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 du diplôme de Master en informatique

Spécialité : Ingénierie des logiciels

Sujet :

Réalisation et Intégration d'applications pour la réutilisation de procédés logiciels à base d'architectures logicielles.

Présenté par :
ALLICHE Mohamed.

Promotrice :
Mme F.AOUSSAT.

Soutenu le : 20 septembre 2012

Devant le jury composé de :

Présidente : Mme OUAHRANI

Examinateur : Mr FERFERA

Examinateur : Mr ZAIR

2011 – 2012

« Qui apprendre veut bien, n'est arrêté par rien. »

Proverbe allemand

Dédicaces

Avec une immense joie, je dédie ce travail à ceux que j'aime et qui sont chers à mon cœur :

A mes chers parents qui m'ont toujours éclairé mon chemin et que dieu me les garde :

A ma chère mère pour sa gentillesse, son affection, sa douceur, sa tendresse, ses encouragements et sans elle rien n'aurais été possible.

A mon cher père pour son encouragement, sa patience, son aide continuel le long de mon chemin d'étude et son soutien moral et financier.

A mes chères sœurs : à qui je leurs souhaite pleins de succès dans leur vie.

A mes amis et camarade de la promotion 2010-2011 que dieu les garde.

Enfin, je souhaite tout particulièrement adresser mes chaleureux encouragements à toutes celles et à tous ceux qui m'ont aidé de près ou de loin dans ma réussite.

Remerciements

Je rends grâce à Dieu de m'avoir donné la force et le courage de terminer mes études

Je voudrais tous d'abord exprimé mes sincères remercîmenis à l'ensemble des membres du jury d'avoir accepter d'évaluer mon travail malgré leurs emplois du temps souvent chargés.

Je voudrais exprimer ma gratitude et ma profonde reconnaissance à Mme AOUSSAT, ma Promoirice, je la remercie pour ses orientations et ses commentaires utiles qui m'ont permis de surmonter mes difficultés et de progresser dans mes études.

J'exprime ma profonde reconnaissance à mes parents, je les remercie pour leur soutien, leur encouragemenis tous le long de mon cursus universitaire.

Et enfin, je voudrais adresser mes remercîments à tous ceux qui ont contribué de prés ou de loin à la réalisation de mon projet de fin d'étude.

J'espère que ce mémoire servira de base de travail et support de recherche dans les années à venir.

وما توفيقي إلاّ بالله

ALLICHE Mohamed

Sommaire

Chapitre III : Conception et Implémentation

Chapitre V : Tests et validation

Conclusion générale

Annexe

Glossaire

1. Contexte du travail :

Un modèle de Procédé Logiciel (PL) est la description des enchainements d'activités, des outils, ressources et rôles utilisés pour la réalisation puis la maintenance d'un produit logiciels. 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. 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.

Aussi, la modélisation des PLs n'échappe pas aux contraintes de développement auxquels sont soumis les logiciels ; modéliser des PLs de qualité dans des délais et à des prix compétitifs reste une priorité. Réutiliser les connaissances acquises par les précédentes expériences de modélisation et d'exécution de PLs éprouvés **[37] [39]**; prendre en comptes les réflexions et les pratiques testées et adoptées par les modélisations précédentes est une des solutions explorée dans le domaine de l'ingénierie des PLs. En effet, le domaine des procédés logiciels a donné naissance à un nombre important de concepts, paradigmes et langages de modélisation afin de fournir des PLs appropriés aux contraintes spécifiques de chaque type de développement.

Par ailleurs, la plupart des modèles de PLs sont connues pour être rigides difficilement manipulables. Afin de modéliser des PLs flexibles, adaptables, la modélisation des procédés logiciels à base d'architectures logicielles est une des approches exploitées. L'intérêt est non seulement d'augmenter la réutilisabilité des modèles de procédés logiciels (MPL) mais aussi de bénéficier des avancées considérables du domaine des architectures logicielles et de le mettre au service de l'ingénierie de procédés logiciels.

En se basant sur le constat fait sur la richesse du domaine en termes de concepts et d'expériences, une approche de modélisation de PLs à base d'architectures logicielles a été proposée. Cette approche a pour objectif d'élargir et de faciliter la réutilisation pertinente de l'expertise capitalisée à partir des modèles de PLs existants. L'objectif de l'approche est de décrire des architectures de procédés logiciels. L'architecture de PL permettra par la suite la génération automatique du modèle de Procédé logiciels. Cette génération est possible grâce à l'exploitation d'une ontologie de domaine qui regroupe les connaissances procédé logiciels réutilisables.

2. Les objectifs :

L'objectif de notre travail est la réalisation d'une première version exécutable de l'approche cité précédemment.

La réalisation du prototype implémentant cette approche est assez difficile, en effet, l'approche, explore plusieurs domaines de recherche pour la modélisation des PLs (les architectures logicielles, les ontologies). Pour cela, la réalisation du prototype a été divisée en plusieurs projets de fin d'études, ces projets ont été tous réalisés par les étudiants d'université Saad Dahlab Blida, et cela sur plusieurs années universitaires.

L'objectif de notre projet et d'intégrer les résultats des ces projets en un seul prototype cohérent fonctionnant correctement.

