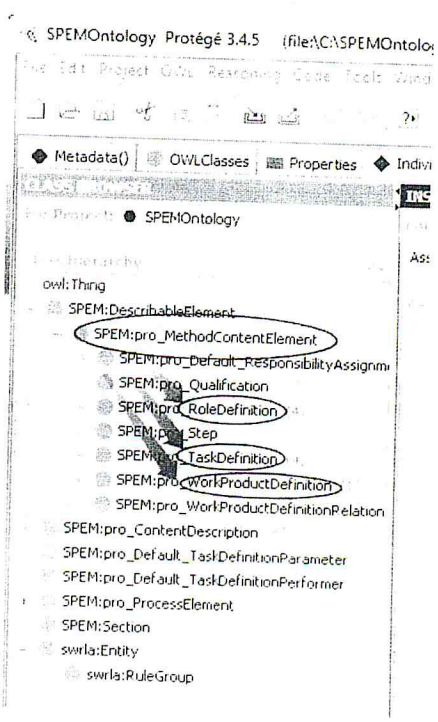


Figure 2.6 : Les classes (Concepts) concerné par l'instanciation

Après avoir vue, la structure de SPEMOntologie, on ouvre cette ontologie avec le logiciel PROTEGE pour la voir plus détaillé, on remarquera l'arborescent des classes de SPEMOntlogie.

La classe MethodeContentElement contient les RoleDefinition, Les TaskDefinition et les WorkProductDefinition, comme sur la figure.



MethodeContentElement est un élément abstrait qui représente une généralisation abstraite pour toutes Méthode

RoleDefinition : c'est l'ensemble de la compétence liée, utilisés par les TaskDefinition pour définir qui les accomplit, et qui est responsable des WorkProductDefinitions.

Figure 2.7 : Les classes (Concepts) MethodeContentElement.

TaskDefinition : est la tache accomplie par les rôles, et chaq'une

d'elle est associé a des produit en entrés et en sorties

WorkProductDefinition : est utilisé, modifié, et produite par des TaskDifinition.

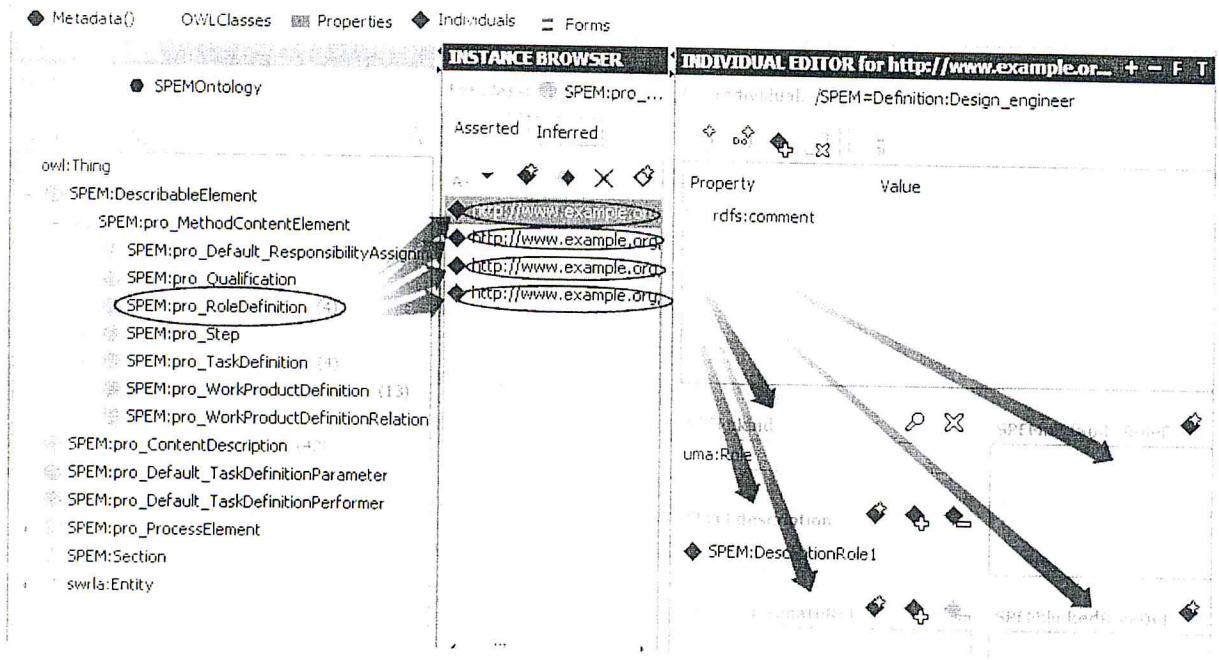


Figure 2.8 : Propriété des instances des classes des définitions

Chaque RoleDefinition, TaskDefinition et WorkProductDefinition, à ses propres propriétés, comme c'est précisé sur la figure.

La classe BreakDownElement contient les RoleUse, Les TaskUse et les WorkProductUse, comme sur la figure.

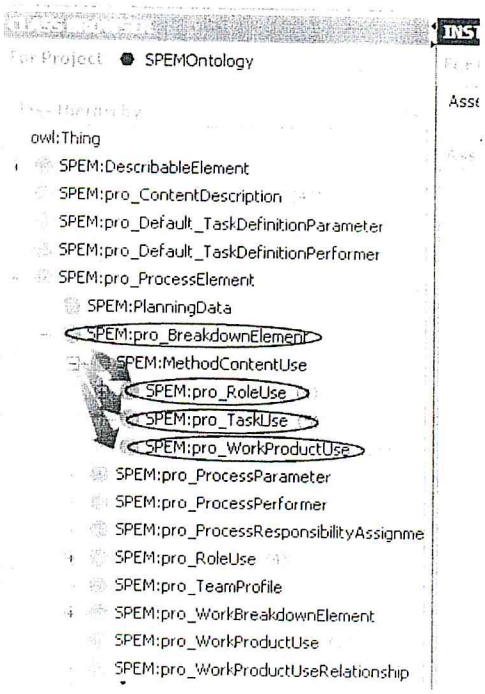


Figure 2.9 : Les classes (Concepts) BreakDownElement

Pour chaque RoleUse, TaskUse et WorkProductUse, à ses propres propriétés, comme c'est précisé sur la figure

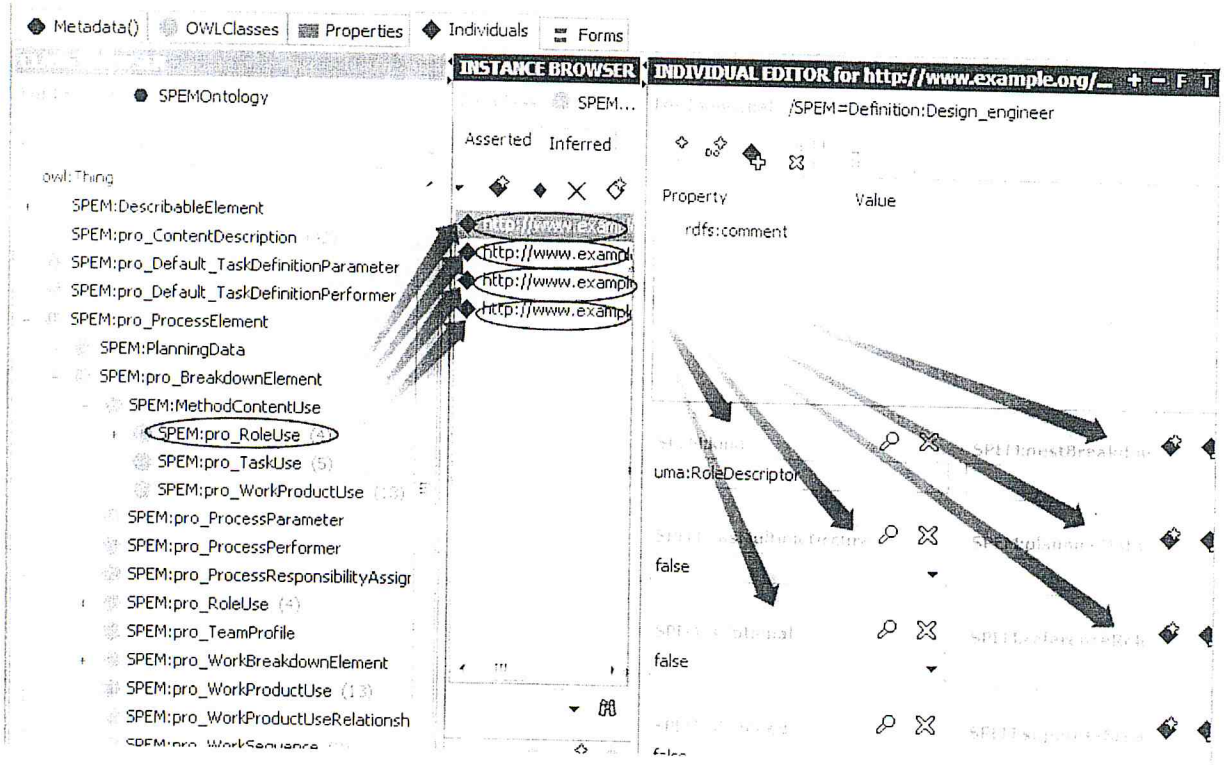


Figure 2.10 : Propriétés des instances des classes (concepts USE).

7. Conclusion :

Nous avons présenté dans ce chapitre le concept d'ontologie, nous expliquons le besoin d'utiliser les ontologies dans le domaine informatique, nous avons par la suite donné plusieurs définitions du terme ontologie, nous avons par la suite comparé le concept d'ontologie et les bases de connaissances.

Puis, nous avons expliqué les ontologies OWL (Ontology Web Language) tout en mettant l'accent sur le langage utilisé (OWL).

Notre objectif est d'instancier automatiquement l'ontologie OWL SPEMontology.

Chapitre 3
Eclipse Process FrameWork

1. Introduction

La maîtrise des processus de développement du logiciel est une tâche complexe qui doit prendre en considération différents produits, utilisateurs, outils et contraintes.

La communauté du Génie Logiciel a fait beaucoup d'efforts pour décrire et formaliser les procédés logiciels, en proposant des techniques similaires à celles qui sont utilisées pour les logiciels. Ces propositions englobent des notations, des Langages de Description de Procédés (LDP) [41], des outils et environnement pour automatiser l'exécution des procédés.

Plusieurs langages et Framework de description de procédés logiciels existent : Appele, PBOOL+ [41] , EPF (Eclipse Process Framework)... Etc .

L'instanciation de notre ontologie se fait par l'extraction des données à partir d'un fichier XML, ce dernier et obtenue à partir d'EPF (Eclispe Process Framework).

Eclipse Process FrameWork est une plateforme d'outils qui permet aux ingénieurs de processus et les gestionnaires à mettre en œuvre, déployer et maintenir les processus des organisations ou des projets individuels.

Ainsi dans ce chapitre, nous présentons les principales caractéristiques du FrameWork EPF qui est un Framework de description de procédés logiciels.

Nous commençons par définir EPF (Eclipse Process Framework) ainsi que les différents modèles qui le composent. Nous prendrons un exemple pour le dérouler avec EPF pour nous permettre de présenter son fonctionnement, car pour instancier SPEMOntlogie il nous faut en premier lieu créer le fichier XML, de plus EPF est complexe et difficile à assimiler.

2. Définition d'Eclipse Process Framework:

De [41] :

L'Eclipse Process Framework (EPF) est un projet open source qui est géré par la Fondation Eclipse. Il se trouve sous le niveau supérieur Eclipse Technology Project. Il a deux objectifs:

- Pour fournir un Framework extensible et des outils exemplaires pour l'ingénierie des processus logiciels
- Pour fournir un contenu processus extensible pour une gamme de développement de logiciels et de soutenir les processus de gestion du développement itératif, progressive, et applicable à un large éventail de plates-formes de développement et les applications.

Eclipse Process Framework (EPF) Composer est une plateforme qui permet aux ingénieurs des procédés et des gestionnaires à mettre en œuvre, déployer et maintenir les processus des organisations ou des projets individuels

EPF Composer peut créer des processus de développement logiciel en le structurant d'une manière spécifique en utilisant un schéma prédéfini, ce schéma est une évolution de SPEM spécification OMG dénommé Unified Method Architecture (UMA). Des parties importantes de l'UMA a dans la révision récente de SPEM.

3. Fonctionnalités d'Eclipse Process Framework

De [41] :

- Amélioration de la réutilisation et les capacités d'extensibilité. Le plug-in mécanismes de versions antérieures ont été étendues pour supporter les extensions des structures de répartition.
- Prise en charge réutilisables Des modèles de processus lié dynamiquement des meilleures pratiques en matière de processus d'assemblage rapide par glisser-déposer.
- Fournit des outils entièrement repensé pour la création, la configuration, de visualisation et des méthodes de publication et les processus.
- Gère le contenu méthode en utilisant de simples interfaces utilisateurs basées sur des formulaires. Par conséquent, les compétences de modélisation UML ne sont plus nécessaires.
- Fournit des éditeurs de texte intuitive riche pour créer des descriptions de contenu illustratif. Editors permettre l'utilisation de styles, images, tableaux, liens hypertextes, et un éditeur HTML directement.

4. Principaux termes et concepts d'Eclipse Process Framework

De [41] :

Pour travailler efficacement avec les EPF Composer, on doit comprendre quelques concepts qui sont utilisés pour organiser le contenu. Les pages de contenu Vue d'ensemble de la méthode et processus de création Aperçu contiennent plus de détails et des exemples concrets de la façon de travailler dans l'outil. Cette page vous fournit un aperçu général de ces concepts.

Le principe le plus fondamental dans EPF Composer est la séparation du contenu de base réutilisables méthode de son application dans les processus. Cela se rapporte directement à ces deux fins de l'EPF Composer décrit dans la première section. Presque tous les concepts EPF Composer's sont classés le long de cette séparation. Méthode contenu décrit ce qui doit être produit, les compétences nécessaires et les explications étape par étape décrit comment les objectifs de développement spécifiques sont atteints. Ces descriptions de contenu méthode sont indépendants d'un cycle de développement. Processus décrire le cycle de développement. Processus prendre les éléments de contenu méthode et de les relier en séquences semi-commandés qui sont adaptées aux types de projets spécifiques.

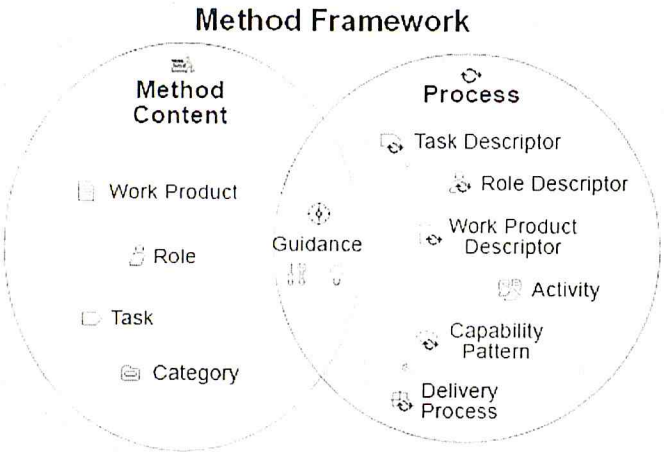


Figure 3.1 : Les éléments utilisés dans Eclipse Process Framework. [41]

De [41] **:**

L'image ci-dessus fournit un résumé des éléments clés utilisés dans EPF Composer et comment ils se rapportent au contenu méthode ou du procédé. Le contenu méthode est principalement exprimé en utilisant les produits de travail (WorkProduct), les rôles (Role), les tâches (Task).

Sur le côté droit du diagramme, il y a les éléments utilisés pour représenter les processus dans l'EPF Composer. L'élément principal est l'activité qui peuvent être imbriqués pour définir les structures de répartition ainsi que liés les uns aux autres pour définir un flux de travail. Activités contiennent également des descripteurs de contenu méthode de référence. Les activités sont utilisés pour définir les processus qui EPF Composer soutenir deux types principaux:

les processus de prestation et les modèles de capacité. Processus de livraison représentent un modèle de processus complète et intégrée pour l'exécution d'un type de projet spécifique. Ils décrivent un cycle de vie du projet de bout en bout et sont utilisés comme référence pour la gestion de projets ayant des caractéristiques similaires. Cela permet la réutilisation optimale et l'application de leurs meilleures pratiques clés dans le processus de création d'activités dans EPF Composer.

5. Méthode de création d'un Processus avec EPF

De [42] **:**

Pour comprendre la création d'un processus avec EPF, on prendra ISPW-6 comme exemple, L'exemple processus ISPW-6 a été publié dans le 6e Atelier International Software Process, Le but de cet exemple est de faciliter la compréhension, la comparaison et l'évaluation des différentes approches qui sont poursuivis pour la modélisation des processus logiciels.

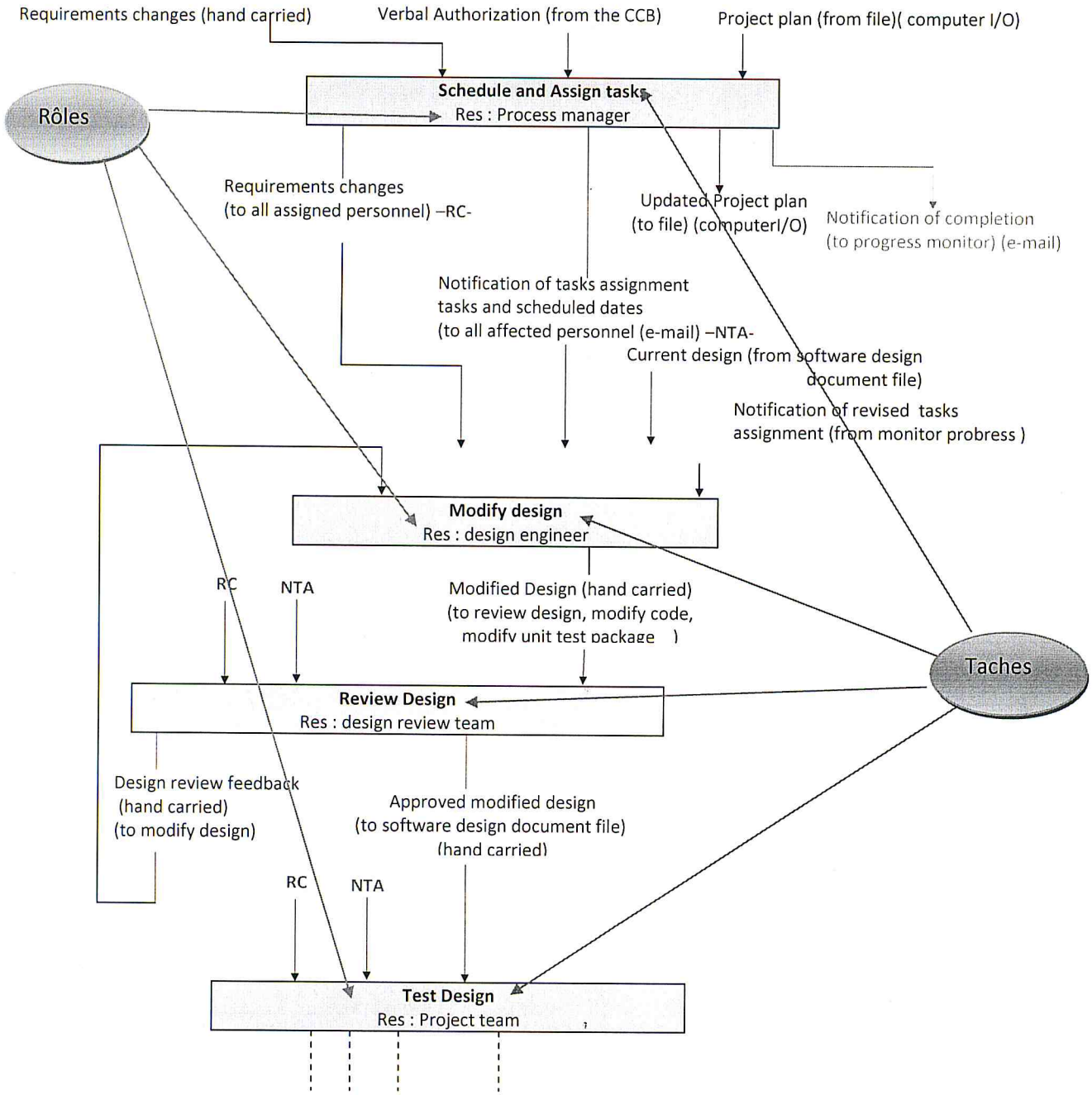


Figure 3.2: Processus (Proceedings of the 6th International Software Process Workshop), **ISPW-6**

Dans l'exemple, les rectangles représente les taches avec le rôle de chaque tache, et tous les entrés et sortie, se sont des produits, donc des produits en entré et en sortie, par exemple :

La tache **Schedule and Assign tasks** est réalisé par le role **Process manager** et les produits : **Requirementschanges (hand carried)** , **Verbal Authorization (from the CCB)** et **Project plan (from file)(computer I/O)** sont des produits en entrer pour cette taches et les produits : **Requirements changes (to all assigned personnel), Notification of tasks assignment tasks and scheduled dates** et **Updated Project plan** sont des produit en sortie ,et ces derniers sont des produits en entrer pour la tache **Modifiy design,** etc.