Même si le travail parait facile à réaliser, cette phase constitue la phase la plus importante et même la plus difficile de la réalisation du prototype. Le travail est rendu encore plus difficile pour les raisons suivantes :

- La nécessite de comprendre la solution proposée dans sa globalité et en détail contrairement aux projets précédents qui se focalisait sur une partie de l'approche.
- La nécessite de comprendre chaque solution conceptuelle proposée pour chaque projets de fin d'études.
- La nécessite de comprendre chaque solution de programmation adoptés pour chaque projets de fin d'études.
- Plusieurs versions de l'ontologie ont été utilisées, ce qui créé des problèmes de compatibilité entre les projets.
- Proposer une solution optimale pour regrouper tous les travaux, en minimisant la reprogrammation des applications déjà réalisés.

3. Organisation du mémoire :

Pour présenter notre travail nous organisons notre mémoire comme suit :
Le chapitre I : présente les concepts de base des domaines traités : à savoir les procédés logiciels, et les architectures de procédés logicielles.

Le chapitre II: présente l'approche de modélisation de PLs à base d'architectures logicielles, nous présentons aussi le résumé des projets réalisés, leurs avantages et leurs inconvénients pour l'intégration.

Le chapitre III : présente la conception de notre prototype ainsi que son implémentation.
Le chapitre V : présente les testes de validation.

Et nous terminons notre mémoire par une conclusion qui synthétisera nos principales contributions et donnera quelques perspectives à notre travail.

Introduction :

Le sujet traité dans ce chapitre relève de l'ingénierie des procédés logiciels (PLs). Nous nous intéressons principalement à la réutilisation des PLs à base d'architectures logicielles. Par conséquent l'introduction des concepts de base du domaine des procédés logiciels et des architectures de procédés logicielles s'avère indispensable. Aussi, dans la solution que nous devons implémenter une ontologie de domaine est exploitée par conséquent, nous introduisons aussi les concepts de base du domaine des ontologies.

1- MODELISATION ET META MODELISATION DES PLs :

Afin d'organiser et de structurer ses modèles, l'OMG (Object Management Group) a défini une architecture à plusieurs niveaux de modélisation appelée : Architecture multi niveaux des modèles (Figure 1). Cette architecture fournit un support d'abstraction basé sur les modèles qui permet de maitriser la complexité croissante des logiciels.

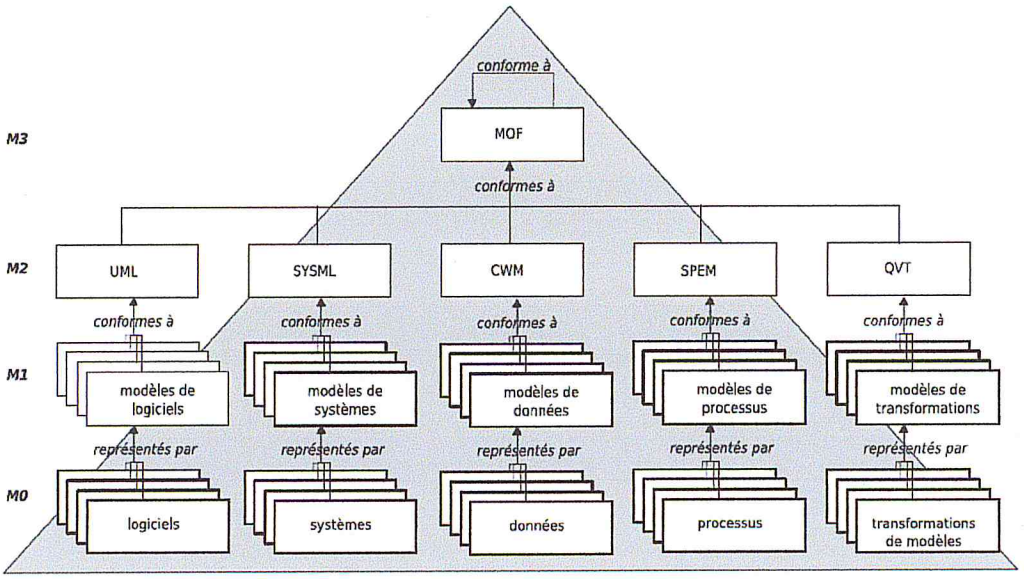

Figure 1 : l'architecture à 4 niveaux de l'OMG [3]

Les PLs étant régis par les modèles, ils respectent eux aussi cette architecture, ainsi, la modélisation des PLs est organisée comme suit (Figure 2) :

–Le niveau M0 décrit les procédés logiciels.

–Le niveau M1 décrit les modèles de procédés logiciels.

–Le niveau M2 décrit les métas modèles de procédés logiciels.

–Le niveau M3 décrit les métas métas modèles de Procédés logiciels.

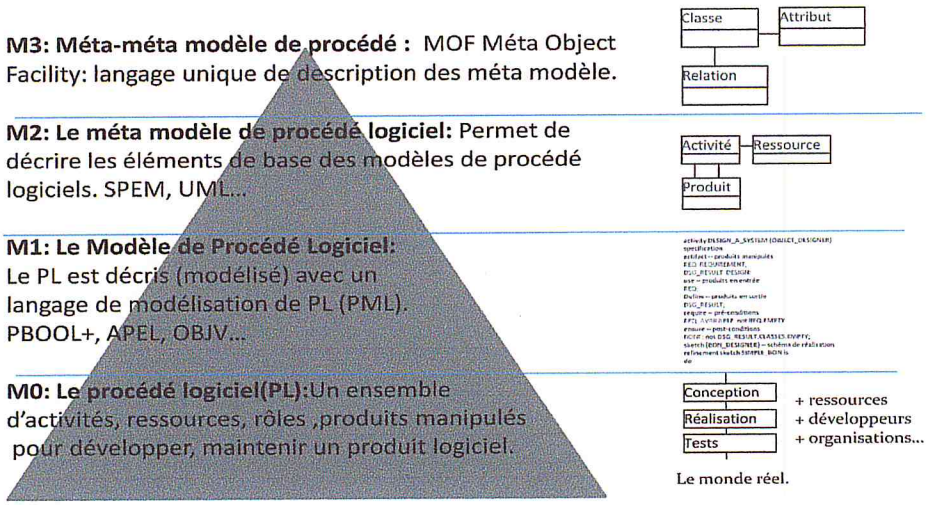

Figure 2 : Modélisation des procédés logiciels selon l'architecture à 4 niveaux de l'OMG.

1.1 Le niveau M0 : Les Procédés Logiciels :

Un procédé est défini comme suite d'opération ou d'événements faisant intervenir des équipes de personnes, des outils et des techniques pour assurer le développement et la maintenance de produits ou de services. Un procédé logiciel est, par conséquent, un procédé qui permet d'assurer le développement et la maintenance de produits logiciels.

Dans la littérature plusieurs définitions de procédés logiciels ont été avancées, nous citons celles que nous jugeons assez significatives :

"... The set of partially ordered process steps, with sets of related artifacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables [5].

"A software process can be defined as the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product" **[2]**.

"...A software process is defined as a sequence of steps that must be carried out by the human agents to pursue the goals of software engineering..." [6].

"... software process means the set of activities required to produce a software system, executed by a group of people organized according to a given organizational structure and counting on the support of techno-conceptual tools..." [7].

"... Software process is a partially ordered set of activities undertaken to manage, develop and maintain software systems..." [7].

« ...procédés logiciels comme étant une suite d'étapes réalisées dans un but donné qui servent à gérer et assister le développement logiciel... » [8].

Tous les définitions présentées s'accordent à définir le PL comme un enchainement d'unité de travail à suivre, de l'ensemble des produits logiciels requis et produits par ces unités de travail ; des équipes, outils, techniques et stratégies utilisées dont le but d'assurer le développement et la maintenance de produits logiciels. Concrètement, le PL représente le monde réel du développement (Figure 3).

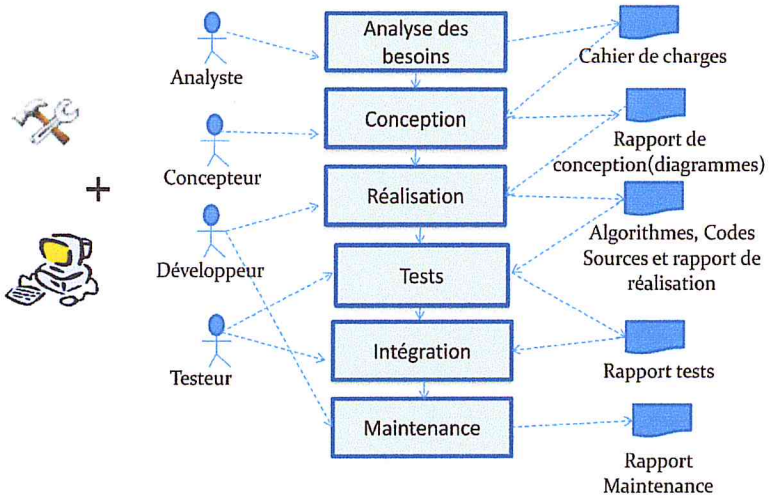

Figure 3 : Procédé logiciel (le monde réel du développement)

1.2 Le niveau M1 : Les Modèles de Procédés Logiciels :

Un modèle est une abstraction, une simplification d'un système qui est suffisante pour comprendre le système modélisé [9].

"a Process Model (PM) is an abstract description of an actual or proposed process that represents selected process elements that are considered important to the purpose of the model and can be enacted by a human or machine" [15].

Les travaux sur les PLs s'accordent, pour la plupart, sur la définition du modèle de PLs. Un modèle de PL est la représentation formalisée du PL, il explicite les propriétés et les variables régissant le monde réel du développement.

L'objectif principal du Modèle de PL est de fournir une représentation plus au moins formelle pour le développement logiciel. Il doit pendre en charge, entre autres, l'évolution prévue et imprévue de la réalité du développement.

Différents modèles de PL peuvent décrire différents angles de développement logiciels [7]. En effet, par soucis de clarté et de précision, les Modèles de PLs peuvent se focaliser sur un aspect particulier du développement et reléguer d'autres détails au second plan. D'autres Part, les différents langages de modélisation de PL, les différentes exécutions possibles augmentent le nombre de modèles de PLs existants, par conséquent, Plusieurs classifications de PLs ont été définies, ces classifications dépendent :

- Niveau de formalisation du modèle de PL : non formel, semi formel, exécutable.
- De l'élément central du modèle de PL : centré activité, centré rôle, centré artefact… [14]
- De l'orientation du Modèle de PL : orienté gestion de configuration, orienté décision, orienté stratégie… [14]
- Du type du Langage de Modélisation de procédé (PML) utilisé : orienté objet, réseau de pétri, à base de règles… [7][10][11]
- Du type d'exécution supporté : exécution distribuée, simulation… [7] [10] [11].