On va introduire cet exemple dans EPF, pour obtenir à la fin un fichier XML, qui contiendra tous ces informations.

- ✓ On crée une nouvelle bibliothèque méthode appelée MyLibrary, qui contiendra la méthode plugin, configuration de la méthode, c'est la ou tous sera stocké.

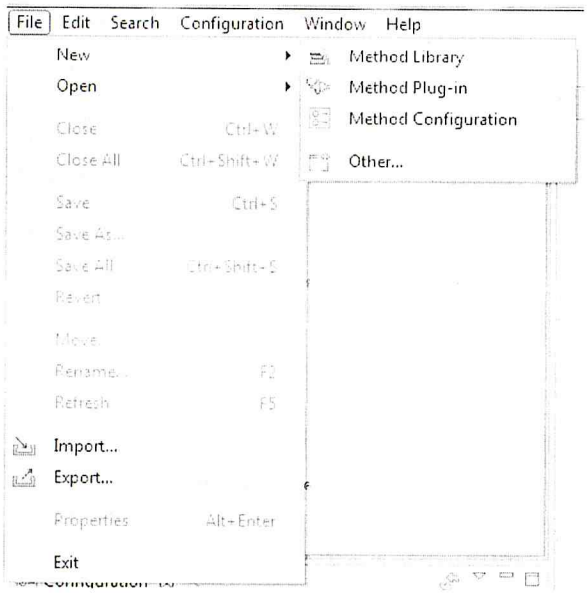


Figure 3.3 : Creation de Methode Library

On crée une nouvelle configuration appelée MyConfiguration, elle est utilisé pour créer des processus et a la publication en définissant les éléments qui seront publié.

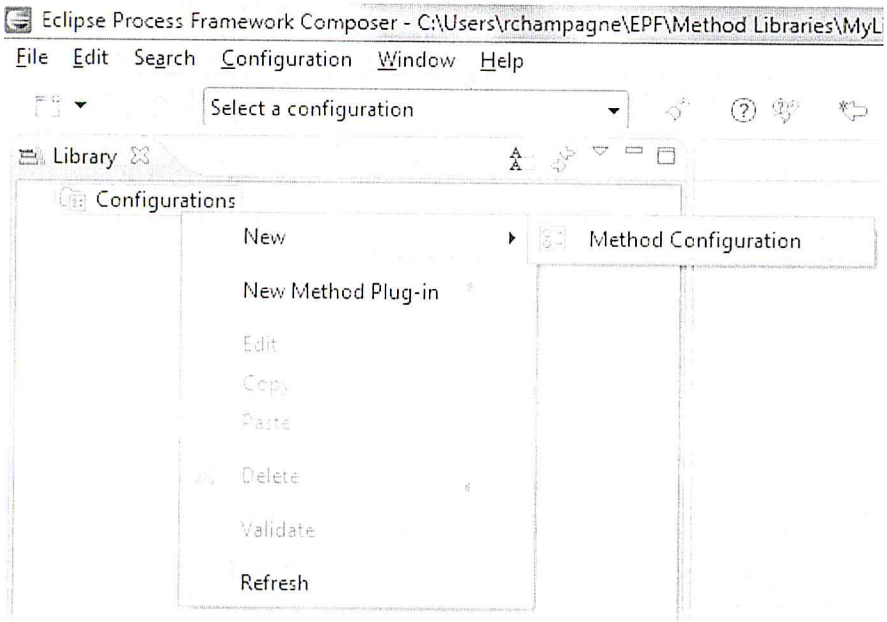


Figure 3.4 : Creation de Methode Configuration

✓ Près avoir créé la bibliothèque et aussi la configuration, on crée une nouvelle méthode plugin appelé MyPlugin, dans ce dernier on trouvera Methodecontent, et aussi Processes,

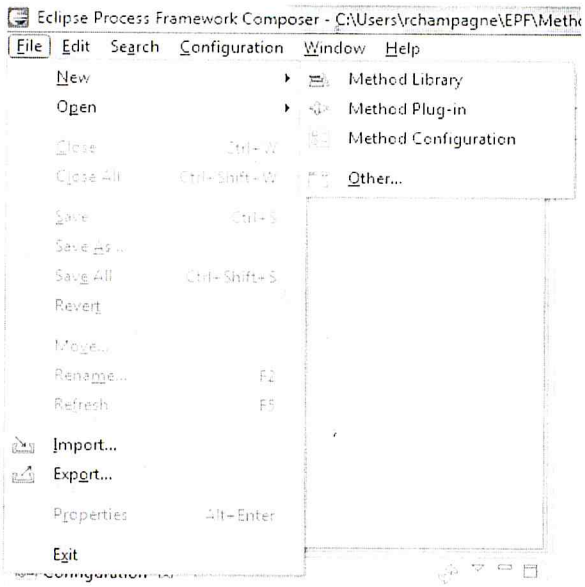


Figure 3.5 : Creation de Methode Plug-in

✓ On crée un package appelé MyContentPackage dans MyPlugin, qui contiendra les rôles, taches et les produits.

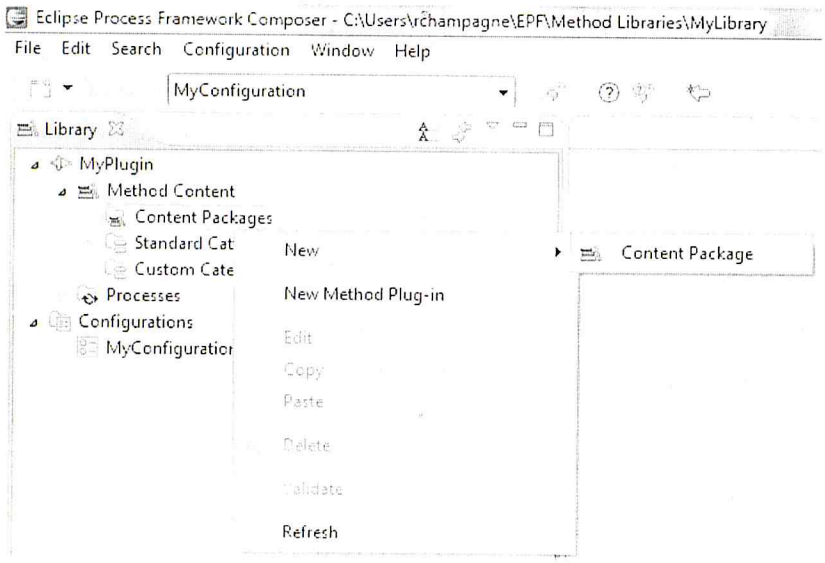


Figure 3.6 : Creation de Content Package

✓ On ajoute tous les rôles du Processus **ISPW-6** dans MyContentPackage, on obtiendra la figure suivant :

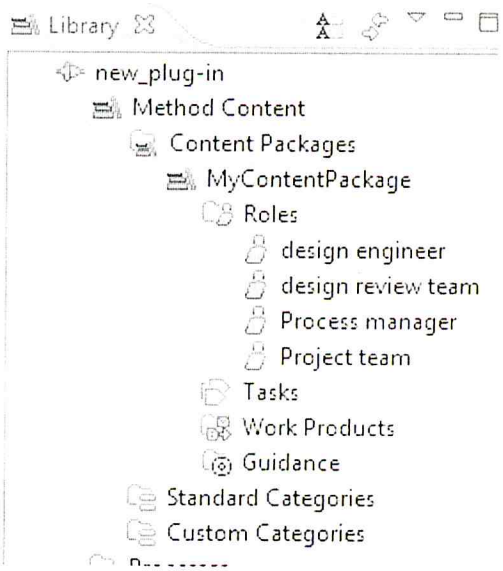


Figure 3.7 : Création des Rôles pour le processus **ISPW-6**

- ✓ Après avoir ajouté les rôles, on ajoute maintenant les taches et les produits

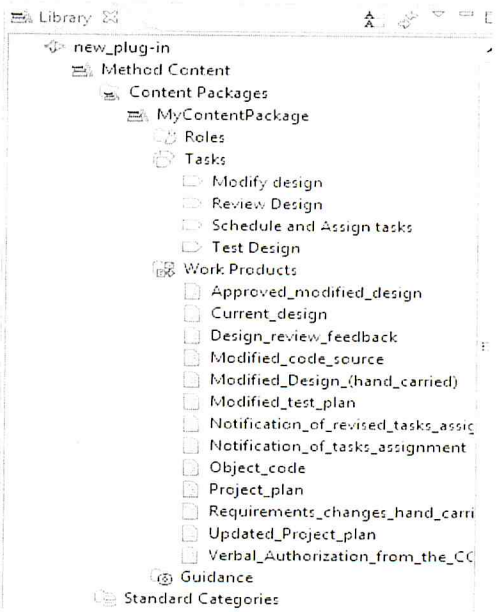


Figure 3.8 : Création des produits et des Taches pour le processus **ISPW-6**

- ✓ on attribut pour chaque taches sont rôle approprier

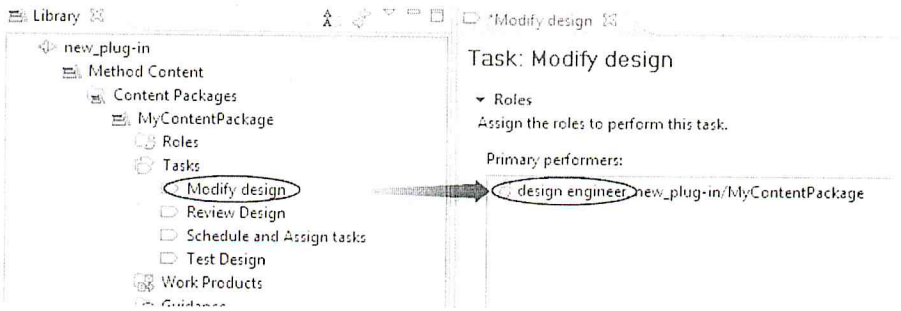


Figure 3.9 : Attribution des rôles a des taches

✓ Après avoir ajouté tous les rôles, les taches et les produits, en affecte chaque rôle a sa tache, et les produits en entrer et en sortie pour chaque tache, selon l'exemple.

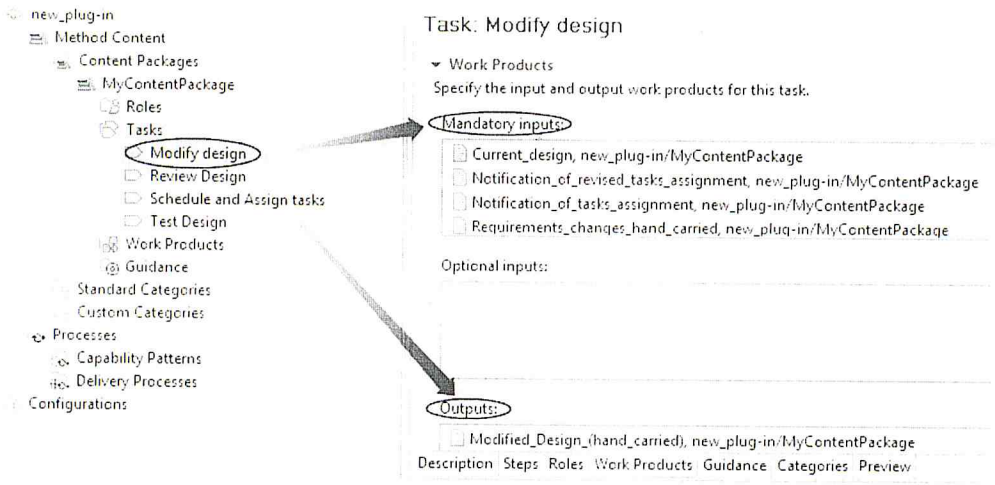


Figure 3.10 : Attribution des produits en entrer et en sortie a des taches

✓ On défini les propriétés des RoleUse, les TaskUse et les WorkProductUse

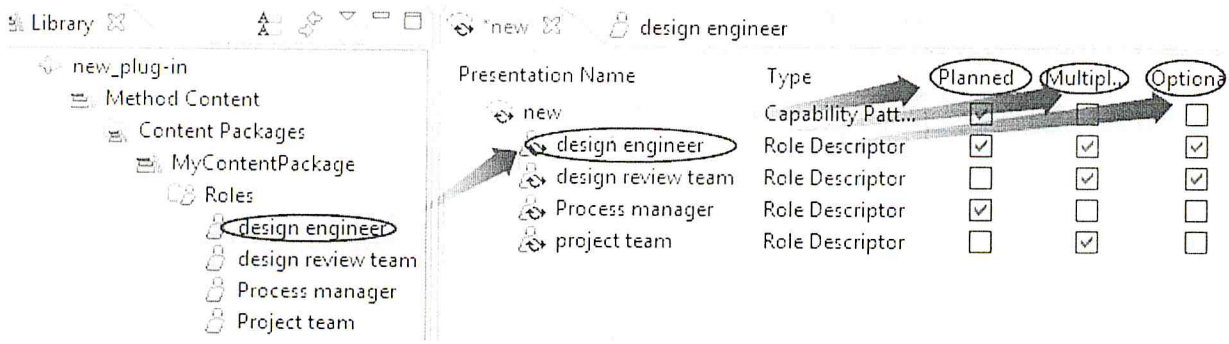


Figure 3.11 : Définir les propriétés de chaque RoleUse, TaskUse et WorkProductUse .[]

✓ **Exporter en XML**

- après avoir ajouté les rôles les tâches et les produits et les relation entre eux on exporte le tous en XML car notre XML consiste a l'extraction de donnée de xml pour l'instanciation de SPEMOntologie

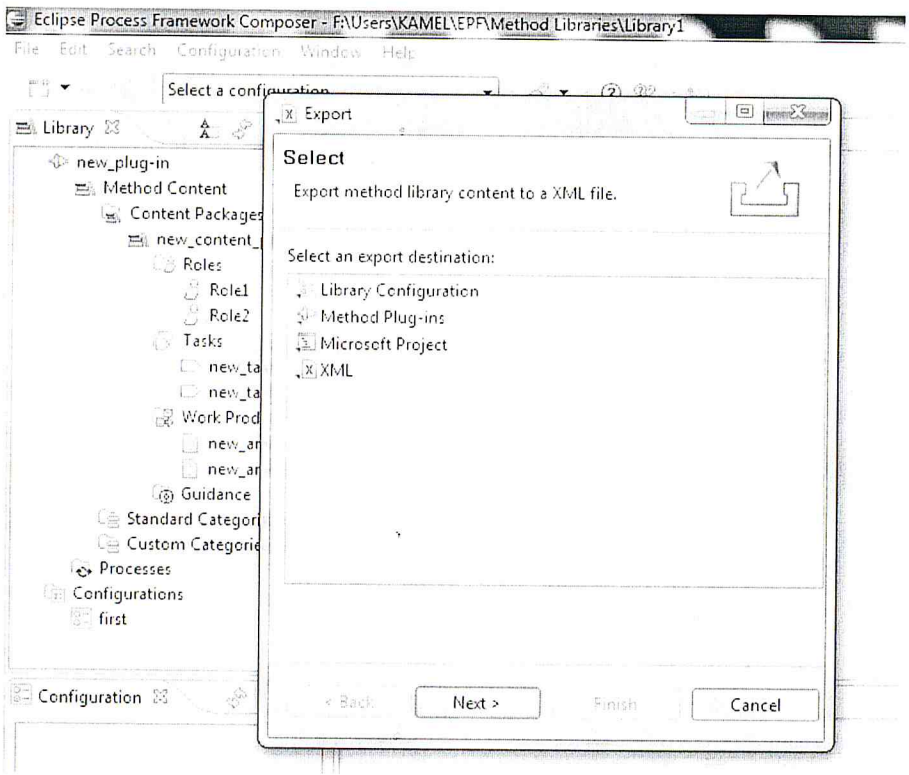


Figure 3.12 : Exporter en fichier XML.[]

➢ **Le fichier XML obtenue**

```
<?xml version="1.0" encoding="UTF-8" ?>
<uma:MethodLibrary xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns
  id="_dK-9sGEqEeCKPJCz43JaLQ" orderingGuide="" presentationName="" suppressed="fal
  <MethodElementProperty name="library_synFree" value="true" />
- <MethodPlugin name="Deuxième" briefDescription="" id="_cprPUG1-EeCFQpmePXtGgw'
    changeDescription="" version="" supporting="false" userChangeable="true">
    <MethodElementProperty name="plugin_synFree" value="true" />
  + <MethodPackage xsi:type="uma:ContentCategoryPackage" name="ContentCategori
  + <MethodPackage xsi:type="uma:ContentPackage" name="Deuxième_package" brief
      presentationName="" suppressed="false" global="false">
  - <MethodPackage xsi:type="uma:ProcessPackage" name="deuxième" briefDescription=
      suppressed="false" global="false">
    - <MethodPackage xsi:type="uma:ProcessComponent" name="hh" briefDescription="" id
      suppressed="false" global="false" authors="" changeDescription="" version="">
      - <Process xsi:type="uma:DeliveryProcess" name="hh" briefDescription="" id="_nW8p
          suppressed="false" isAbstract="false" hasMultipleOccurrences="false" isOptional="f
          isRepeatable="false" variabilityType="na">
        - <Presentation xsi:type="uma:DeliveryProcessDescription" name="hh,_nW8pAX2
            orderingGuide="" presentationName="" suppressed="false" authors="" changeDesc
            <MainDescription />
            <KeyConsiderations />
            <Alternatives />
            <HowToStaff />
            <Purpose />
            <Scope />
            <UsageNotes />
            <Scale />
            <ProjectCharacteristics />
            <RiskLevel />
            <EstimatingTechnique />
            <ProjectMemberExpertise />
            <TypeOfContract />
          </Presentation>
```

Figure 3.13 : Notre fichier XML

6. XML

6.1 Introduction

De [43]

Dan Connolly ajoute le Standard Generalized Markup Language à la liste des activités du World Wide Web Consortium lorsqu'il s'y joint en 1995. Les travaux débutent à la mi-1996 lorsque l'ingénieur Jon Bosak **(en)** de Sun Microsystems élabore une charte et recrute des collaborateurs. Bosak se fait connaître dans la petite communauté de personnes qui avaient de l'expérience à la fois dans le SGML et dans le Web.

XML est compilé par un groupe de travail de onze membres, soutenu par environ 150 membres de divers groupes d'intérêt.

XML (*Extensible Markup Language*, « langage de balisage extensible») est un langage informatique de balisage *générique* qui dérive du SGML. Cette syntaxe est dite *extensible* car elle permet de définir différents espaces de noms, c'est à dire des langages avec chacun leur vocabulaire et leur grammaire, comme XHTML, XSLT, RSS… Cette syntaxe est reconnaissable par son usage des *chevrons* (< >) encadrant les *balises*. L'objectif initial est de faciliter l'échange automatisé de contenus complexes (arbres, texte riche…) entre systèmes d'informations hétérogènes (interopérabilité). Avec ses outils et langages associés une *application* XML respecte généralement certains principes :

- la structure d'un document XML est définie par un schéma,
- un document XML est entièrement transformable dans un autre document XML.

Un Document bien formé répond aux règles de base de XML

- Balises (éléments) ouvrantes /fermantes avec texte au milieu
- Balises « vides »
- Pas d'imbrication (un élément ne peut pas contenir un élément du même nom)

Avantage de XML **De** [43]

- La grande limite d'HTML est qu'il est quasiment impossible de réutiliser l'information.
- SGML était considéré comme une norme trop lourde et inadaptée au traitement des documents pour le web.
- XML essaie de combiner la flexibilité de SGML avec la simplicité de HTML.

6.2 Caractéristique de XML De[43]

- Markup, Balisage: basé sur des balises (ou éléments) ouvrantes et fermantes
- Langage, Comporte des règles (de grammaire) strictes.
- Extensible, On peut inventer ses propres balises

6.3 les différences entre XML/HTML De[43]

HTML	XML
□Langage figé: défini par le W3C □Exprime d'avantage la forme que le contenu □Est interprété par un navigateur □Laxisme	□Langage extensible □Exprime uniquement le contenu □Doit subir un retraitement pour être visible à travers un navigateur □rigueur

6.4 Que fait-on avec XML? De[43]

- Des documents structurés : Livre, article, etc.... Les projets « revues.org», « euclid», « cyberthèses», «sparte»... sont basés sur des documents en XML. (ou convertis en XML)
- Des métadonnées : De très nombreux projets et applications échangent des métadonnées en XML. L'exemple le plus « universel » est OAI.
- *Des sites web :* On peut imaginer un site dont tout le contenu est en XML, et la conversion en HTML se fait « à la volée ».
- XML passe pour être un «format pérenne».

6.5 XML et les métadonnées De[43]

- XML sert souvent de format d'échange de métadonnées.
- Contrairement à des données en HTML, nous pouvons extraire les données d'un fichier XML, et les réutiliser (alimenter une Onrtologie par exemple).
  - Il existe des « parseurs » (analyseurs syntaxiques) pour chaque langage de programmation, permettant de décortiquer le XML, et le réutiliser aisément, comme Dom, Jdom , Sax

6.6 Composition de XML

```
<?xmlversion="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheettype="text/css" href="ecolthem.css"?>
<rencontre type="ecole">
<titre>Documentation en mathématiques</titre>
  <organisateurs>
    <organisateur>Réseau National des Bibliothèques de Mathématiques</organisateur>
    <organisateur>Cellule MathDoc</organisateur>
  </organisateurs>
  <dates>
    <debut>2004-10-11</debut>
    <fin>2004-10-15</fin>
  </dates>
  <responsables>
    <resp_scientifique>
    <nom>Bost, Jean-Benoît</nom>
    <email>bost@math.u-psud.fr</email>
  </resp_scientifique>
  <resp_admin>
    <nom>Marchand, Monique</nom>
    <email>mmarchan@ujf-grenoble.fr</email>
    </resp_admin>
  </responsables><
</rencontre>
```

entête

« type» est un attribut

« organisateur» est un élément

Ainsi que :
« rencontre »,
« responsables»,
etc…

Figure 3.14 : Fichier XML.

✓ **Composition de notre fichier XML**

Notre fichier XML est composé de deux package :

- **MethodeConfiguration**
- **MethodePlugin**.

MethodeConfiguration contient les **MethodePluginSelection** et **MethodePackageSelection,** ces derniers contiennent les plugins et les package sélectionné dans la configuration.

MethodePlugin contient quatre package, le **premier package** regroupe les rôles, les taches et les produits dans des catégories appelé **ContentCategory**, pour qu'il soit bien organisé, le **deuxième package** contient des **ContentElement,** qui contient des description détaillé des rôles, taches et les produit avec leur propriétés, et pour les taches il contient une description détaillés sur les produit en entré et en sortie pour chaque tache.

Le **troisième package** contient des **BreakDownElement**, qui contient les **RoleUse,** les **TaskUse** et les **WorkProductUse** avec leur propriétés.

Comme c'est présumé sur la figure suivante.

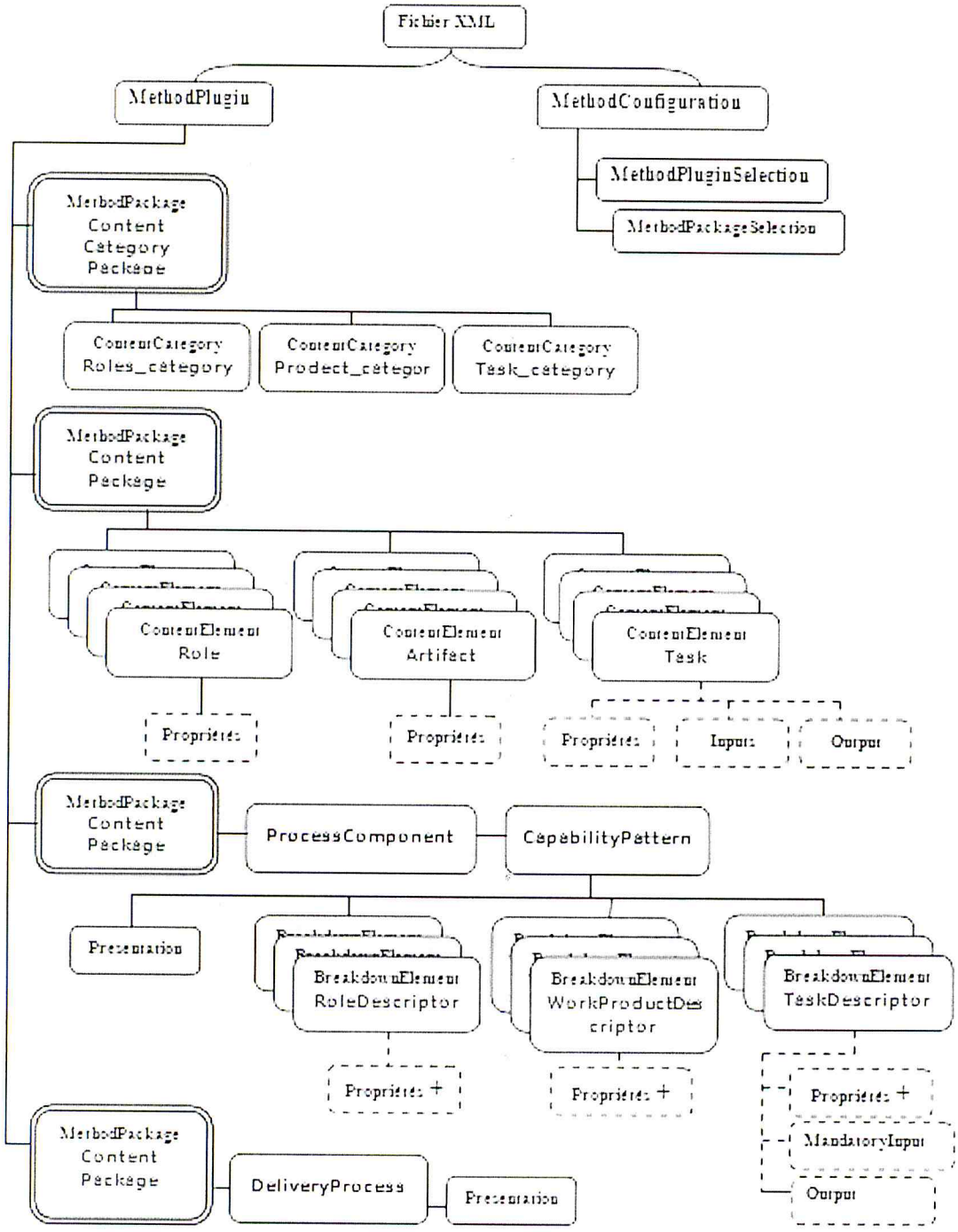


Figure 3.15 : Composition du fichier XML

7. Conclusion

Nous avons présenté dans ce chapitre EPF (Eclipse Process Framework) ainsi que les différents modèles qui le composent, nous avons pris un exemple pour le dérouler avec EPF pour nous permettre de bien présenter son fonctionnement.

Puis, nous avons présenté le fichier XML, obtenue d'Eclipse Process FrameWork, qui était l'objectif à attendre en utilisant EPF, car notre projet consiste a extraire des données d'un fichier XML et les instanciés dans notre ontologie SPEMOntologie, dans ce qui suit nous allons présenter notre conception.

Chapitre 4
Conception

1. Introduction

La réutilisation des procédés logiciels existants peut se faire grâce à une ontologie qui regroupe les concepts de base de procédés logiciels. SPEMontology c'est une ontologie généré par transformation modèle.

Le méta-modèle SPEM est un standard de l'OMG représentant la plupart des modèles de procédés logiciels existants.l

L'instanciation de SPEMontology permettra de capitaliser les connaissances des PLs.

Le but de notre travail est d'instancier cette ontologie automatiquement à partir des modèles de procédés logiciels existants. Dans notre cas est décrit en XML, obtenue a partie de EPF (Eclispe Process Framework).

Nous présentons dans ce chapitre la conception de notre application. La conception est faite suivant la démarche en cascade en adoptant le langage UML.

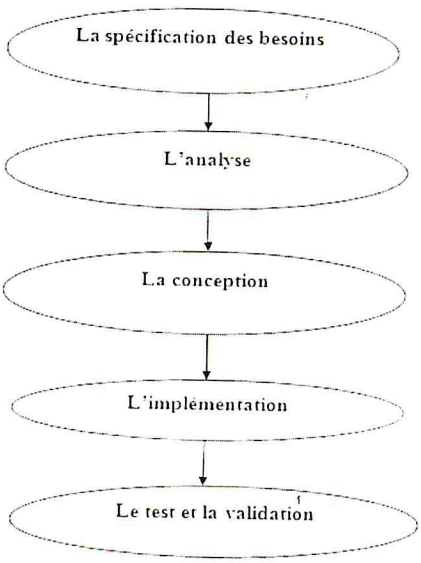
2. Méthode de conception

Notre système à développer a un cycle de vie en cascade, on utilise cette méthode parce qu'elle est :

- Simple
- Agile pour les petits systèmes
- Pas d'itération

3. Présentation de cycle de vie en cascade :

Décrit par Royce en1970, il a été largement employé depuis pour la description générale des activités liées aux logiciels.il décrit le cycle de vie d'un logiciel par une suite de phases qui s'enchainent dans un déroulement linéaire, depuis l'analyse des besoins jusqu'à la maintenance.



Cycle de vie d'un logiciel (modèle en cascade)

4. Quelques notions sur UML:

4.1 Définition de l'UML : De [36]

U.M.L est un langage unifié de la modélisation objet. Il est le résultat d'un long processus initialisé par trois grands méthodistes Grady Booch, Ivar Jacobson et Jim Rumbaugh. Sa notation est un formalisme issu de la fusion de la notation de Booch, d'OMT (*Object Modeling Technique*) et OOSE (*Object Oriented Software Enginneering).* Les concepteurs de l'UML l'ont conçu pour être lisible sur des supports très variés et simples.

UML s'est très rapidement imposé à la fois auprès des utilisateurs et sur le terrain de normalisation.

Le langage UML se concentre sur la description du développement de logiciel plutôt que sur la formation du processus de développement lui-même. Il peut ainsi être utilisé pour décrire les éléments du logiciel obtenu par l'application de différents processus de développement. UML n'est pas une notation fermée, il est générique, extensible et configurable par l'utilisateur. UML ne recherche pas la spécification à outrance; il n'y a pas une représentation graphique pour tous les concepts imaginables ; en cas de besoin particuliers des précisions peuvent être apportés au moyen de mécanismes d'extension et de commentaires textuels. Une grande liberté est donnée aux outils pour le filtrage et la visualisation d'information. L'usage de couleurs, de dessins et d'attributs graphiques particuliers sont laissés à la discrétion de l'utilisateur.

4.2 Les diagrammes d'UML :

Un diagramme donne à l'utilisateur un moyen de visualiser et de manipuler des éléments de modélisation. UML définit neuf sortes de diagrammes pour représenter les points de vue de modélisation. L'ordre de présentation de ces différents diagrammes ne reflète pas un ordre de mise en œuvre dans un projet mais simplement une démarche pédagogique.

Les diagrammes peuvent montrer tout ou une partie des caractéristiques des éléments de modélisation, selon le niveau de détails utiles dans le contexte d'un diagramme donné.

Voici les différents diagrammes d'UML :

1. **Les diagrammes d'activités** qui représentent le comportement d'une opération en termes d'actions.
2. **Les diagrammes de cas d'utilisation** qui représentent les fonctions du système du point de vue de l'utilisateur.
3. **Les diagrammes de classes** qui représentent la structure statique en termes de classes et de relations.
4. **Les diagrammes de collaboration** qui sont une représentation spatiale des objets, des liens et des interactions.
5. **Les diagrammes de composants** qui représentent les composants physiques d'une application.
6. **Les diagrammes de déploiement** qui représentent le déploiement des composants sur les dispositifs matériels.

7. **Les diagrammes d'états transitions** qui représentent le comportement d'une classe en termes d'états.
8. **Les diagrammes d'objets** qui représentent les objets et leurs relations et qui correspondent à des diagrammes de collaboration simplifiés, sans représentation des envois de messages.
9. **Les diagrammes de séquence** qui sont une représentation temporelle des objets et de leurs interactions.

Dans notre conception, nous utilisons le diagramme de cas d'utilisation, le diagramme de séquence et le diagramme de composants. Ces quatre diagrammes sont suffisants pour établir une description et une conception détaillée de notre système.

5. Les concepts concernées par l'instanciation

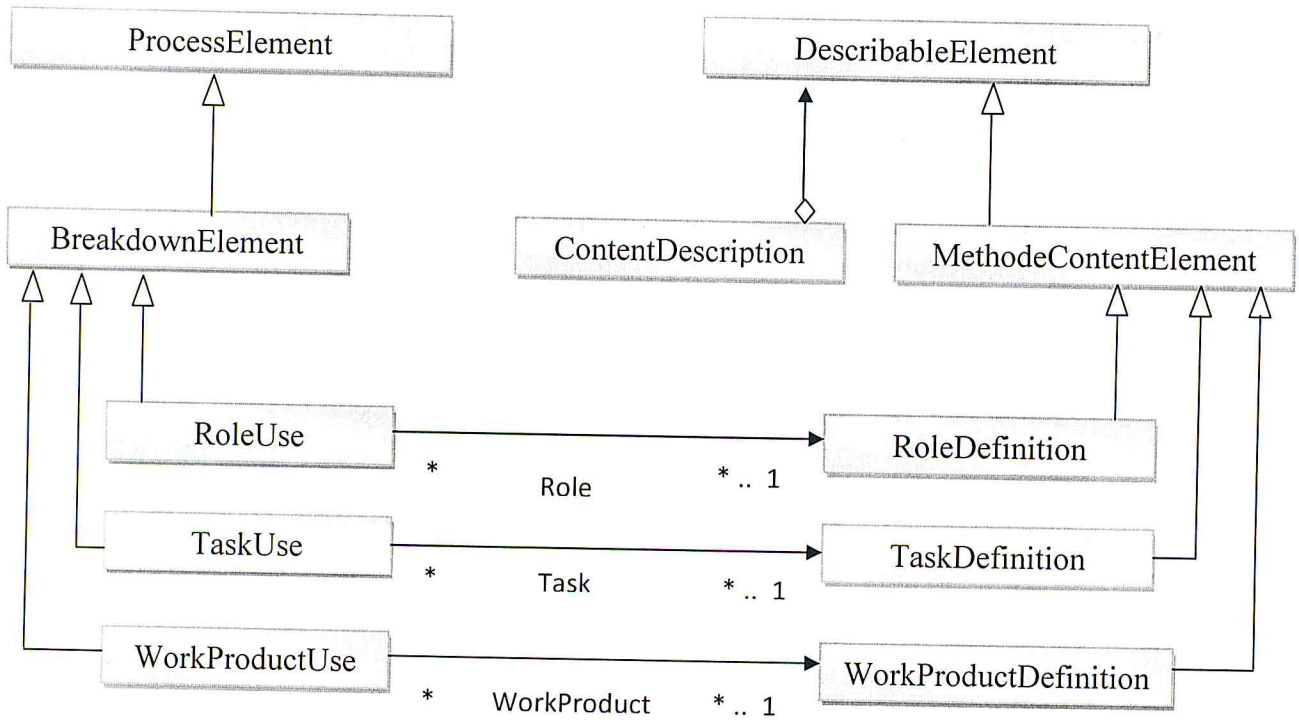


Figure 4.1 Schéma des concepts concernés par l'instanciation

Dans SPEMontology :

- Un **RoleUse** est défini par la classe **RoleDefinition** qui lui est reliée par l'ObjectProperty **Role.**
- Un **workProductUse** est défini par la classe **workProductDefinition** qui lui est reliée par l'ObjectProperty **workProduct.**
- Une **TaskUse** est définie par la classe **TaskDefinition** qui lui est reliée par l'objectProperty **Task.**

Remarque :

TaskDefinition contient la description de la classe **TaskUse, TaskUse** est un **BreakDownElement** , ce dernier est relié à la classe **ContentDescription** via l'ObjectProperty **description.**

Toutes ces définitions servent de référence à **SPEMontology**, Dans notre travail l'instanciation de ces instances se fera **manuellement** après **l'instanciation automatique** de **SPEMontology** à partir du modèle **XML de EPF.**

6. Correspondance de XML avec l'ontologie SPEMOntology

Le code **XML** généré a partir de EPF représente des tâches avec leurs rôles leurs produits en entrée et en sortie et la succession de ces tâches. Ces notions sont identifies dans **SPEMontology,** car l'ontologie respect le méta modèle SPEM.

Par conséquent,

- Dans la classe **ContentElement :**

La définition d'une Tache dans le code **XML** sera représentée par une instance de la classe **Pro_TaskDefinition** dans **SPEMontology**.

La définition d'un Produit dans le code **XML** sera représenté par une instance de la classe **Pro_workProductDefinition** dans **SPEMontology**.

La définition d'un Rôle dans le code **XML** sera représenté par une instance de la classe **Pro_RoleDefinition** dans **SPEMontology**.

- Dans la classe **BreakDownElement :**

Une Tache dans le code **XML** sera représentée par une instance de la classe **Pro_Task** dans **SPEMontology**.

Un Produit dans le code **XML** sera représenté par une instance de la classe **Pro_workProductUse** dans **SPEMontology**.

Un Rôle dans le code **XML** sera représenté par une instance de la classe **Pro_RoleUse** dans **SPEMontology**.

Pour affecter une description a un rôle, tache et produit , nous devons instancier la classe **ContentDescription** qui relie une **description** à un rôle, tache et produit grâce aux ObjectProperty **description.**

Pour définir la succession et l'ordre de déroulement des Taches dans un modèle de procédés logiciel, nous devons instancier la classe **WorkSequence** qui est une sous-classe de la classe **workBreakDownElement** (relation de subsomption) et un de ces deux objectProperty :

I. **linkKind** :type lien(FinishToStart, StartToFinish,StartToStar,FinishToFinish).
II. **Predecessor** : qui relie une tache à la tache qui la précède.

Les tableaux suivant illustre les concepts concernés par l'ontologie, qui on été identifié en comparant les concepts du fichier XML avec ceux de l'ontologie :

XML	ONTOLOGIE
Contentelement • Role **type** **Name** **briefDescription** **presentationName** • Artifact **type** **briefDescription** **presentationName** • Task **type** **briefDescription** **presentationName**	MethodeContentelement • Roledefinition **Kind** **Description** **Roles** • WorkProductDefinition **Kind** **Description** • Taskdefinition **Kind** **Description** **task**

Tableau 4.1 Comparaison XML / Ontologie (1)

XML	ONTOLOGIE
BreakdownElement • RoleDescriptor Type hasMultipleOccurrences isOptional isPlanned • WorkProductDescriptor Type hasMultipleOccurrences isOptional isPlanned briefDescription presentationName • TaskDescriptor type hasMultipleOccurrences isEventDriven isOngoing isOptional isPlanned isRepeatable presentationName briefDescription	**BreakdownElement** • RoleUse Kind hasMultipleOccurrences isOptional isPlanned • WorkProdectUse Kind hasMultipleOccurrences isOptional isPlanned description • TaskUse Kind hasMultipleOccurrences isEventDriven ongoing isOptinal isPlanned isRepeatable description linktoToPrdécesor linkToSuccesor

Tableau 4.1 Comparaison XML / Ontologie (2)

7. Le cas d'utilisation:

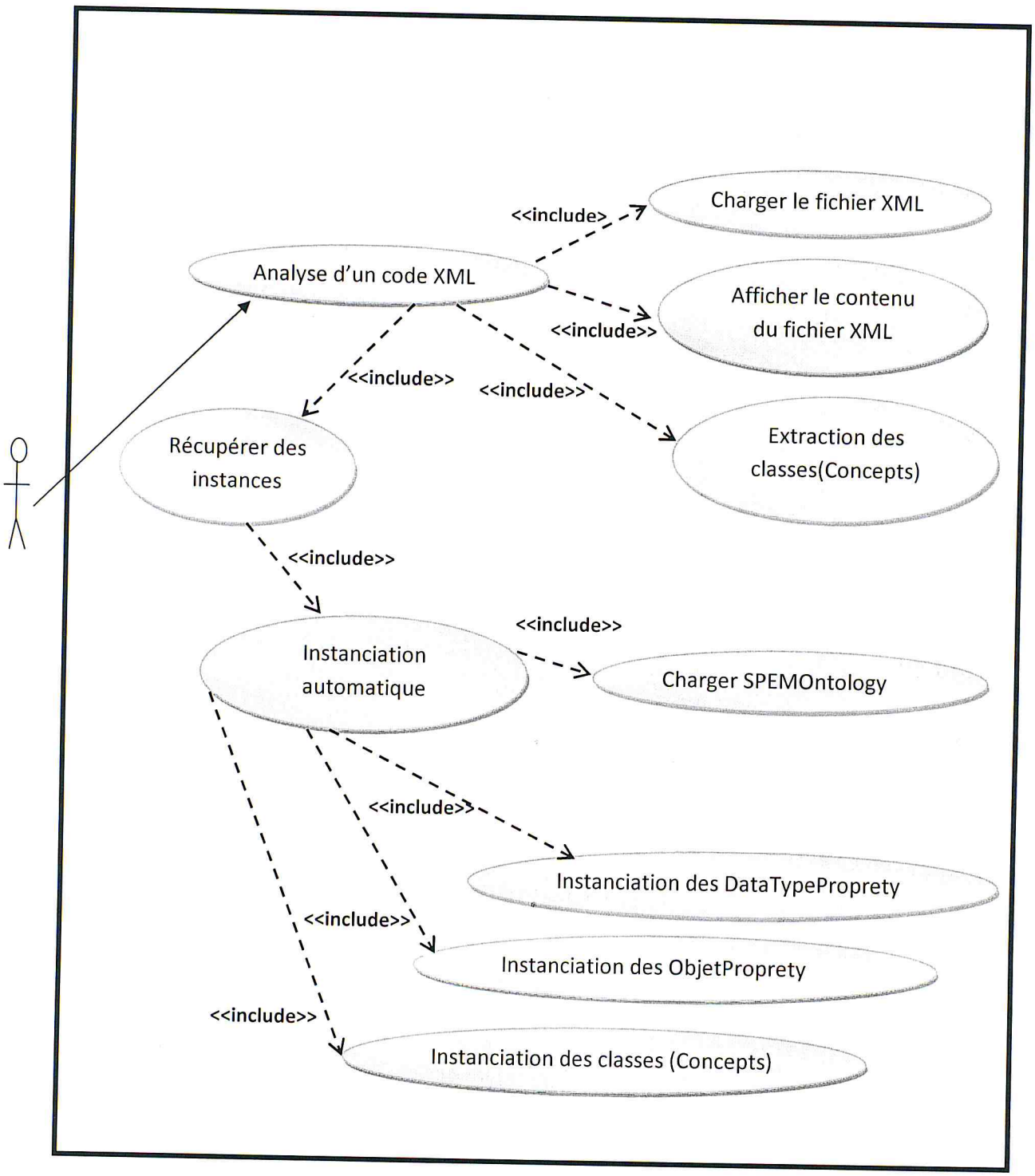


Figure 4.2 Diagramme de cas d'utilisation.