1.3 Le niveau M2 : Les métas modèles de Procédés Logiciels :

Le méta modèle de PL est un modèle regroupant les concepts de base du langage d'expression d'un modèle de PL. Le méta modèle de PL doit prendre en considération les points de vues et les orientations des PLs, en effet, les concepts représentés varient selon l'orientation du PL, par conséquent, il existe autant de méta modèles que d'orientations de PLs [14]. Cependant tous les PLs sont conformes au même noyau conceptuel (Figure 4) : Le PL est un enchainement d'unités de travail. Chaque unité de travail " Work Unit" a besoin de produits "Work Products" en entrée « inputs » pour fournir des produits en sortie « outputs ». Le PL étant centré humain, Une unité de travail est sous la responsabilité d'un rôle "Role".

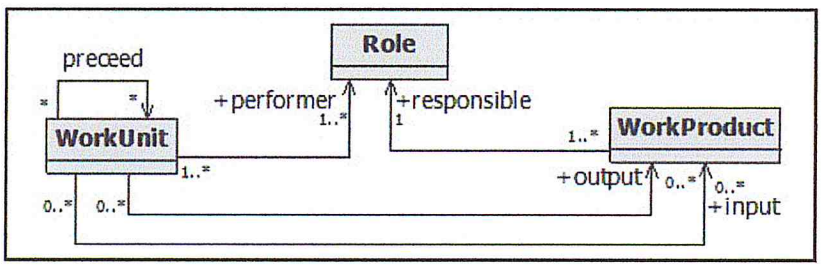

Figure 4 : Méta modèle noyau du procédé logiciel

Ainsi, les concepts de base sont :

Ainsi, les concepts de base sont :

- **Work Unit (Activité):** représente les unités de travail effectuées. Les enchainements, état d'avancements exploités par les activités peuvent être décris aussi.
- **Work Product (produit):** représente les produits manipulés. Les transmissions, le format, les versions et le stockage de ces produits peuvent être traités aussi.
- **Rôle :** Décrit en général les responsabilités et qualifications requises ou existantes que peut ou doit avoir un acteur, lors du développement logiciel.

Il est clair que d'autres concepts tel que outils, organisation, stratégie… peuvent être présents dans un méta modèle de PL, cela dépendra de l'orientation et de la spécialisation des modèles de PL que l'on veut décrire.

Dans la section suivante, Nous nous intéressons au méta modèle SPEM 2.0 (System and Software Process Engineering Meta model) définit par l'OMG comme standard pour la modélisation des PLs [16], nous présentons ce méta model car il sera exploité dans notre solution de réutilisation de PLs.

1.4 Le niveau M3 : Les métas métas modèles de Procédés Logiciels :

Le méta méta modèle est une représentation qui permet de décrire les concepts, structures des métas modèles. L'OMG spécifie un méta méta modèle unique et auto descriptif qui est le MOF (Model Object Facility) **[23]**. Le MOF décrit les concepts de base de tous les métas modèles de tous les domaines y compris celui des PLs.

2- LE MÉTAMODÈLE SPEM (SYSTEMS AND SOFTWARE PROCESS ENGINEERING METAMODEL) :

SPEM (**Systems and Software Process Engineering Meta model**) est un méta modèle qui permet de décrire les concepts et principes de base pour la modélisation des PLs. L'objectif de SPEM n'est pas de devenir un langage générique de modélisation de PLs mais de couvrir la description des concepts d'un large éventail de PLs, sans se spécialiser dans un type particulier, tout en prenant en compte la diversité reconnue des PLs.

C'est un standard adopté par l'OMG, il se base sur la notation UML (profiles et modèles) pour décrire les concepts PL sous différents angles.

2.1 Noyau Conceptuel de SPEM :

Le noyau conceptuel de SPEM est celui de tous PL, ainsi, les concepts de base de SPEM sont les mêmes que ceux des PLs : Unité de Travail, Produit de travail, rôle et les relations entre ces concepts.

Pour représenter ces concepts et les relations entre ces concepts SPEM a défini 4 types de classes :

- Classes représentant la notion **d'unité de travail.**
- Classes représentant **le réalisateur de l'unité de travail**.
- Classes représentant **les entrés et sorties de l'unité de travail**.
- Classes représentant la notion **d'élément ;** ces classes permettent de décrire les concepts de base d'un PL tels que Role, Produit ainsi que d'autres concepts nécessaires à la modélisation des PLs.

Ces 4 types de classes sont bien agencées et sont présentes à plusieurs niveaux de SPEM, offrant une vision particulière des ces concepts.

2.2 Les métas modèles de SPEM :

Pour couvrir les besoins de modélisation et d'exécution d'un large éventail de PLs, SPEM 2.0 est structuré en **7 packages qui représentent 7 méta modèles**, chaque méta modèle décrit une vision particulière du PL (Figure 5).