8. Les diagrammes de séquences

8.1 Diagramme de séquence du chargement du fichier XML

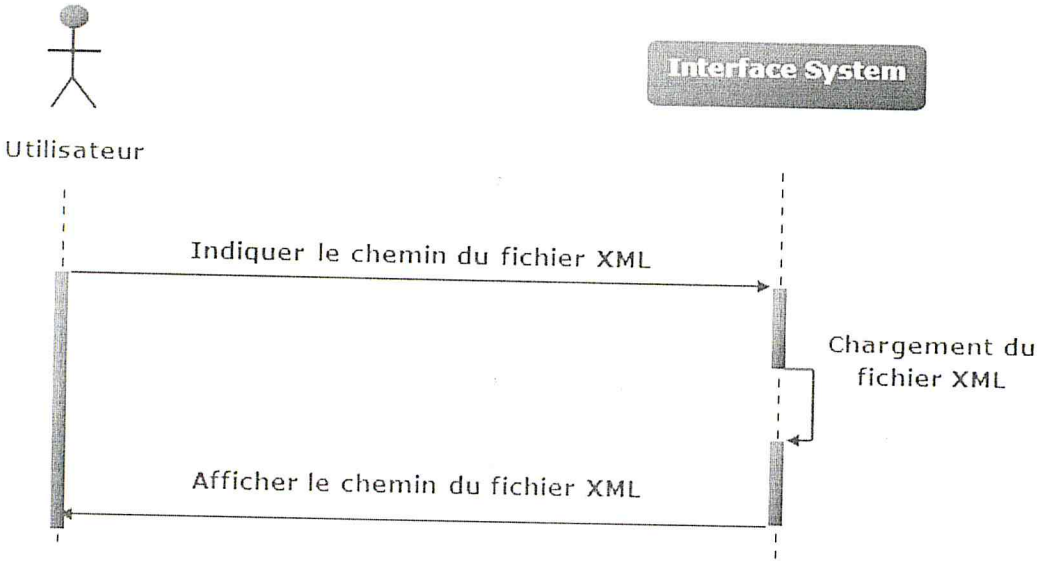


Figure 4.3: Diagramme de séquences pour le chargement du code XML

Cas	Chargement du Fichier XML
Résumé	Chargement du Fichier XML
Acteur principal	Gestionnaire de procédés logiciels
Résultat	Code XML chargé et affiché
Description	1. L'utilisateur indique le chemin du fichier XML 2. Le système charge le fichier XML 3. Le système affiche le code XML.

Tableau 4.3 : Chargement et affichage du fichier XML

8.2 Diagramme de séquence Extraction des classes(Concepts)

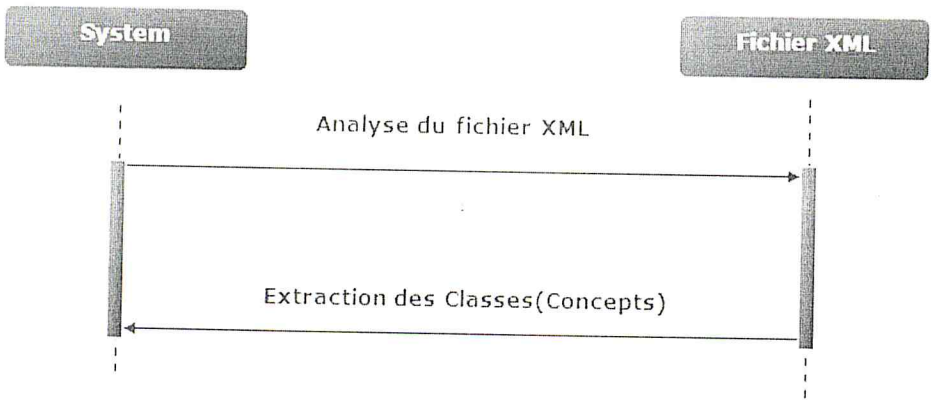


Tableau 4.4 : Extraction des classes (Concepts)

Cas	**Extraction des classes(Concepts)**
Résumé	**Extraction des classes (Concepts)**
Acteur principal	Système
Résultat	Des classes (Concepts) extraites
Description	1. Analyser le fichier XML 2. Le système extrait les classes(Concept)

Tableau 4.4 : Extraction des classes(Concepts)

8.3 Diagramme de séquence du chargement de SPEMOntologie

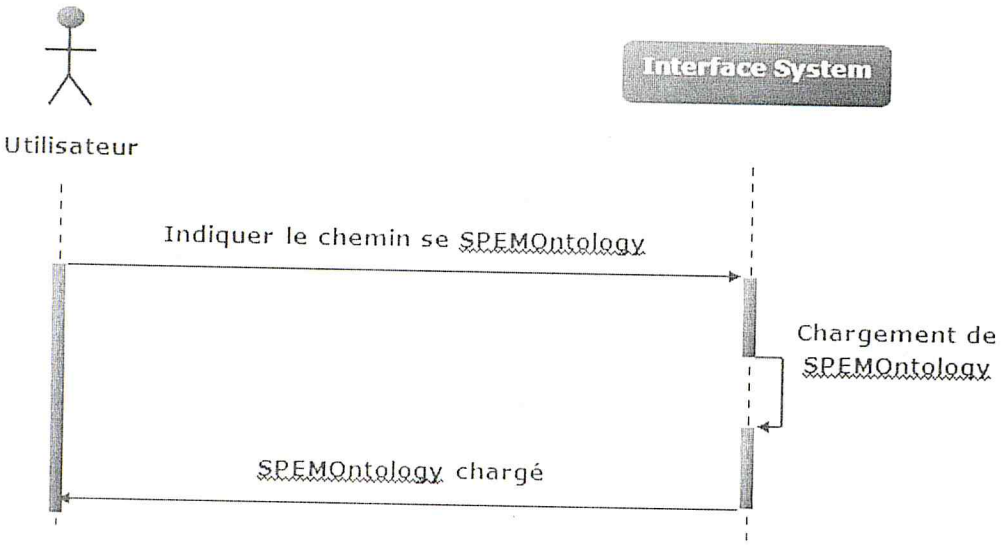


Figure 4.5 : Diagramme de séquences pour le chargement de SPEMOntology.

Cas	**Chargement de SPEMOntology**
Résumé	Charger l'ontologie SPEMOntology
Acteur principal	Gestionnaire de procédés logiciels
Résultat	SPEMOntology chargée
Description	1. Le Gestionnaire de procédés logiciels indique le chemin de SPEMOntology. 2. Le système charge SPEMOntology.

Tableau 4.5 : Chargement de SPEMOntology.

8.4 Diagramme de séquence d'Instanciation des classes (Concepts)

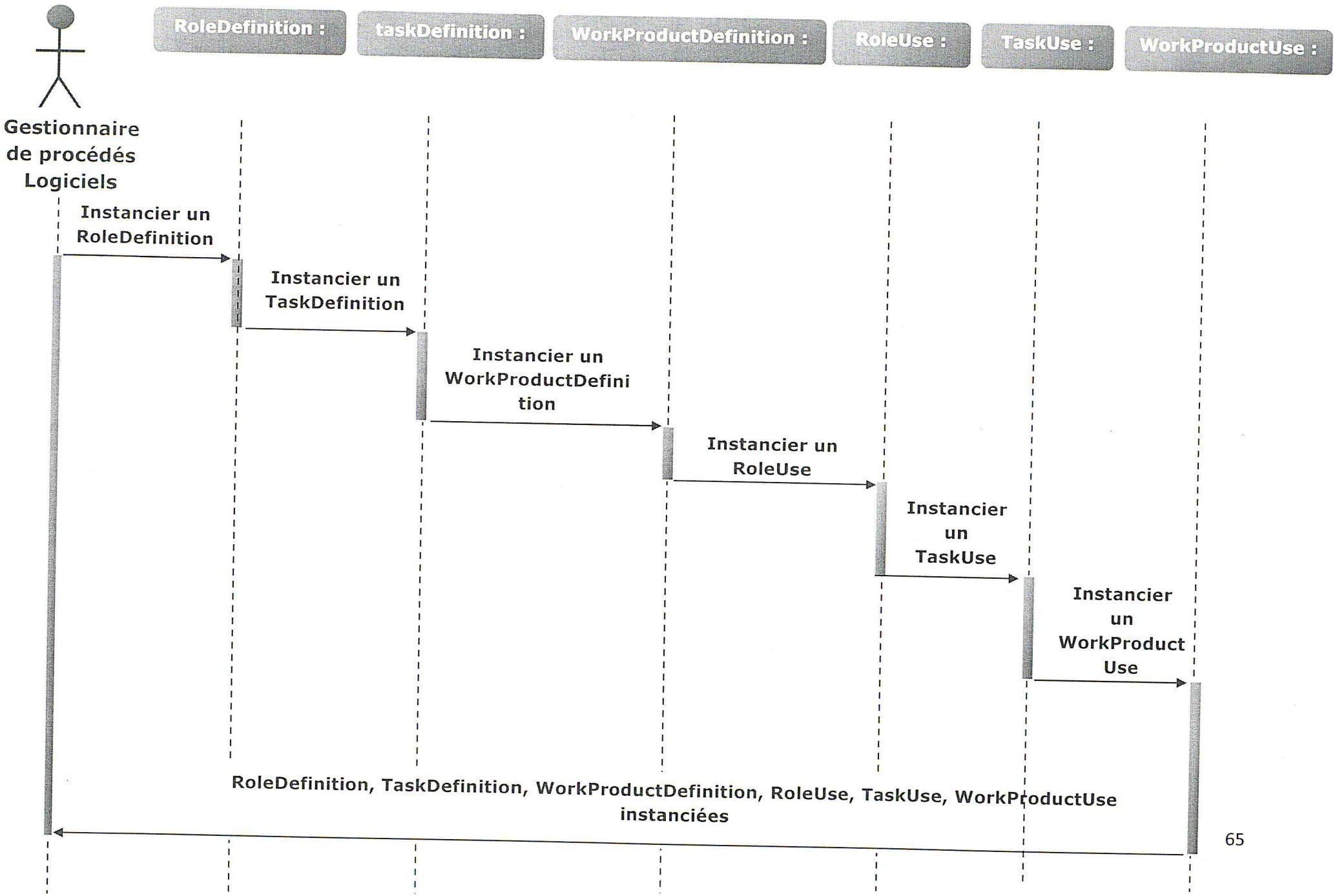

RoleDefinition :
taskDefinition :
WorkProductDefinition :
RoleUse :
TaskUse :
WorkProductUse :
Gestionnaire de procédés Logiciels
Instancier un RoleDefinition
Instancier un TaskDefinition
Instancier un WorkProductDefinition
Instancier un RoleUse
Instancier un TaskUse
Instancier un WorkProduct Use
RoleDefinition, TaskDefinition, WorkProductDefinition, RoleUse, TaskUse, WorkProductUse instanciées

Figure 4.6 : Diagramme de séquences pour l'instanciation des classes (Concepts).

Cas	l'instanciation des classes (Concepts)
Résumé	l'instanciation des classes (Concepts)
Acteur principal	System
Résultat	SPEMOntology Instancié
Description	1. Le système instance les RolesDefinition. 2. Le système instance les TasksDefinition 3. Le système instance les WorkProductDefinition 4. Le système instance les RolesUse 5. Le système instance les TasksUse 6. Le système instance les WorkProductUse 7. Le système récupère tous les instances

Tableau 4.6 : instanciations des classes (Concepts)

8.5 Diagramme de séquence d'Instanciation des DataTypeProprety

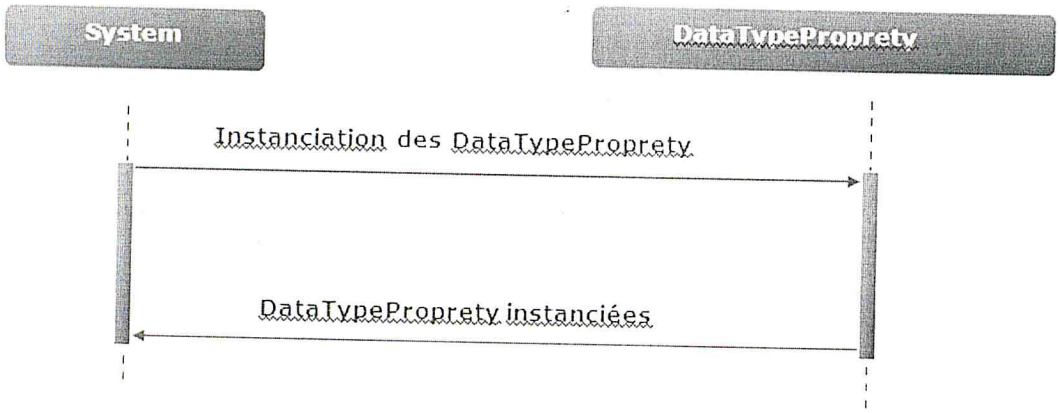


Figure 4.7 : Diagramme de séquences l'instanciation des DataTypeProprety

Cas	instanciation des DataTypeProprety
Résumé	instanciation des DataTypeProprety
Acteur principal	System
Résultat	DataTypeProprety instanciés
Description	1. Le système instance les DataTypeProprety 2. Le système reçoit l'accusé de l'instanciation

Tableau 4.7 : l'instanciation des DataTypeProprety

8.6 Diagramme de séquence d'Instanciation des Objetpreprety

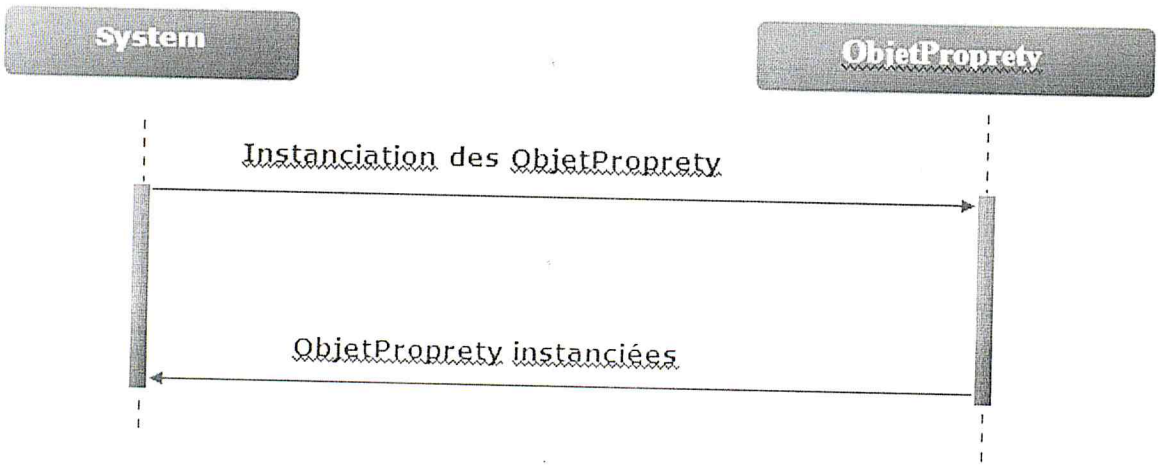


Figure 4.8 : Diagramme de séquences l'instanciation des Objetpreprety

Cas	instanciation des Objetpreprety
Résumé	instanciation des Objetpreprety
Acteur principal	System
Résultat	Objetpreprety instanciés
Description	1. Le système instance les Objetpreprety 2. Le système reçoit l'accusé de l'instanciation