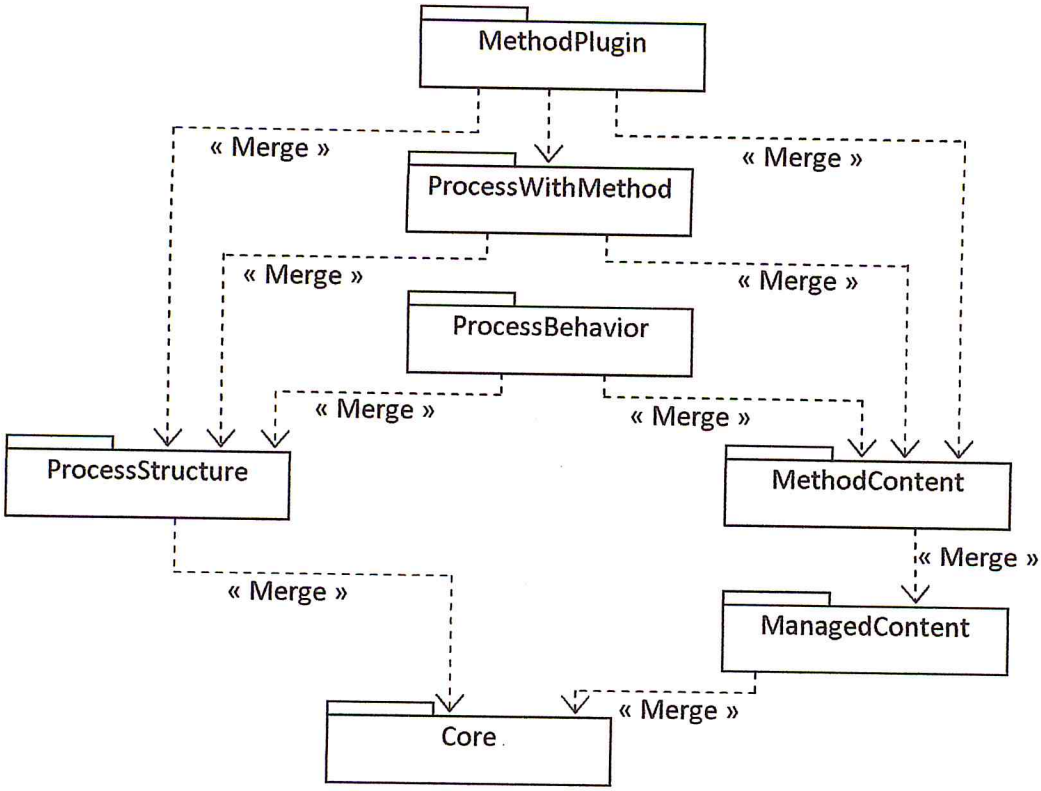

Figure 5 : Packages de SPEM

SPEM est organisé en plusieurs métas modèles afin, d'une part, de faciliter la compréhension et l'extension de SPEM, et d'autres part, d'augmenter la réutilisation des concepts décris par SPEM que se soit structure des PLs, connaissances des méthodes de développement ou documentation et assistance.

Ainsi, le méta modèle « CORE » est dédié à la description des concepts de base de SPEM, il fournit la base conceptuelle pour l'extension des autres métas modèles.

Le méta modèle « MANAGED CONTENT » fournit les concepts nécessaires pour la description de la documentation et l'assistance qui peut être exploité par des modèles de PLs, il permet aussi de décrire des PLs non formels qui exploitent le langage naturel.

Le méta modèle « PROCESS STRUCTURE » contient les éléments de base pour la structuration d'un PL. Il décrit des fragments de PL ou des enchainements d'activités indépendamment de toute méthode de développement. Ce méta modèle permet de décrire des modèles de PLs Ad hoc qui n'ont pas de méthodes ou de cycle de vies particuliers, il permet aussi de décrire des patrons de PLs et des enchainements d'activités récurrents et réutilisables. Contrairement, au méta modèle « PROCESS STRUCTURE », le méta modèle « METHOD CONTENT » fournit les concepts de base pour définir les méthodes, techniques, meilleures

pratiques de développement réutilisables indépendamment d'un projet de développement ou d'une structure de PL particulière.

Comme son nom l'indique, Le méta modèle « PROCESS WITH STRUCTURE » permet de décrire un PL structuré qui respecte les concepts du méta modèle PROCESS STRUCTURE, et en même temps, qui suit une méthode de développement définie dans le méta modèle « METHOD CONTENT », plaçant le PL dans un contexte particulier et en respectant une méthode ou cycle de vie particulier.

L'objectif du méta modèle « METHOD PLUGIN » est de fournir les outils pour utiliser et réutiliser les connaissances des métas modèles « METHOD CONTENT », « PROCESS STRUCTURE » et « PROCESS WITH METHOD ».

L'utilisation passe par la définition de configurations (Method Configurations) et de correspondance entre éléments « METHOD CONTENT » et éléments « PROCESS STRUCTURE » (Method Plugins).

Par contre, la réutilisation des PLs est matérialisée par l'introduction de la réutilisation à base de composants procédés. Ainsi, des concepts architecturaux tel que : composant (Process Component), port (Work Product Port) et connecteur (Work Product Port Connectors) sont introduit, cependant la réutilisation des PLs n'est qu'à ses débuts et plusieurs problèmes de réutilisation ont été relevés [16].

La réutilisation des Modèles de PLs et des comportements externes et qui ne sont pas conforme à SPEM sont pris en charge par le package « PROCESS BEHAVIOR ».

3- LES ARCHITECTURES DE PROCEDES LOGICIELS :

3.1 Définition :

Dans ce mémoire, notre travail se limite à la manipulation d'un type particulier de PLs : les architectures de procédés logiciels, par conséquent il est important de définir ce type de procédé logiciels. Les concepts de base des architectures de procédés logiciels s'inspirent des concepts architecturaux connus dans le domaine des architectures logicielles. Ainsi, l'architecture de procédé logiciel est une architecture logicielle qui a comme éléments architecturaux de base : le composant, le connecteur et la configuration.

3.2 Eléments de base de l'architecture de procédé logiciel :

La distinction entre Activés de "création" de produit (qui constitue les composants PL) et Activités "d'adaptation et de contrôle" de flux (qui constitue les connecteurs PL) est la caractéristique principale de ce type de procédé logiciel. Ainsi, cette distinction impose une

sémantique spécifique par rapport aux solutions existantes. L'interprétation des concepts architecturaux des architectures de PLs est comme suit **[19]**:

– **Composant PL :** Décrit un traitement réalisé sur des produits en entrée pour "la création" de nouveaux produits en sortie.

– **Port PL (Interface du composant) :** L'interface d'un composant est un ensemble de points d'interactions du composant PL ; elle spécifie les services fournis et requis nécessaires de l'exécution du composant PL. L'interface du composant PL est un ensemble de "Ports PLs". Les ports requis correspondent aux "flux en entrée" nécessaires à l'exécution du composant procédé, Les ports fournis correspondent aux "flux en sortie". Deux types de ports PL sont définis :

– Ports flux de données (Data Flow Ports) : sont des points d'interactions spécifiques aux produits des PLs, Ils permettent le transfert des produits logiciels du PL.

– Ports flux de contrôle (Control flow Ports) : sont des points spécifiques aux flux d'exécution des PLs, Ils permettent d'identifier l'ordre et l'état d'exécution du PL.

– **Connecteur PL :** Décrit un traitement réalisé sur des produits en entrée afin de les adapter ou les évaluer pour les besoins du composant PL suivant.

– **Rôles connecteur PL** (Interface connecteur) : L'interface de connecteur PL est représentée par de "Rôles Connecteur PL". De la même manière que les ports PL, ils représentent les points d'interaction pour la transmission des données (produit ou flux d'exécution) requis ou fournis par les connecteurs PL. comme pour les ports PLs Deux types de " Rôle Connecteur PL" sont définis : les "Data Flow" connecteur Rôle et les "Control flow" connecteur rôle.

– **Binding :** Un composant et une configuration peuvent être composées hiérarchiquement d'un ou plusieurs sous-composants. Les connexions entre les ports PLs d'une configuration ou d'un composant composite et les ports PL de leurs sous composants sont appelées des bindings. Le binding permet de formaliser la délégation du flux de donnée (respectivement flux d'exécution) aux éléments internes du composant composite, du connecteur composite ou de la configuration.

De la même manière, les connecteurs PLs peuvent être composé de sous connecteurs. Le binding décrit aussi la connexion entre le rôle d'un connecteur PL composite et le rôle de son sous connecteur PL. Les Binding sont possibles qu'entre des ports PL de même type, ou entre des rôles connecteur de même type (Data flow ou Control flow).

– **Attachement :** Est le lien entre le Port PL d'un composant PL et un Rôle Connecteur PL d'un connecteur PL. l'attachement se fait entre le port PL et rôle connecteur PL de même type (Data Flow ou Contrôle Flow). L'attachement permet de formaliser soit la transmission de données ou bien l'ordre d'exécution des composants PL et connecteurs PL.

– **Configuration PL :** Elle décrit l'ensemble logique des composants PLs et des connecteurs PL en déterminant explicitement les contraintes d'assemblage de la structure PL. Une configuration peut respecter un ou plusieurs styles PL ou pas.

– **Style PL :** Fournit une description partielle de la logique d'assemblage des structures prédéfinies et récurrentes dans les PLs. Le style PLs est introduit formellement, nous définissons les concepts types, invariants et des contraintes topologiques de manière explicite.