Tableau 4.8 : l'instanciation des Objetpreprety

8.7 Diagramme de séquence Globale de l'instanciation de SPEMOntology

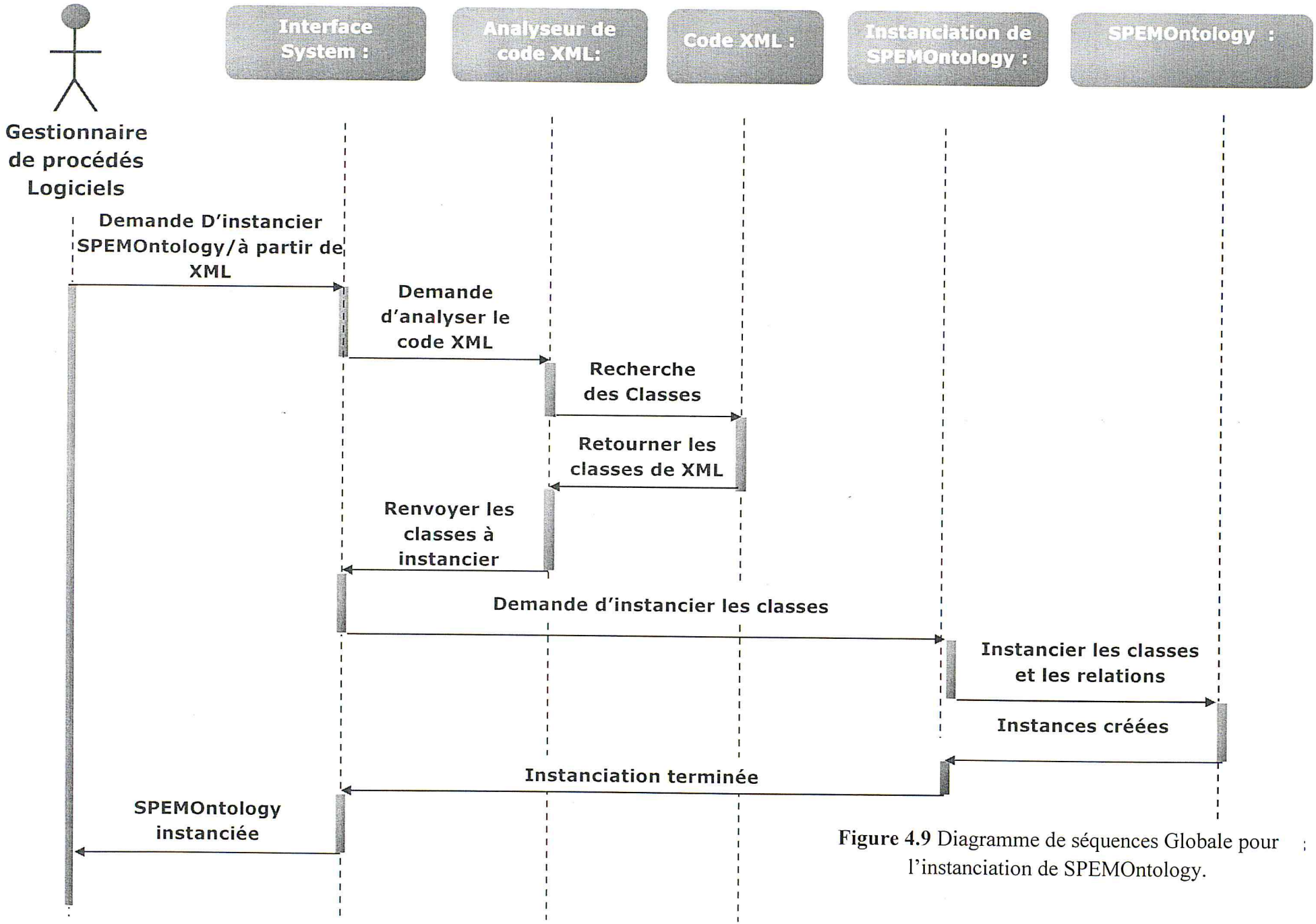


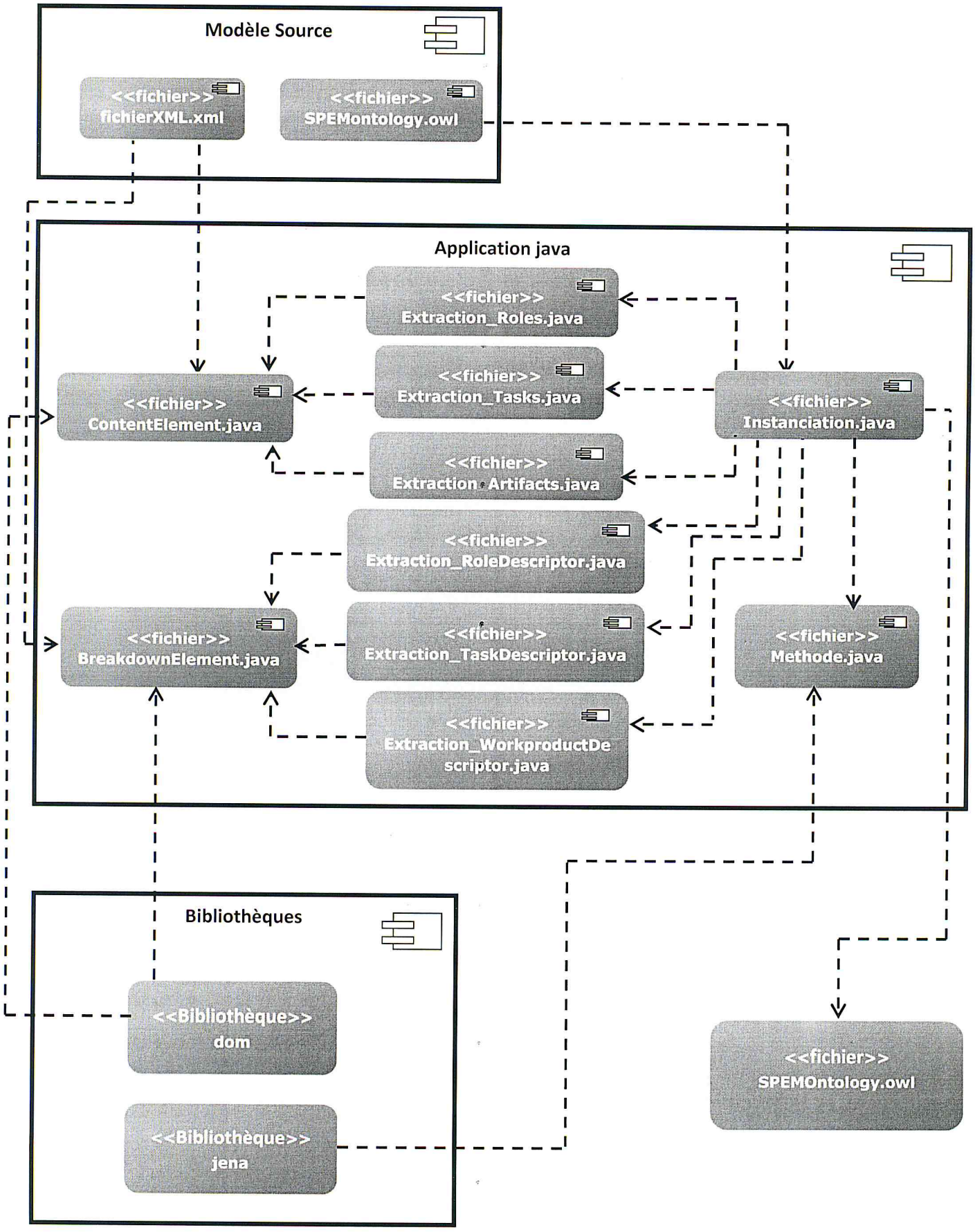
Figure 4.9 Diagramme de séquences Globale pour l'instanciation de SPEMOntology.

Cas	**Instanciation des SPEMOntology**
Résumé	Instancier à partir de l'ontologie SPEMOntology
Acteur principal	Gestionnaire de procédés logiciels
Résultat	SPEMOntology instanciée
Description	1. Le gestionnaire de procédés logiciels demande d'instancier automatiquement SPEMOntology à partir du code XML 2. Le système demande d'analyser le code XML 3. Le système recherche des classes dans le code XML 4. Le système retourné les classes de XML 5. Le système instancie les classes et les relations trouvées dans le code XML dans l'ontologie SPEMOntology

Tableau 4.9 : Instanciation de SPEMOntology.

9. Diagramme de Composants :

Figure 4.10 Diagramme de composants



10. Conclusion :

Nous avons présenté dans ce chapitre la conception de notre application. Elle a été faite suivant la démarche en cascade en adoptant le langage UML.

Nous avons commencé par décrire les cas d'utilisation qui sont la base de la démarche en cascade. Ensuite, pour chaque cas, nous avons établi le diagramme de séquences correspondant. Enfin, nous avons présenté les fichiers sources, les bibliothèques java utilisées et ceci grâce au diagramme de composants.

Chapitre 5
Implementation

1. Introduction

Nous présentons dans ce chapitre l'application que nous avons développée en se basant sur les solutions proposées lors de la conception et en utilisant le langage de programmation java.

Notre application permet d'analyser un code XML, obtenue à partir d'EPF et la récupération des concepts et des objets à instancier dans l'ontologie SPEMontology. Après avoir récupéré les concepts et des objets, l'instanciation de l'ontologie se fait automatiquement, permettant ainsi d'obtenir une base de connaissances.

Nous présentons également dans ce chapitre les différents outils et langages de programmation utilisés pour la réalisation de l'application, ainsi des captures d'écran de plusieurs phases d'exécution du programme.

2. Outils utilisés

2.1 Eclipse

Incontestablement, Eclipse est le plus populaire des environnements de développement intégré (EDI) dédiés au langage Java. Il réunit en effet en une seule application un plan de travail, des outils de gestion de projets en équipe (mode collaboratif), et des fonctions avancées de débogage. Eclipse peut même être « dopé », au moyen de nombreux plug-ins – comme JESS ou Jena - disponibles sur le Web. [37]

De [38]

Eclipse est un environnement de développement intégré libre extensible, universel et polyvalent, permettant de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

La spécificité d'Eclipse IDE vient du fait de son architecture totalement développée autour de la notion de plugin (en conformité avec la norme OSGi) : toutes les fonctionnalités de cet atelier logiciel sont développées en tant que plug-in.

Plusieurs logiciels commerciaux sont basés sur ce logiciel libre, comme par exemple *IBM Lotus Notes 8*, *IBM Symphony* ou *WebSphere Studio Application Developer*.

2.2 Protégé

Protégé [39] est un IDE d'ontologies distribué en open source par l'université de Stanford (spécialisée en informatique médicale). Protégé, c'est un éditeur "de base", modulaire et extensible via plugins. Grâce à ces plugin, il est capable de gérer d'autres ontologies, comme OWL. (figure 5.1)

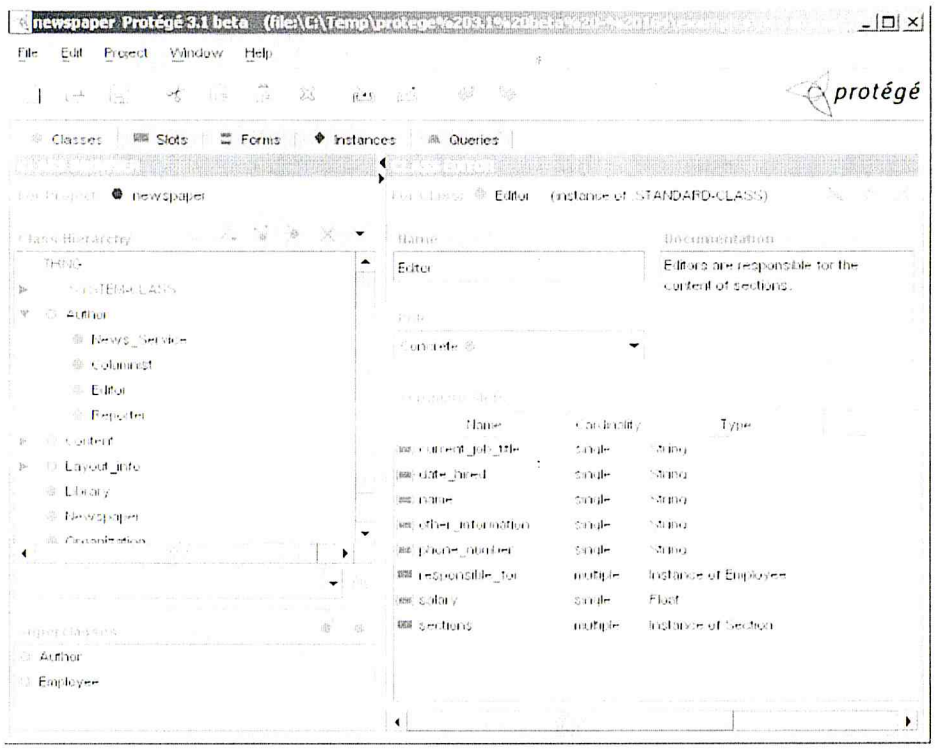


Figure 5.1 : Protégé version 3.1

Protégé offre trois approches pour développer une ontologie :

- l'éditeur Protégé-OWL permet de construire des ontologies pour le web sémantique (Protégé 3.4.2 et Protégé 4.0.2),
- l'éditeur Protégé-Frames permet de construire et peupler des bases de connaissances basées sur des frames (Protégé 3.4.2),
- l'équipe de développement de Protégé, travaille également sur l'édition d'ontologies à travers une interface web (WebProtégé 0.5 alpha).

Protégé permet d'utiliser le langage OWL, qui est le langage informatique utilisé pour modéliser des ontologies. OWL est un dialecte XML qui étend RDFS. Il permet aussi un fonctionnement en mode client/serveur avec une gestion des droits ajustables en fonction des profils souhaités. Protégé intègre des outils de travail collaboratif comme le chat ou les annotations.

Les fonctionnalités de Protégé peuvent être étendues grâce à une architecture en plugin et à l'aide de l'API Java fournie. Ce qui permet, en outre, de générer automatiquement du code Java.

3. Bibliothèques java et langages de programmation utilisés

3.1 JAVA [40]

La naissance de java remonte à l'année 1991. A cette époque, des ingénieurs de chez SUN ont cherché à concevoir un langage applicable à de petits appareils électriques (code embarqué). Pour ce faire, ils se sont fondés sur une syntaxe très proche de celle de C++ ; en prenant le concept de machine virtuelle déjà exploité auparavant par le PASCAL UCSD.

L'idée consistait à traduire d'abord un programme source, non pas directement en langage machine, mais dans un pseudo langage universel, disposant des fonctionnalités communes à toutes les machines. Ce code intermédiaire est formé de byte codes se trouve ainsi compact et portable sur n'importe quelle machine, il suffit juste qu'elles disposent de la machine virtuelle.

Java et la programmation orientée objet

La programmation orientée objets possède de nombreuses vertus universellement reconnues désormais. Notamment, elle ne renie pas la programmation structurée, elle contribue à la fiabilité des logiciels et elle facilite la réutilisation de code existant. Elle introduit de nouveaux concepts, en particulier ceux d'objets, d'encapsulation, de classe et d'héritage.

Nous avons choisi JAVA comme langage de programmation pour les raisons suivantes :

- Java est un langage multi-plateformes qui permet l'écriture d'un code fonctionnant dans tous les environnements. Pour cela il suffit que l'environnement possède une JVM (Java Virtual Machine).
- Java est un langage orienté objets, simple, qui réduit le risque d'erreurs d'incohérence.
- Java est dotée d'une riche bibliothèque de classes, comprenant la gestion des exceptions.
- Les différentes versions de JDK (Java Development Kit), fournies gratuitement par Sun, contiennent l'ensemble des éléments permettant le développement, la mise au point et l'exécution des programmes Java.
- Notre travail fait partie d'une chaine de projets réalisés en parallèle, coordonner tous les travaux effectués et les intégrer nous a obligé à utiliser Le langage JAVA qui est d'ailleurs le langage utilisé dans tout ces travaux.

3.2 Jena

De [44]

Jena est l'un des API les plus utilisés de Java pour RDF et OWL, fournissant des services pour la représentation du modèle, l'analyse, la persistance de base de données, l'interrogation et quelques outils de visualisation.

Cette bibliothèque Java facilite le développement des applications spécialisées dans la web sémantique, ce qui lui permet de manipuler, stocker des connaissances RDF ou XML en mémoire ou disque dur, et aussi gérer les ontologies RDF-Schéma, OWL et inférer sur leurs connaissances.

3.3 Dom

De [45]

Le **DOM** (Document Object Model) est un standard proposé par W3C (World Wide Web Consortium), a pour but d'accéder et manipuler des documents XML.
Le DOM est séparé en 3 parties différentes / niveaux:

- Core DOM - modèle standard pour tout document structuré
- XML DOM - modèle standard pour les documents XML
- DOM HTML - modèle standard pour les documents HTML

4. Présentation de l'application

Nous avons développé une application qui permet :

- L'analyse et l'extraction d'un code XML
- L'instanciation automatique de SPEMOntology à partir de XML/ EPF.

Pour comprendre le fonctionnement de notre application, on prendra **ISPW-6** du chapitre 3 comme exemple, et quelque pris d'écran de notre application seront faitent.

Pour cela, on prendra la Tache Modify design comme exemple, pour suivre les démarches de notre application.

4.1 Analyse et extraction d'un code XML :

Cette phase de l'application permet le chargement et l'extraire des informations à partir du code XML, c'est ce qu'on appelle aussi la rétro-ingénierie.

Un **procédé logiciel** est un ensemble des tâches, elles contiennent des **produits** en entrée et d'autres en sortie, et elles sont réalisées par des **rôles** qui peuvent être des agents humains ou logiciels.

Pour que notre application soit plus lisible on a divisé l'extraction des rôles, des tâches et des produits : la classe **Extraction_Roles.java** pour les rôles, la classe **Extraction_Artifacts.java** pour les produits et la classe **Extraction_Tasks** pour les tâches ainsi pour les classes **Use**

Pour faire la rétro-ingénierie d'un code XML, nous avons développé les classes précédentes pour les deux supers classe **ContentElement** et **BreakDownElement** .

Pour l'extraction des concepts et leurs propriétés on doit charger le fichier XML dans notre interface, pour cela il faut cliquer sur **File** puis choisir **Open** et bien indiquer le chemin du fichier XML (*.XML) (figure 6.2, figure 6.3), on utilise La méthode **parseXmlFile(String chemin)** .

Après on a utilisé les méthodes **getRole()** qui se trouve dans la classe **Extraction_Roles.java, getTak()** qui se trouve dans la classe **Extraction_tasks.java** et **getArtifact()** qui se trouve dans la classe **Extraction_Artifacs.java,** pour analyse le code XML, et l'extractions des concepts et leurs propriétés.

Après l'extractions des concepts avec leurs propriétés, on utilise la méthode **parseDocument()** qui se trouve sur chacune des classes précédentes, qui récupère les données extraites par la méthode **getRole**(), **getTak() et Extraction_tasks** et les regroupées dans des listes pour facilités l'instanciation de notre ontologie, comme c'est précisé dans la **Figure 5.3**