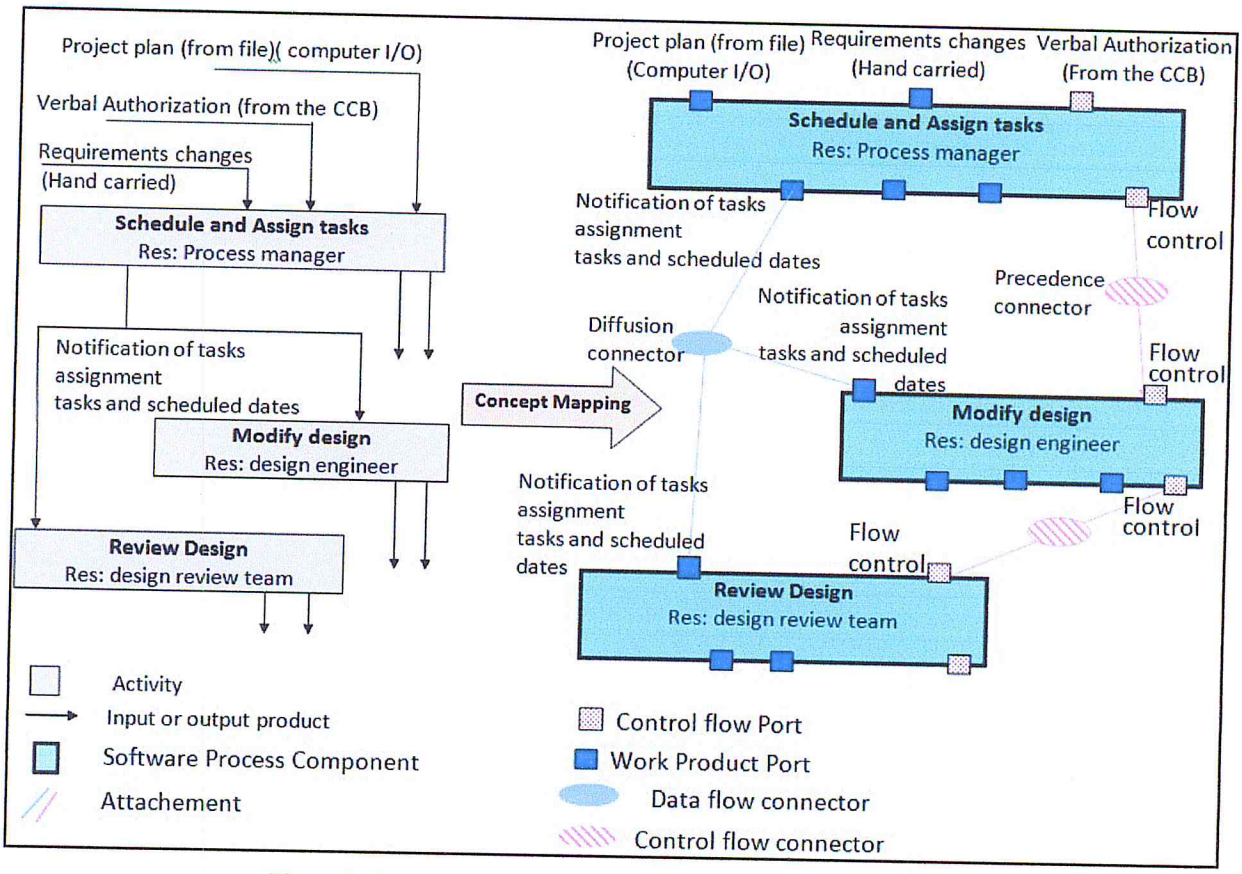

Figure 6 : les concepts de base de l'architecture de procédé logiciel

4- Les ontologies :

4.1 Définition :

En 1993, GRUBER formule la définition suivante qui deviendra par la suite la définition la plus célèbre et la plus utilisée : « une ontologie est une spécification explicite d'une conceptualisation» **[17]**. Dans cette définition :

- **Conceptualisation** correspond à un modèle abstrait à base de concepts et relation entre concepts des phénomènes du monde.
- **Explicite** signifie que le modèle en question doit être décrit de façon non ambiguë dans un langage formel pour pouvoir être manipulé par un agent logiciel ou par un agent humain.

Le terme ontologie est d'origine grecque, il a vu le jour dans le domaine de la philosophie, où il signifie : explication systématique de l'existence, l'étude de ce qui existe dans le monde **[18]**. Par la suite le terme a été utilisé en informatique et en science de l'information où une ontologie est une représentation des connaissances du monde au niveau conceptuel qui par la

suite sera employée pour permettre le raisonnement automatique sur les objets du domaine concerné.

Les ontologies ont pour but de comprendre la connaissance dans un domaine, d'une façon générale et de fournir une représentation communément acceptée qui pourra être réutilisée et partagée par diverses applications **[18].**

- **Réutilisable :** car cette représentation est faite de façon déclarative c'est-à-dire sans lien avec la manière dont ces connaissances vont être utilisées.
- **Partageable :** car cette représentation est établie sur la base des concepts qui caractérise un domaine ainsi que sur des concepts fondamentaux communs qui sont utilisable à travers divers domaines, ce qui permet la communication entre les systèmes d'informations qui doivent partager des informations basées sur des concepts communs **[19].**

4.2 Constituants d'une ontologie

Une ontologie constitue un modèle de données représentatif d'un ensemble de concepts dans un domaine donnée, réel ou imaginaire ainsi que les relations entre ces derniers. Les concepts sont organisés dans un graphe dont les relations peuvent être sémantiques ou de subsomption.

4.2.1 Concepts :

Un concept peut représenter un objet, une notion ou une idée. Il peut se définir comme une entité composée de trois éléments :

1. Un terme (ou plusieurs) exprimant le concept en langue.
2. Une notion également appelée intension, contenant la sémantique du concept, exprimée en terme de propriétés et d'attributs, de règles et de contraintes. Les règles qui décrivent des inférences possibles sont des affirmations sous la forme : antécédent -> conséquent.
3. Un ensemble d'objets appelé aussi extension du concept, regroupant les objets manipulés à travers le concept : ces objet sont appelés instances du concept.

Exemple : Soit le concept Voiture :

- **Terme :** Voiture.
- **Intension :** Véhicule de transport automobile conçu et aménagé pour le transport d'un petit nombre de personnes.
- **Règle :** Une voiture rare est chère.

- **Extension :** Liste de toutes les voitures du monde.

Deux extensions peuvent ne pas être disjointes alors que deux intensions s'excluent mutuellement par au moins une propriété.
Deux concepts partageant la même extension sans avoir la même intension, peuvent être désignés par un même terme ; ceci correspond à des points de vue différents sur un même objet.

Exemple : Les chiens peuvent être considérés comme des animaux de compagnie ou comme des sources culinaires.

Les concepts sont désignés par les nombreux termes que contient le langage naturel. Il se pourrait qu'un terme désigne plusieurs concepts différents d'où la présence d'une ambiguïté.

Exemple : Le terme table pour un meuble et table pour tableau de valeurs numériques.
La machine, où on identifie généralement un concept à partir de ses termes, ne gère pas ces ambiguïtés. Mais la restriction à un domaine de connaissances permet d'éviter la homonymie de concepts c'est-à-dire des concepts différents désignés par un même terme, par contre afin d'assurer une grande souplesse d'utilisation de l'ontologie, il serait souhaitable de permettre la désignation d'un concept par plusieurs termes.