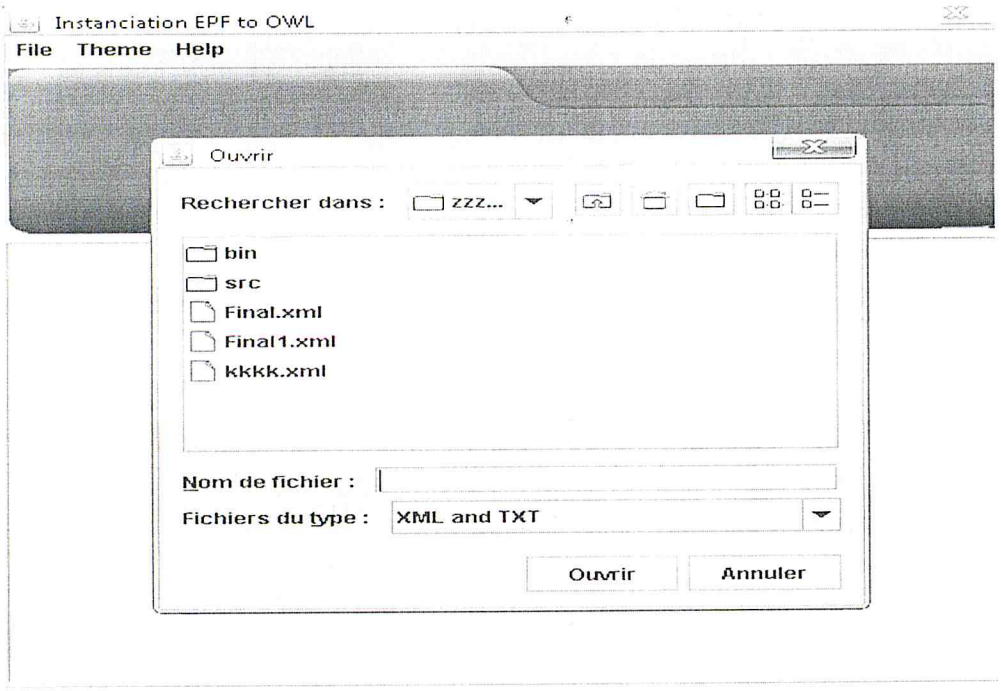


Figure 5.2 : Chargement du fichier XML

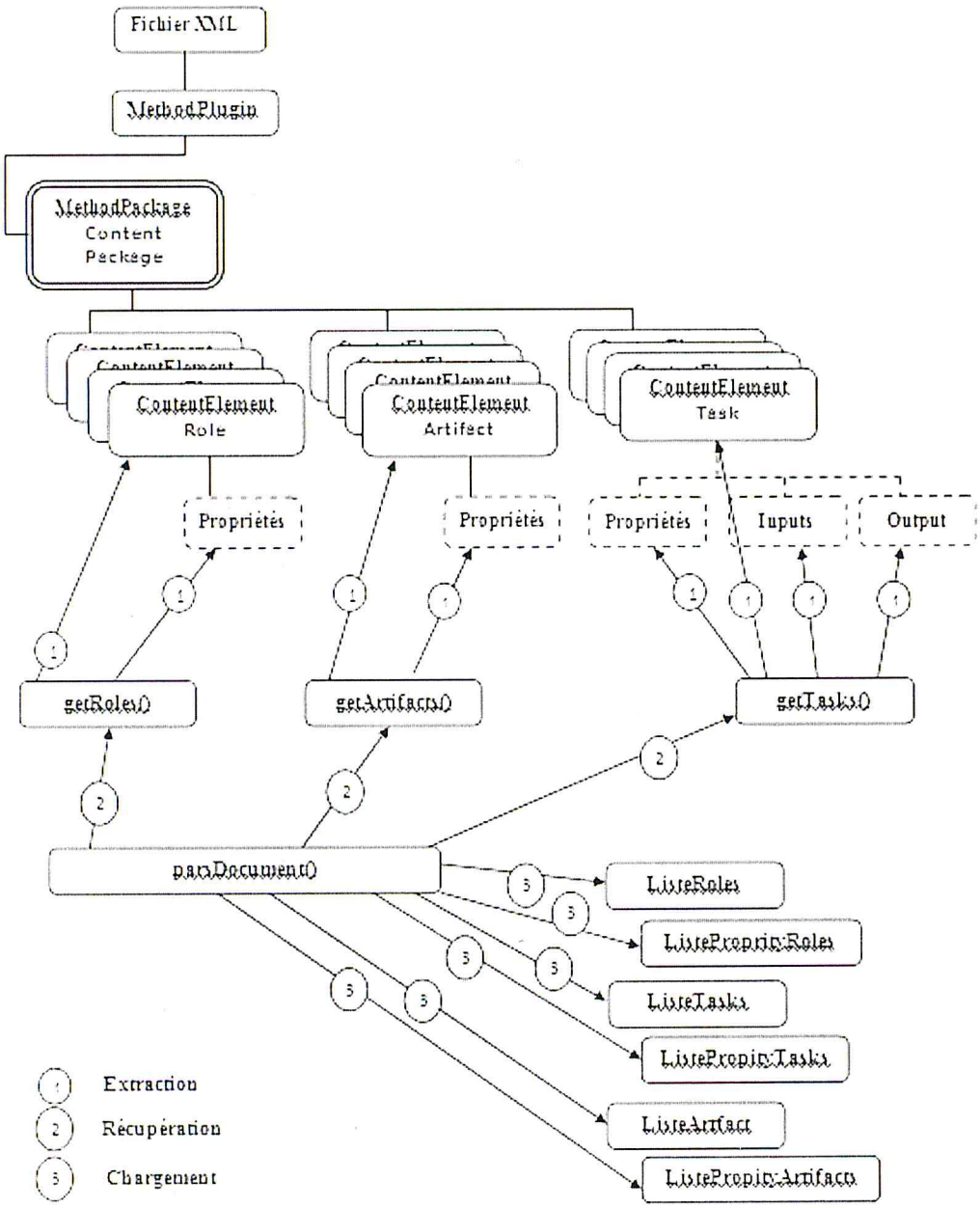


Figure 5.3 : Organigramme d'Extraction

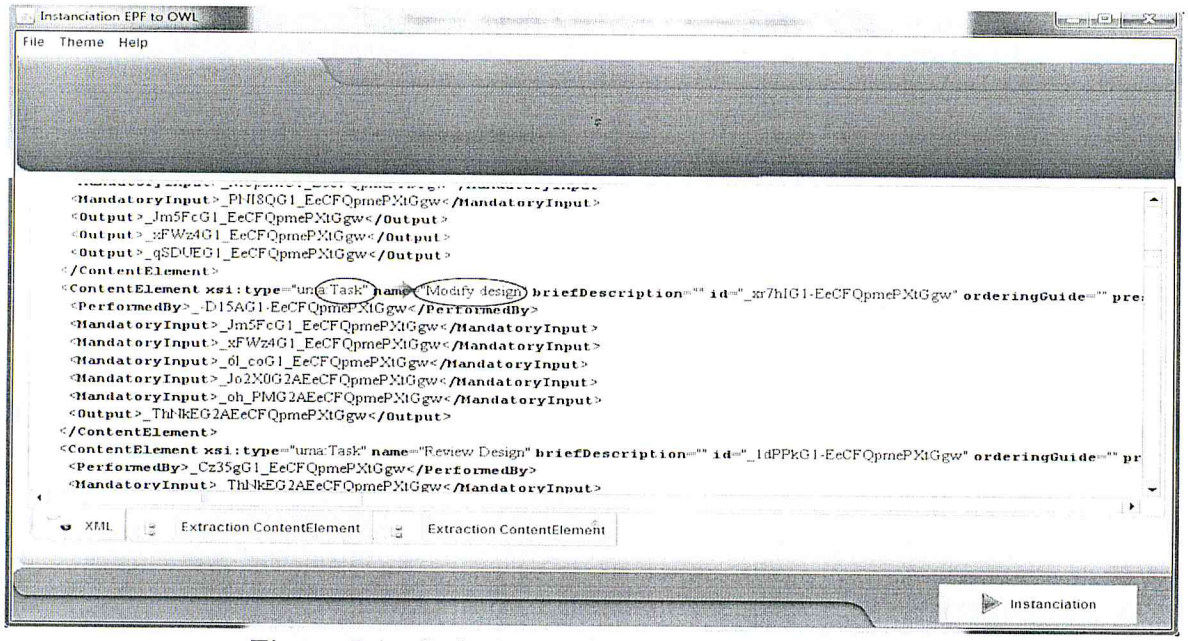


Figure 5.4 : Code XML chargé dans l'interface

Après avoir regroupé toutes les taches avec leurs rôles et leurs produits en entrée et en sortie dans des listes, nous faisons appel à la méthode **getJTree()** qui se trouve dans la classe **Principale.java** pour mieux visualiser notre application, pour bien comprendre le fonctionnement de notre application , on prendra la tache **Modify design** comme exemple et la suivre tout au long de notre application.

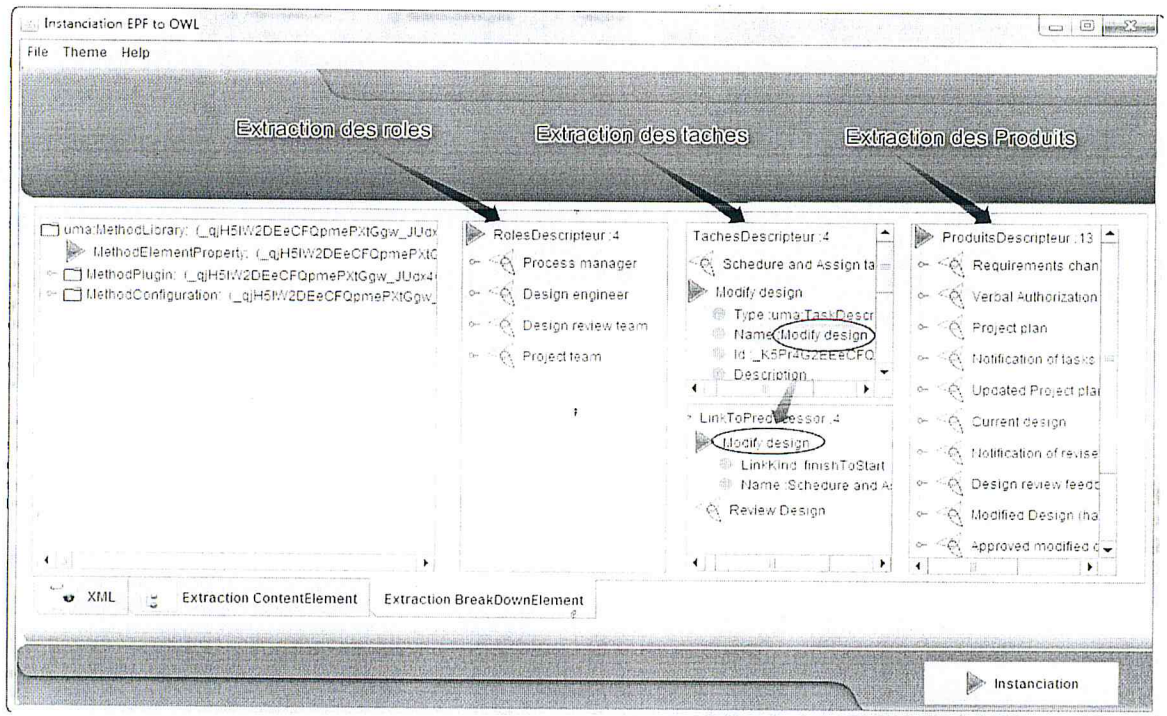


Figure 5.5 : Extraction des Rôles, Tâches et Produits

On remarquera que tous les concepts son extraité avec leur propriétés, et spécialement notre exemple, la tache **Modify design** est extraite avec ses propriétés comme sur la **Figure 5.5**

4.2 Instanciation de SPEMOntology

L'instanciation de l'ontologie SPEMOntology peut se faire en utilisant l'éditeur d'ontologies Protégé, mais nous avons développé une application qui fait l'instanciation de manière automatique.

L'instanciation de SPEMOntology est réalisée en utilisant la bibliothèque java : Jena.Nous avons développé une classe java nommée : **Methode.java** (**Figure 5.9**) qui comportant un ensemble de méthodes permettant d'instancier les différents objets définis dans l'ontologie et de les sauvegarder dans le fichier SPEMOntology.owl correspondant à l'ontologie. La méthode permettant de sauvegarder toute instance créée est la méthode :
writeOntModelOnOWLFile(). Cette méthode est appelée lorsque que l'utilisateur clique sur le bouton **Instanciation** afin de valider toutes les instances qu'il vient de créer.

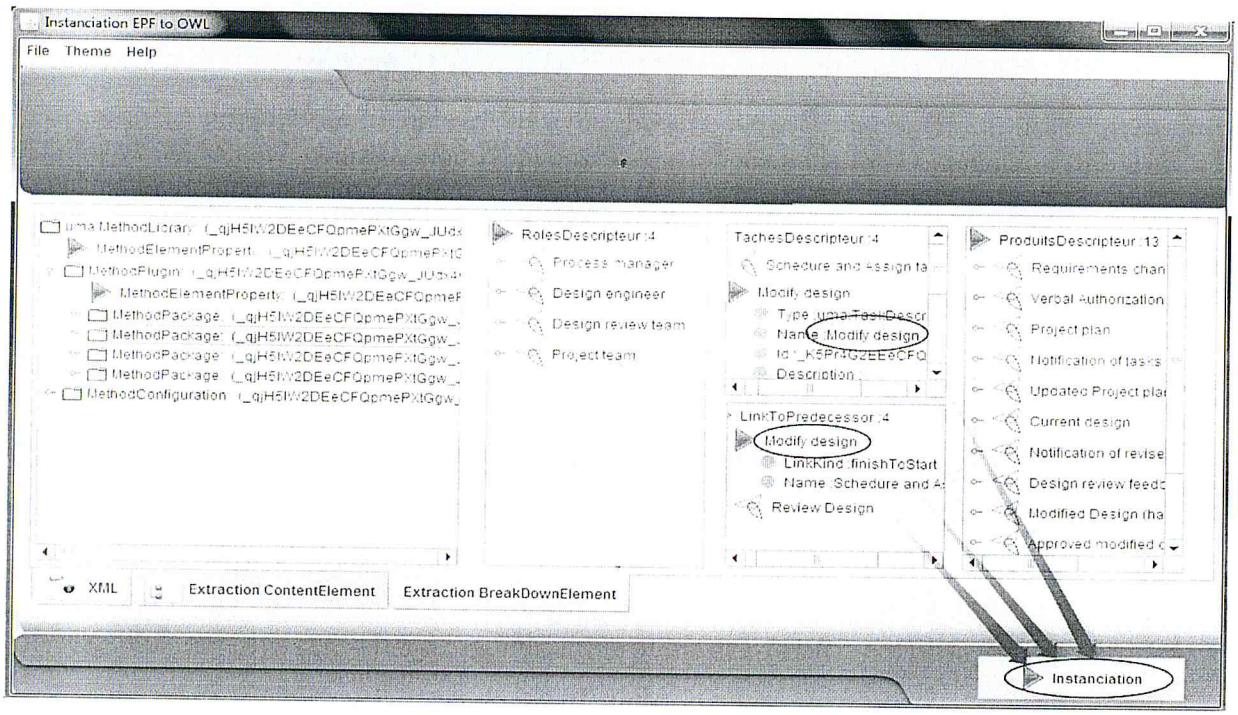


Figure 5.6 : Instanciation de SPEMOntologie

La méthode permettant de créer un individu d'une classe OWL de SPEMOntology est la méthode : **creerIndividu(String individu, String nomClasse).**

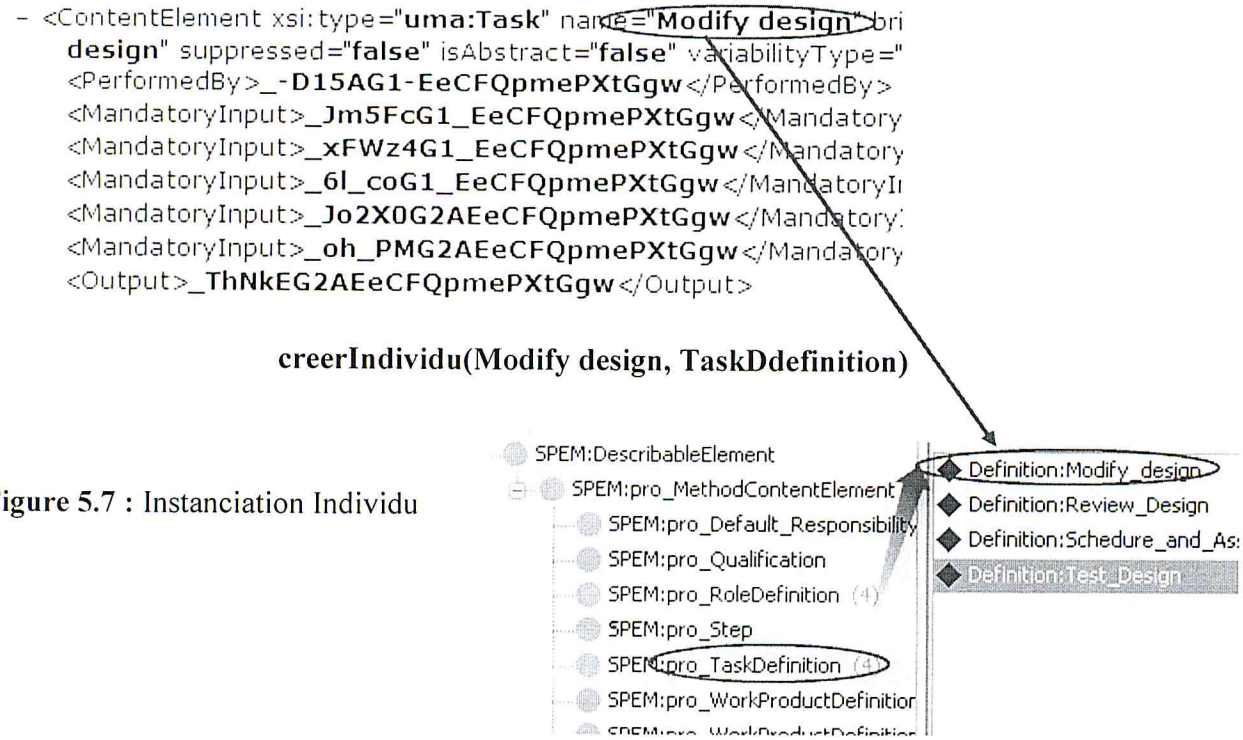


Figure 5.7 : Instanciation Individu

La méthode permettant d'assigner une valeur à un **DataTypeProperty** pour un individu donné est la méthode : **creerInstanceDataTypeProperty (StringnomDataTypeProp, String valeur, String individu).**

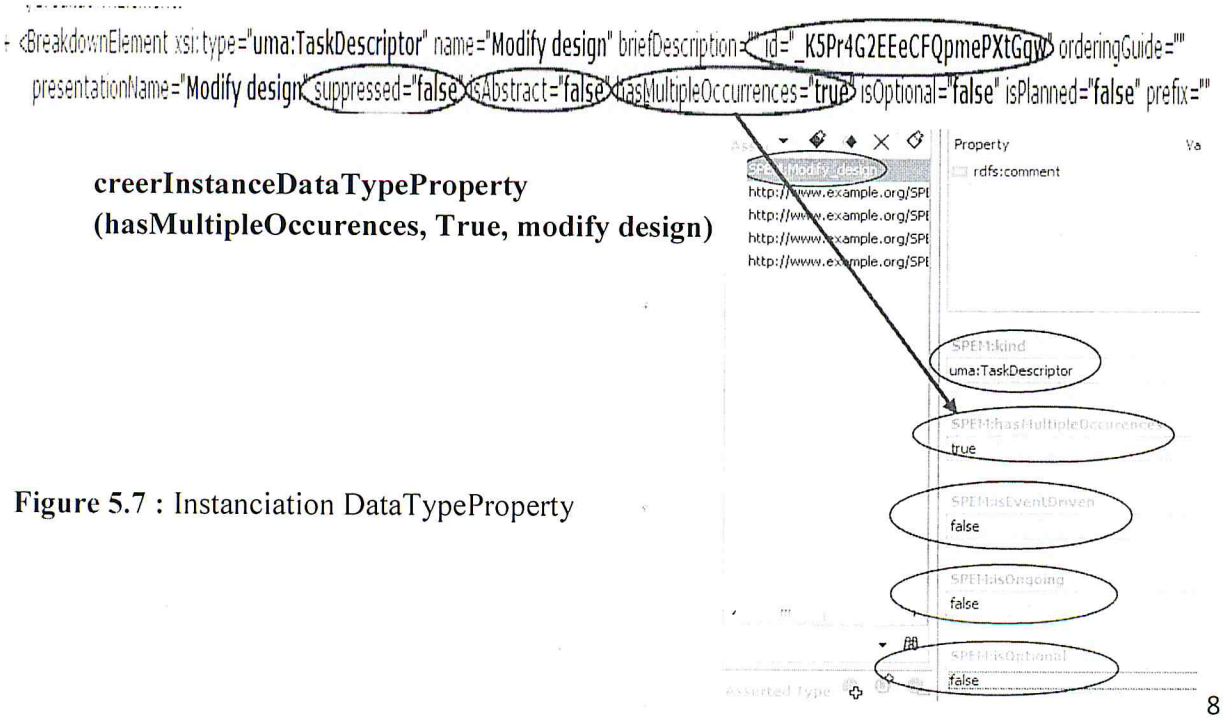


Figure 5.7 : Instanciation DataTypeProperty

La méthode permettant de relier deux individus par un **ObjectProperty** est la méthode **creerInstanceObjectProperty(String nomObjectProp, String classeDomaine,String classeRange).**

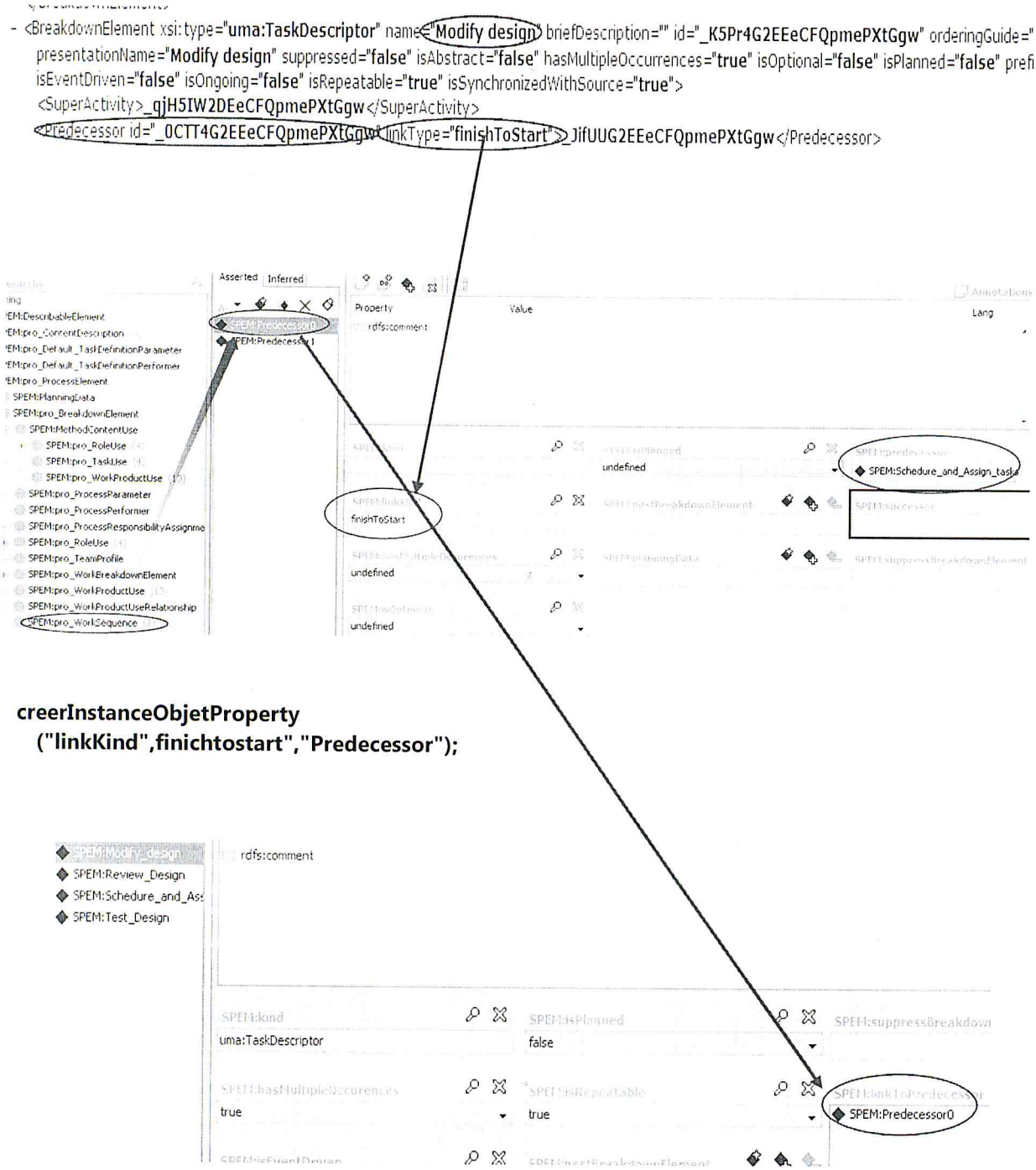


Figure 5.8 : Instanciation ObjectProperty

Donc, quand on ouvre la classe **Worksequence**, on trouvera les attribue de la propriété **Predecessor** : la tache qui précède la tache **Modify design** qui et **Schedure_and_Assign_task** et le type de lien qui est **finishToStart.**

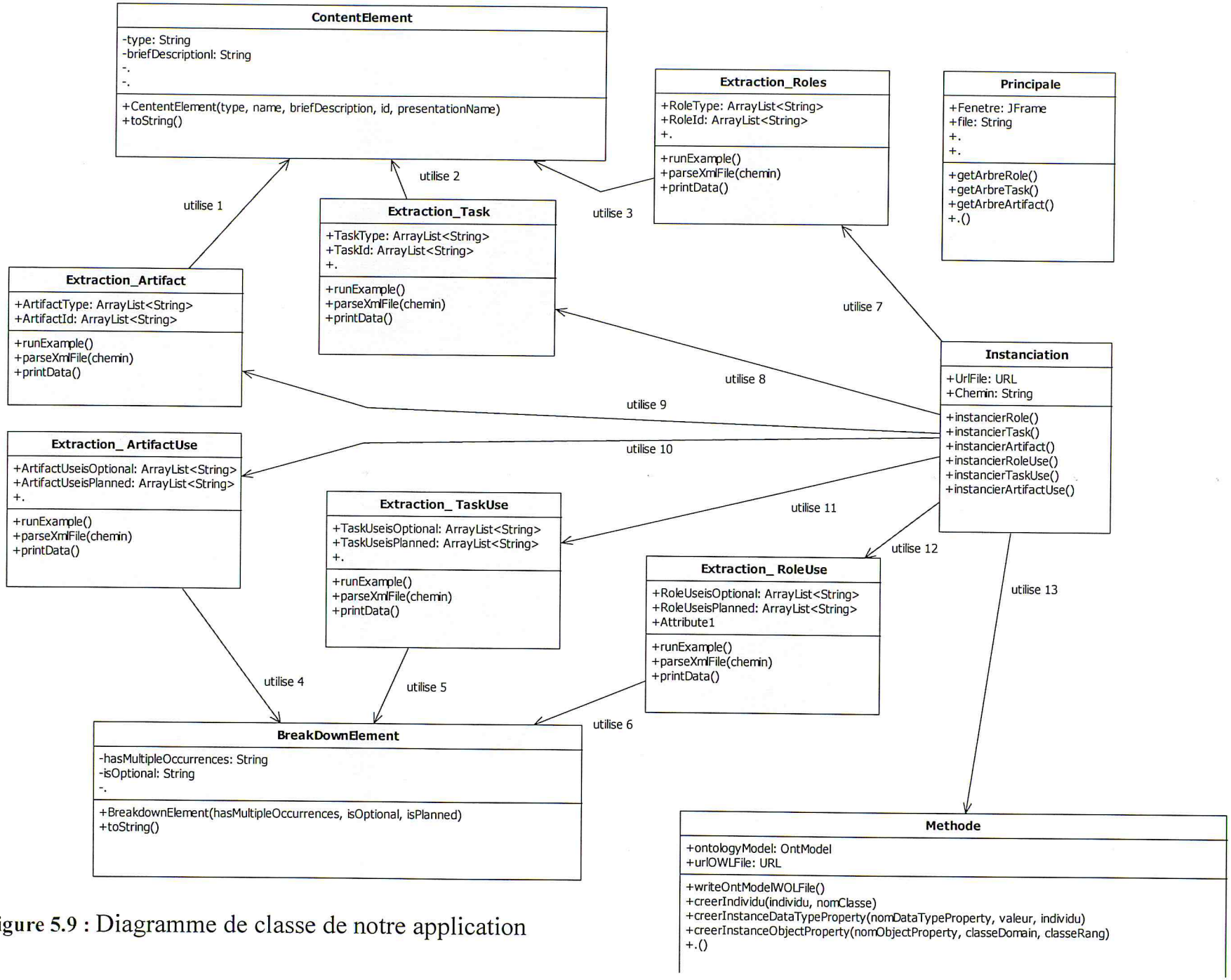


Figure 5.9 : Diagramme de classe de notre application

5. Conclusion

Nous avons présenté dans ce chapitre l'application que nous avons développée, ainsi que les langages et les outils que nous avons trouvés nécessaires d'utiliser. Nous avons décrit aussi toutes les fonctionnalités de l'application, avec des illustrations de notre interface pour chaque étape d'exécution

CONCLUSION GENERALE

CONCLUSION GENERALE

Le travail effectué intervient principalement dans le domaine de l'ingénierie des procédés logiciels et plus exactement dans la réutilisation des procédés logiciels. Ce travail nous a permis d'élargir nos connaissances dans les domaines du génie logiciel et de l'intelligence artificielle.

En effet, notre travail a pour objectif l'implémentation d'une partie d'une approche proposée pour la réutilisation des PLs. La solution proposée nous a permis de nous familiariser avec plusieurs domaines : les procédés logiciels et les ontologies.

La solution proposée repose sur l'exploitation d'une ontologie de domaine qui regroupe les concepts concernant les PLs ; le travail demandé est de proposer une solution pour l'instanciation automatique de cette ontologie à partir de modèles de PLs déjà existants.

Pour arriver a cet objectif nous avons étudié l'ontologie concernée en détaille, ses concepts, ses propriétés et son organisation.

Les modèles candidats à la réutilisation sont des modèles décris sous EPF (Eclipse ...), ces modèles sont générés en XML. Ainsi pour bien comprendre ces modèles nous avons étudié en détail EPF et appris à générer les modèles de PLs à partir de ce Frame Work.

La difficulté du travail lors de l'instanciation automatique des modèles EPF est de trouver les bonnes correspondances entre les concepts de l'ontologie et les concepts du fichier XML. En effet, malgré que les modèles EFP et l'ontologie générée respectent le métamodèle SPEM (les même concepts). Lors de la génération de l'ontologie certains noms de concepts ou de propriétés ont été modifié lors de la génération de l'ontologie ce qui nous posé des problèmes de correspondance.

Comme perspective de travail, nous pensons qu'il est possible de généraliser la solution en permettant d'introduire l'instanciation à partir de plusieurs modèles de PLs décrit avec différent langage de modélisation de procédé, l'idée est de regrouper les comportements, concepts communs des langages de modélisation de procédés logiciels et lors de l'utilisation de l'application, de configurer l'application selon le type du langage de modélisation de PL.

Bibliographie

[1] : Sonia Jamal,
Environnement de procédé extensible pour l'orchestration Application aux services web,
Thèse de doctorat de l'université Joseph Fourier de Grenoble. Décembre 2005.

[2] : Ma. Lizbeth Gallardo,
Une approche à base de composants pour la modélisation des procédés logiciels,
DEA, préparé dans l'équipe ADELE, au laboratoire LSR, Septembre 2000.

[3] : Amiour Mahfoud,
Vers une fédération de composants interopérables pour les environnements centrés procédés logiciels,
Thèse de doctorat de l'université Joseph Fourier, Grenoble I, juin 1999.

[4] : Hanh Nhi Tran,
Modélisation de procédés logiciels à base de patrons réutilisables,
Thèse de doctorat de l'université Toulouse II- Le Mirail, Novembre 2007.

[5] : Sergio Garcia-Camargo,
Ingénierie Concurrente en Génie Logiciel : Céline,
Thèse de doctorat de l'université Joseph Fourier de Grenoble. Décembre 2006.

[6] : Kamal Zuhairi Zamli,

A SURVEY AND ANALYSIS OF PROCESS MODELING LANGUAGES,

Universiti Sains Malaysia, 2001.

[7] : Olfa Djebbi,
MDA : Vers l'industrialisation de la construction d'applications réparties,
DEA SIR (Systèmes Informatiques Répartis), LIP6 (Laboratoire d'informatique Paris 6), 2004.

[8] : Sylvain Andre,
MDA (Model Driven Architecture) principes et états de l'art,
Examen probatoire du diplôme d'ingénieur C.N.A.M. en informatique, option ingénierie et intégration informatique : système de conduite, Novembre 2004.

[9] : Kamal Zuhairi Zamli,

A SURVEY AND ANALYSIS OF PROCESS MODELING LANGUAGES,

Universiti Sains Malaysia, 2004.

[10] : Profiles UML et langage J : Contrôlez totalement le développement d'applications avec UML.
White Paper – UML Profile Builder. © Softeam 1999.

[11] : Constantin Werner,
UML Profile for Communicating Systems. A New UML Profile for the Specification and Description of Internet Communication and Signaling Protocols,
Thèse de doctorat de l'université Georg-August, 2006.

[12] : Skander Turki,
Ingénierie système guidée par les modèles: Application du standard IEEE 15288, de l'architecture MDA et du langage SysML à la conception des systèmes mécatroniques.
Thèse de doctorat de l'Université du Sud Toulon-Var, Novembre 2008.

[13]: Chikhi Imane, Daouadji Hadjar, Génération d'une ontologie OWL à partir du méta-modèle SPEM. Université de Blida, 2008/2009.

[14]: Benoît Combemale,
Spécification et vérification de modèle de procédés de développement.
Master de recherche, université de Toulouse, Juin 2005.

[15] : Benoît Combemale,
Approche de métamodélisation pour la simulation et la vérification de modèle,
Application à l'ingénierie des procédés, Thèse de doctorat de l'université de Toulouse, France, 2008.

[16]: B. Combemale, X. Crégut, A. Caplain, B. Coulette, A. Garcia,
Vers une vérification d'un procédé de développement modélisé en SPEM.

[17]: Hanh Nhi Tran,
Modélisation de procédés logiciels à base de patrons réutilisables,
Thèse de doctorat de l'université Toulouse II- Le Mirail, Novembre 2007.

[18]: OMG Adopted Specification. Software & Systems Process Engineering Metamodel Specification, v2.0 (Beta 2).

[19] : Frédéric FÜRST,
Contribution à l'ingénierie des ontologies : une méthode et un outil d'opérationnalisation,
Thèse de doctorat de l'Université de Nantes, Novembre 2004.

[20] : Lortal Gaëlle,
État de l'art Ontologies et Intégration/Fusion d'ontologies,
2002.

[21] : Kaveh Bazargan,
Le rôle des ontologies de domaine dans la conception des interfaces de navigation pour des collections en ligne de musées: évaluations et proposition,

Mémoire de DEA en Management et Technologies des Systèmes d'Information, Université de Genève, Suisse, Juin 2000.

[22] : Amedeo Napoli,
Éléments sur La conception d'ontologies,
LORIA, Strasbourg, Février 2005.

[23] : Mustapha Baziz,
Application des Ontologies pour l'Expansion de Requêtes dans un Système de Recherche d'Informations,
Rapport de DEA de l' Université Paul Sabatier & Institut National Polytechnique de Toulouse, 2002.

[24]: Naçima Mellal,
Réalisation de l'interopérabilité sémantique des systèmes, basée sur les ontologies et les flux d'information,
Thèse de doctorat de l'université de Savoie, Décembre 2007.

[25] : http://nkos.slis.kent.edu/KOS_taxonomy.htm

[26] : Riichiro Mizoguchi,
Le rôle de l'ingénierie ontologique dans le domaine des EIAH,
Institut de recherche scientifique et industrielle, Université Osaka, 2004.

[27] : Frédéric Fürst,
L'opérationnalisation des ontologies : une méthodologie et son application au modèle des Graphes Conceptuels,
Institut de Recherche en Informatique de Nantes.

[28] : N. Zouggar, B. Vallespir, D. Chen,
Enrichissement de la modélisation d'entreprise par les ontologies,
Université Bordeaux1, Avril 2006.

[29] : Natalya F. Noy et Deborah L. McGuinness,
Développement d'une ontologie : Guide pour la création de votre première ontologie,
Université de Stanford.

[30] : Frédéric Bertrand,
Utilisation d'ontologies pour l'intégration de données,
Laboratoire L3i, Université de La Rochelle.

[31] : Delia Codruta Rogozan,
Gestion de l'évolution d'une ontologie : méthodes et outils pour un référencement sémantique évolutif fondé sur une analyse des changements entre versions de l'ontologie,
Télé-Université du Québec.

[32] : Xavier Lacot,
Introduction à OWL, un langage XML d'ontologies Web,
Juin 2005.

[33] : Li Liao, Yuzhong Qu, Hareton K. N. Leung,
A Software Process Ontology and Its Application,
Department of Computer Science and Engineering, Southeast University, Nanjing, Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong.

[33] : Ricardo de Almeida Falbo, Giancarlo Guizzardi, Katia Cristina Duarte,
An Ontological Approach to Domain Engineering,
Computer Science Department, Federal University of Espírito Santo.

[34] :

[35] : Système d'information: Conduite de Projet, Cours de SI Bordeaux 2004

http://dept-info.labri.fr/~counilh/systeme-d-information/SI_0202.pdf

[36] : Pierre-Alain Muller, Nathalie Gaertner, Modélisation Objet avec UML

edition Eyrolles, 2000.

[37] : Programmez en java dans l'environnement ECLIPSE, Edition OReily, 2004 auteur STEVE Holzner.

[38]: Environment Eclipse: http://www.eclipse.org/

[39]: L'éditeur d'ontologie OWL : www. protege.stanford.edu.

[40] : Programmer en java troisième édition. Claude Delannoy édition Eyrolles

[41] : http://www.eclipse.org/epf/.

[42]: F. Aoussat, M. Ahmed-Nacer, M. Oussalah, Reusing Approach for Software Processes based on Software Architectures, ICEIS'10, pages 366-369, 2010

[43] : http://www.w3.org/XML/

[44] : http://jena.sourceforge.net/documentation.html

[45] : http://www.w3.org/DOM/

ANNEXE

1. Définitions des Procédés

Les procédés sont définis comme étant une suite d'étapes réalisées dans un but donné : *"A sequence of steps performed for a given purpose"*.

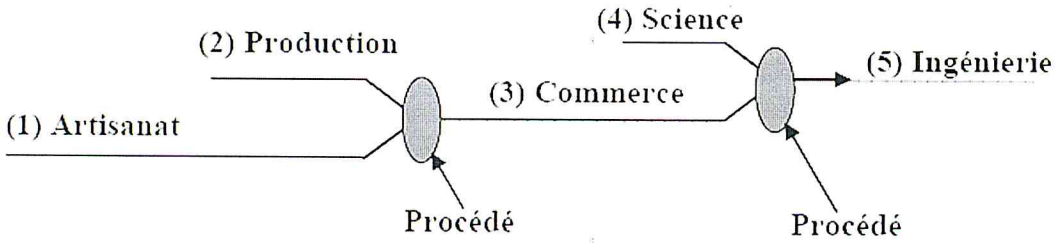


Figure 1.1: L'apparition des procédés dans le monde. [1]

Les procédés sont apparus dans le monde de l'informatique dans les années 80, lorsque les informaticiens se sont aperçus de la répétitivité de certaines suites de tâches, et ont cherché à les formaliser et/ou les automatiser. [1]

Les informaticiens se sont intéressés en premier lieu à la formalisation des tâches du développement logiciel, ce qu'ils ont appelé le "**Procédé Logiciel**".

L'utilisation des procédés s'est ensuite étendue à d'autres domaines permettant de les faire évoluer , en introduisant cette nouvelle technique.

La figure 1.2 représente une liste des domaines utilisant les procédés, et les dates du début de l'utilisation de procédé pour chacun d'entre eux.

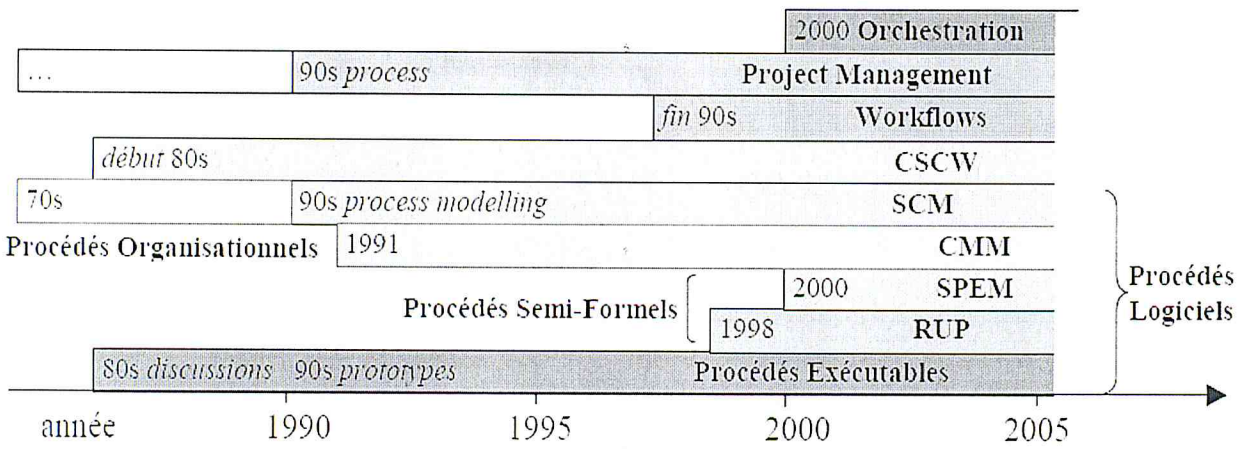


Figure 1.2 : L'histoire des procédés. [1]

Par **procédé** on entend l'ensemble des activités faisant intervenir des équipes de personnes (souvent nombreuses) , des outils et des techniques pour assurer le développement et la maintenance des produits ou des services. [2]

Les **procédés** sont définis comme étant une suite d'étapes réalisées dans un but donné. [1]

L'utilisation de **procédés** est le meilleur moyen pour réaliser un travail de manière bien organisé, que ce soit pour faire du développement logiciel, pour automatiser les tâches à exécuter dans un système logiciel, dans le domaine de la gestion, etc. En effet, le procédé offre de bons moyens pour modéliser et de décrire ce travail.

L'utilisation des **procédés** informatiques a permis de faire évoluer divers domaines dans le monde de l'informatique, ils sont utilisés de manière différente pour chaque domaine : [1]

- Les **procédés logiciels** (Software Process) servent à gérer et à assister le développement logiciel. Il existe plusieurs sous classes de procédés logiciels : les procédés exécutables, les procédés semi formels….etc.
- Dans le domaine **CSCW** (**C**omputer **S**upported **C**ooperative **W**ork), Travail Coopératif Assisté par Ordinateur, les procédés sont utilisés pour coordonner les membres d'un groupe.
- Dans les **workflows**, les procédés servent à automatiser partiellement ou totalement les activités du travail surtout les activités de bureautique et aussi à gérer l'intégration et l'interopérabilité des systèmes d'information HAD (**H**étérogènes **A**utonomes et/ou **D**istribués).
- Dans le domaine de la gestion de projet (**Project Management**), les procédés sont utilisés pour gérer les ressources et les cycles de vie du projet.
- Dans l'orchestration et la chorégraphie de **services web**, les procédés servent à la coordination des services web.

2. Définition des Procédés Logiciels

On peut définir les procédés logiciels comme étant une suite d'étapes réalisées dans un but donné qui servent à gérer et assister le développement logiciel. [1]

On appelle procédé logiciel (Software Process) l'ensemble des activités, des équipes, des outils et des techniques dont le but est d'assurer le développement et la maintenance des systèmes logiciels. [3]

Un procédé logiciel définit les tâches à réaliser, les rôles participants et les produits manipulés pour élaborer ou maintenir un système logiciel. [4]

La qualité des systèmes logiciels dépend de la qualité des procédés par lesquels ils sont créés. [5]

Nous retenons la définition suivante : un procédé logiciel est l'ensemble des activités, des équipes, des outils et des techniques dont le but est d'assurer le développement et la maintenance des systèmes logiciels.

Un procède logiciel peut être représentée à l'aide d'un **Langage de Modélisation de Procédé** (PML). Une représentation du procède de logiciel dans un PML est appelé un **modèle de procède logiciel**. [6]

3. Classification des Procédés Logiciels [1]

Les procédés logiciels servent à gérer et assister le développement logiciel. Il existe plusieurs sous-classes de procédés logiciels :

3.1. Les Procédés Logiciels Exécutables

Le Procédé Logiciel Exécutable définit la manière dont le développement logiciel est organisé, géré, mesuré, assisté, et amélioré (indépendamment du type de support technologique choisi pour le développement).

Il y a eu trois catégories de procédés logiciels exécutables : les Procédés centrés Activités, centrés Produits, et centré Rôles, cela suivant le concept central sur lequel l'accent a été mis.

A la fin des années 90, les entreprises ont commencé à formaliser leur fonctionnement, en mettant en place des workflows. Les workflows sont des procédés exécutables, centré Activités, souvent restreints à des enchaînements simples d'activités.

3.2. Les Procédés Semi-Formels

Les procédés semi-formels sont des procédés logiciels non exécutables, qui ont pour but de décrire la façon dont le logiciel va être développé. Ils recommandent fortement l'utilisation de procédés pour formaliser et modéliser le développement logiciel.

Les procédés semi-formels ont beaucoup apporté au domaine de la modélisation du développement logiciel. Ils ont eu beaucoup de succès en entreprise , surtout RUP, qui présente des meilleures pratiques de développements. Ceci indique le grand intérêt qu'ont ces meilleures pratiques chez les industriels.

3.3. Les Procédés Organisationnels

Les procédés organisationnels définissent les niveaux de maturité, et décrivent les éléments clés d'un procédé logiciel efficace. Ils ont prouvé l'importance de se baser sur un procédé lors du développement logiciel.

CMM (Capability Maturity Model) et CMMI (Capability Maturity Model Integration) considèrent que le développement logiciel ne peut être mature et efficace que s'il est établi sur une infrastructure de procédé fondée sur des pratiques efficaces d'ingénierie logiciel et de gestion. Ils définissent alors les niveaux de maturité, et décrivent les éléments clés d'un procédé logiciel efficace.

CMM et CMMI ont eu un énorme succès auprès des industriels, ils ont réussi à faire entrer le procédés dans l'industrie, et à ancrer dans les esprits l'importance de se baser sur un procédé lors du développement logiciel.

3.4. Les Procédés de Gestion de Configuration

La gestion de configuration est le contrôle de l'évolution des systèmes complexes. Un système complexe est un système ayant :

- Un grand nombre de composants, et de versions de composants (des milliers)
- Un grand nombre de personnes impliquées (des dizaines ou des centaines)
- Des contraintes strictes de temps. (cycles de quelque mois).
- Une longue durée de vie (jusqu'à 10 ou 20 ans).

L'objectif de la gestion de configuration est de contrôler l'évolution du système, et d'aider à satisfaire des contraintes de délais et de qualité.

4. Notion de modèle et méta-modèle

4.1. Présentation de l'OMG

L'OMG (**O**bject **M**anagement **G**roup) est une organisation internationale de standardisation créée en avril 1989. Sa mission est de développer des spécifications standards pour l'industrie des logiciels, qui sont pérennes, fiables, offrant ainsi un cadre commun pour le développement et l'intégration des applications distribuées dans des environnements hétérogènes.

L'OMG produit et distribue gratuitement des spécifications et non des logiciels. Chacun peut réaliser des produits logiciels implémentant ces spécifications. [7]

4.2. Architecture à quatre nivaux de l'OMG

De [1], [7], [8] :

Afin d'organiser et de structurer les modèles, l'OMG (Object Management Group) a défini une architecture appelée : **Architecture à quatre niveaux.** (figure 1.3)

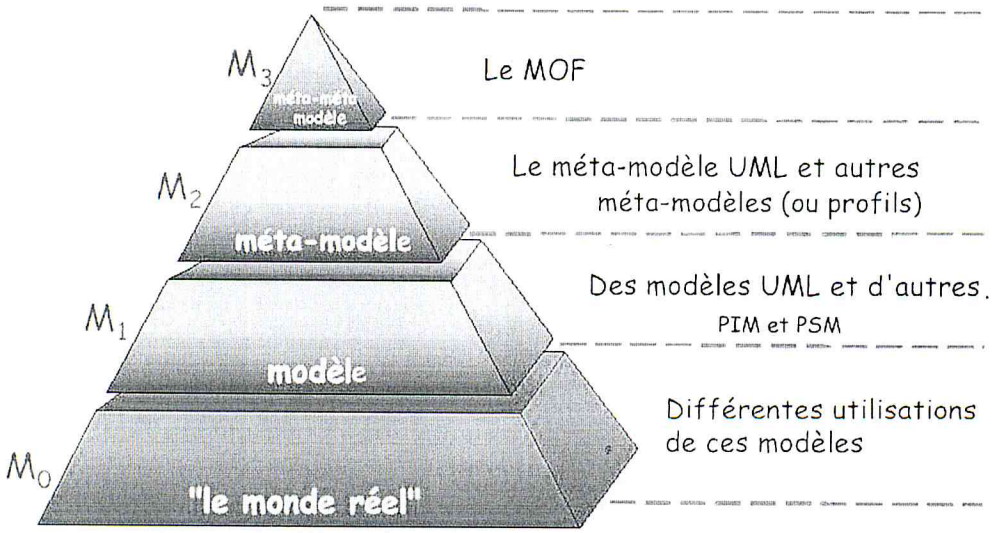


Figure 1.3: Architecture à 4 niveaux de l'OMG

Ces quatre niveaux sont :

4.2.1. Le niveau M0

C'est le niveau des données réelles. Il est composé des informations que l'on souhaite modéliser. Ce niveau est souvent considéré comme étant le monde réel.

4.2.2. Le niveau M1

Lorsque l'on veut décrire les informations appartenant à M0, un modèle appartenant au niveau M1est établi. Un modèle de procédé appartient donc au niveau M1. De même, tout modèle de niveau M1 est exprimé dans un langage dont la définition est fournie explicitement au niveau M2.

4.2.3. Le niveau M2

Le niveau M2 est composé de langages de définition des modèles, appelés aussi méta-modèles. La première définition que l'on peut faire d'un méta-modèle est celle de **modèle de modèles**.

Il existe deux grandes techniques de méta-modélisations, le **MOF** : **M**eta **O**bject **F**acilities (voir niveau M3) et **les profils UML** (voir chapitre 2). Un méta-modèle définit par MOF définit un langage de modélisation spécifique à une famille de modèles. Un profil UML définit une variante de ce langage.

Une multitude de méta-modèles peut être définie, chacun décrivant les structures d'un formalisme donné. Par exemple, le méta-modèle **UML** qui est décrit dans le standard UML définit la structure interne des modèles UML, il représente l'ensemble des concepts UML et les relations qui existent entre eux. Le méta-modèle **SPEM** : **S**oftware **P**rocess **E**ngineering **M**anagement (voir chapitre 2), standardisé également par l'OMG, décrit les concepts nécessaires à la modélisation de processus d'ingénierie logiciels….etc.

4.2.4. Le niveau M3

Ce niveau contient le MOF (Meta-Object Facility), le langage unique de définition des méta-modèles, aussi appelé le méta-méta-modèle. Le MOF définit la structure de tous les méta-modèles qui se trouvent au niveau M2.

4.2.4.1. Présentation du MOF

Le MOF fait partie des standards définis par l'OMG. Il spécifie la structure et la syntaxe de tous les méta- modèles comme UML, SPEM…etc., ils ont le même formalisme. Il définit les concepts de base des méta-modèles : Entité/classe, relation/association, type de données, référence, package.

Il spécifie aussi des mécanismes d'interopérabilité entre ces méta-modèles. Il est donc possible de les comparer et de les relier. Grâce à ces mécanismes d'échanges, le MOF peut faire cohabiter plusieurs méta-modèles différents.

Le MOF décrit pour chaque méta-modèle un modèle abstrait d'objets MOF génériques et leurs associations, un ensemble de règles pour exprimer le méta-modèle à l'aide d'interface **IDL** (**I**nterface **D**efinition **L**anguage), un ensemble de règles sur le cycle de vie et la composition des éléments du méta-modèle.

La figure 1.4 illustre ce qui vient d'être dit :

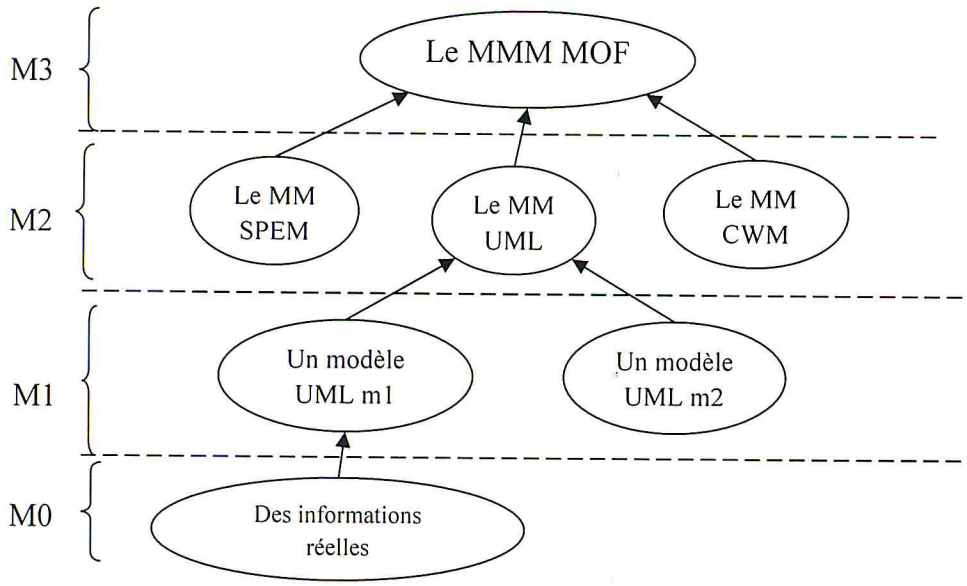
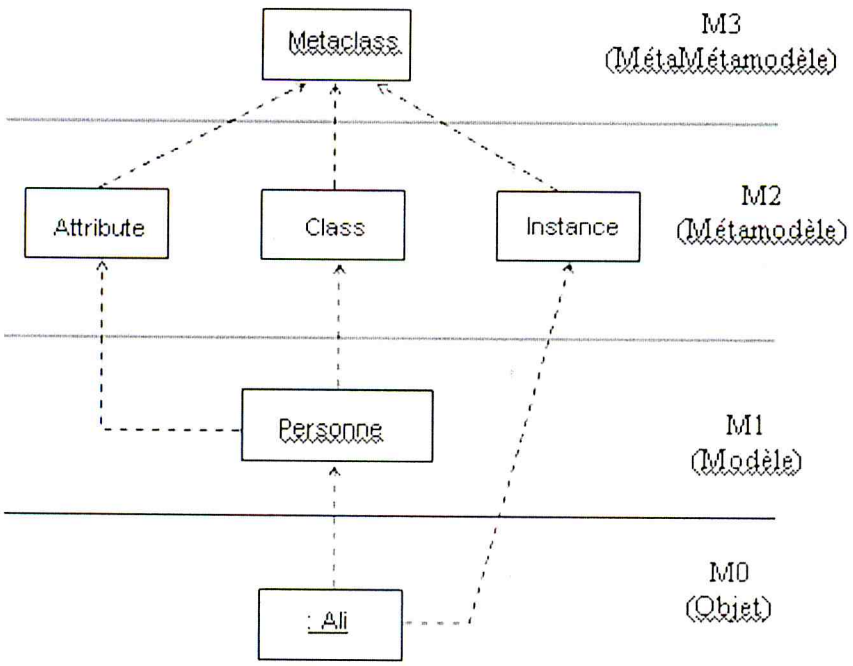


Figure 1.4 : L'hiérarchie à 4 niveaux. [7]

Exemple :



5 . Modèles et méta-modèles de procédés logiciels

Pour pouvoir analyser, améliorer, mesurer et automatiser les procédés logiciels, il faut d'abord les définir explicitement c'est quoi **la modélisation de procédés logiciels.**

L'objectif principal de la modélisation de procédés est d'expliciter les pratiques de développement pour pouvoir les étudier, les améliorer et les utiliser de manière répétitive, gérable et éventuellement automatisable.

5.1. Modèles de procédés logiciels

Un modèle de procédé logiciel est la représentation explicite d'un procédé logiciel dans un langage de description des procédés logiciels. [4]

Un modèle de procédé est la description explicite d'une famille de procédés. La différence entre un procédé et un modèle de procédé est semblable à celle qui existe dans les langages de programmation entre une classe ou un type, et une instance de la classe. [4]

Le modèle de procédé logiciel comprend des activités de développement du logiciel et de maintenance, des activités de gestion de projet et d'assurance qualité, et des activités de méta-procédé. [1]

Un modèle de procédé est une représentation des activités du monde réel. Il (modèle de procédé logiciel) est développé, analysé, raffiné, transformé, et/ou exécuté conformément au méta-modèle du procédé. [1]

La modélisation d'un procédé logiciel est l'élaboration d'une description de ce procédé en utilisant un (ou plusieurs) modèle(s) de procédé. Dans cette définition, un modèle de procédé est une abstraction de procédé représentée par un Langage de Description de Procédés **PML** (**P**rocess **M**odeling **L**anguage). [5]

5.1.1. Langages des modèles des procédés logiciels (PMLs)

Un modèle de procédé doit être représenté dans un certain langage. Un langage de description de procédés est un formalisme de modélisation développé ou adapté pour décrire les procédés. Il définit les concepts dédiés aux procédés, et fournit une syntaxe et un système de notations pour représenter des modèles de procédé en utilisant ces concepts. [4]

Un **PML** est semblable à un langage de programmation dans le sens où il offre une solution de modèle à un problème particulier.

Les PMLs sont centrés humain, ils nécessitent certaines opérations qui doivent être exécutées par des ingénieurs de logiciels, utilisant certains outils logiciels prédéfinis, tandis que les autres opérations sont adaptées pour l'exécution automatique. [9]

Plusieurs langages de modélisation de procédés existent : Appele, PBOOL+,… Etc. Dans notre travail nous utilisons le PML PBOOL+ (voir chapitre 2).

5.1.1.1. Classification des PMLs

Dans [1], les langages de modélisation de Procédé (PMLs) sont classifiés en trois catégories :

1- Les PMLs centrés Produits

Les PMLs centrés Produits se concentrent surtout sur les données échangées, et ont développé d'importantes propriétés concernant ces données (données orientées objet, transactions, persistance, versionnement…).

Nous trouvons dans cette catégorie par exemple, le langage **EPOS**. Il contient un noyau extensible de types d'entités et de types de relations prédéfinies : **tâche**, **donnée**, **outil**, et **rôle** pouvant être spécialisés par héritage. Par exemple, le type tâche permet d'exprimer les pré- et post- conditions des tâches, leur décompositions…. etc.

2- Les PMLs centrés Activités

Les PMLs centrés Activités sont conçus pour fournir un support pour la description d'activités représentant les tâches à réaliser. Comme exemple on peut citer le langage **MSL**, contenant trois concepts principaux : les **classes**, les **règles** (déclarant les pré- et post-conditions et le code des activités), et les **enveloppes d'outils**.

3- Les PMLs centrés Rôles

Les PMLs centrés Rôles sont centrés autour des rôles d'humains, et leurs interactions. Un exemple de ces PMLs est le langage **PWI** (**P**rocess **W**ise **I**ntegrator) : il est basé sur des classes. La hiérarchie de classes contient trois sortes de classes : **Entités**, **Actions**, et **Rôles**.

5.2. Méta-modèle de Procédé Logiciel

Un méta-modèle de procédé logiciel est un modèle définissant les concepts de base d'un langage de description de procédés logiciels. [4]

Les éléments du méta-modèle de procédés logiciels se décomposent en **éléments basiques** et **éléments secondaires**. [1]

Les **éléments basiques** sont les suivants : (figure 1.5)

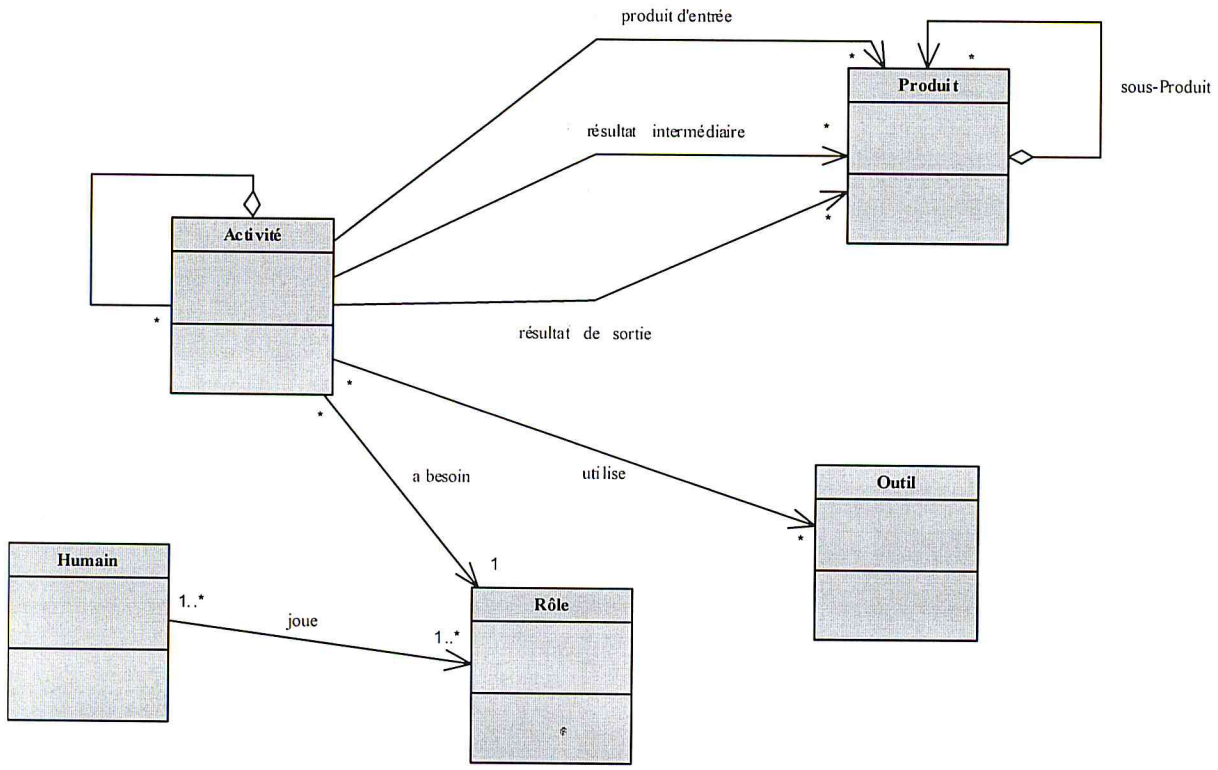


Figure 1.5 : Les éléments basiques de méta-modèle de procédés logiciels. [1]

Une **Activité** est une tâche pendant laquelle des opérations sur le logiciel à développer sont accomplies. Elle est souvent associée à une ou des personnes responsables de cette activité, et à des outils de production. Une activité peut être décomposée en d'autres activités, formant ainsi plusieurs niveaux d'abstraction.

Un **Produit** est souvent un artefact (persistant et versionné), pouvant être simple ou composite, formant les données d'entrée et de sortie des activités. Les produits peuvent être des parties du logiciel à développer, et des documents associés (documents de conception, documentation de l'utilisateur, données de test).

Un **Rôle** décrit les droits et les responsabilités d'un humain. Un humain joue un rôle dans une activité (ou plusieurs rôles dans des activités différentes).

Les **humains** sont des agents (ou des développeurs) du modèle de procédé logiciel, que l'on peut organiser dans des groupes. Un rôle est attribué aux humains ayant les compétences et les responsabilités nécessaires pour jouer ce rôle. Un humain peut avoir plusieurs rôles, il peut également être membre de plusieurs groupes (ces groupes peuvent aussi être imbriqués).

Les **outils** sont des systèmes qui assistent la production de logiciel. Il existe deux sortes d'outils : les outils interactifs (éditeurs textuels, outils graphiques), et les outils simplement exécutables sans interaction (compilateurs, analyseur grammatical…).

Le support d'évolution aide à gérer l'évolution du procédé logiciel (sa modification) à travers les Directions (politiques, règles, et procédures).

Les **éléments secondaires** sont les suivants : Projet/organisation, Espace de travail, Vue utilisateur, Modèle de coopération, Modèle de versionnement/transaction, et Modèle de qualité/performance.