Un concept peut être défini à partir d'autres concepts.
Exemple : Le concept de table peut se définir en utilisant le concept meuble, plateau et pieds.

4.2.2 Relations [17] :

Certain liens conceptuels existants entre les concepts peuvent être exprimés par les propriétés portant sur les concepts, d'autres doivent être représentés à l'aide de relations autonomes.
Comme pour les concepts, les relations peuvent être spécifiées par des propriétés, elles sont organisées de manière hiérarchisée à l'aide de la propriété de subsomption.

- **Relation de subsomption**

Les concepts dans un domaine de connaissances sont manipulés au sein d'un réseau de concepts. Ces derniers sont structurés hiérarchiquement et sont liés par des propriétés conceptuelles. La propriété utilisée pour structurer la hiérarchie des concepts est la subsomption qui lie 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 de C2 est forcément plus réduite que celle de C1, par contre son intension est plus riche.

- **Liens conceptuels**

Une relation permet de lier des instances de concepts ou des concepts génériques. Elle est caractérisée par :

- Un terme (ou plusieurs).
- Une signature qui précise le nombre d'instances de concepts que la relation lie, leurs types et l'ordre des concepts, c'est à dire la façon dont la relation doit être lue.

4.2.3 Fonctions :

Les fonctions sont des cas particuliers de relations dans lesquelles le nième élément de la relation est défini à partir des n-1 premiers. **[18]**

Exemple : La fonction mère-de ou carré-de.

4.2.4 Axiomes

De **[18] et [20]** :

Les axiomes sont utiles à la formulation de phrases qui sont toujours vraies. Ils permettent de :

- Contraindre les valeurs de classes ou d'instances.
- Représenter les intensions des concepts et des relations dans un domaine.
- Spécifier la façon dont les concepts d'un domaine et les relations qui les lient peuvent être utilisés afin d'exprimer des connaissances dans le domaine.

Certains axiomes sont particuliers, ils se retrouvent dans plusieurs ontologies.

Exemple : Soit l'axiome : La relation parent de est l'inverse de la relation enfant de.

Certains sont propres à un domaine, ils participent à la définition de la sémantique du domaine.

Exemple : Dans le domaine de la géométrie, soit l'axiome : Il n'existe pas plus d'une droite à laquelle appartiennent deux points A et B.

4.2.5 Instances :

Les instances sont utilisées pour représenter des éléments dans un domaine. [18]

5- Conclusion :

Dans ce chapitre nous avons définit les concepts de base des domaines qui vont être exploités dans notre mémoire. Nous avons définit les notions de base des procédés logiciels et nous avons détaillé la notions d'architecture de procédé Logiciel. A la fin de ce chapitre nous avons introduis les concepts de base concernant les ontologies et leur structure.

Dans le chapitre suivant nous présentons l'approche que nous allons implémenter, comme cité dans l'introduction générale, notre travail consiste à intégrer des applications qui existent déjà, par conséquent nous présentons aussi les applications qui ont été réalisé leur point fort et points faibles et difficultés de leur intégration.

Introduction :

Dans ce chapitre, nous présentons l'approche étudiée, son principe, objectif et fonctionnement. Nous présentons aussi, les projets réalisés pour la mise en place de cette approche et qui doivent être intégrés pour avoir le prototype global.

1- L'approche de modélisation de Procèdes Logiciels à base d'architectures logicielles :

L'approche de modélisation de PLs à base d'architectures logicielles étudiée est bien originale du fait qu'elle exploite toutes les opportunités pour pousser la réutilisation des PLs à son extrême, et cela, en mettant deux domaines de recherche prônant la réutilisation à large échelle (les architectures logicielles et les ontologies) au service de la réutilisation des PLs.

Outre la réutilisation large et pertinente des modèles de PLs existants, l'apport principal de cette approche réside dans singularité de la modélisation des PLs : le modèle de PL est traité comme une architecture logicielle, ainsi, comme pour une architecture logicielle, une configuration de PL est un assemblage de composant PL qui interagissent à travers des connecteurs PLs.

Par conséquent, modéliser un procédé logiciel reviens à déployer une architecture de PL, le contenu du PL est modélisé indépendamment de sa structure, et la structure logique indépendamment de l'implémentation.

L'approche couvre les deux ingénieries de réutilisation **[32] [21]**:

- « Pour » la réutilisation en offrant une ontologie qui capitalise les expériences positives des modélisations précédentes des procédés logiciels et en permettant l'inférence de nouvelle connaissance PL.
- « Par » la réutilisation en permettant la description d'architectures de PLs puis le déploiement de la configuration PL.

1.1 - Ingénierie pour la réutilisation des PLs :

Pour capitaliser les connaissances du domaine, la solution adoptée est l'utilisation d'une ontologie de domaine regroupant les concepts du domaine des PLs. L'ontologie constitue alors un socle qui permet de contenir les connaissances de ce domaine, et permettra, ainsi, leur réutilisation indépendamment de leur environnement d'origine.

L'avantage de cette solution est de pouvoir inférer de nouvelles connaissances à partir de connaissances capitalisée à partir de sources différentes. Aussi, elle permet l'extraction d'architectures de PL (Figure 7), ainsi que les connaissances nécessaires au déploiement final.

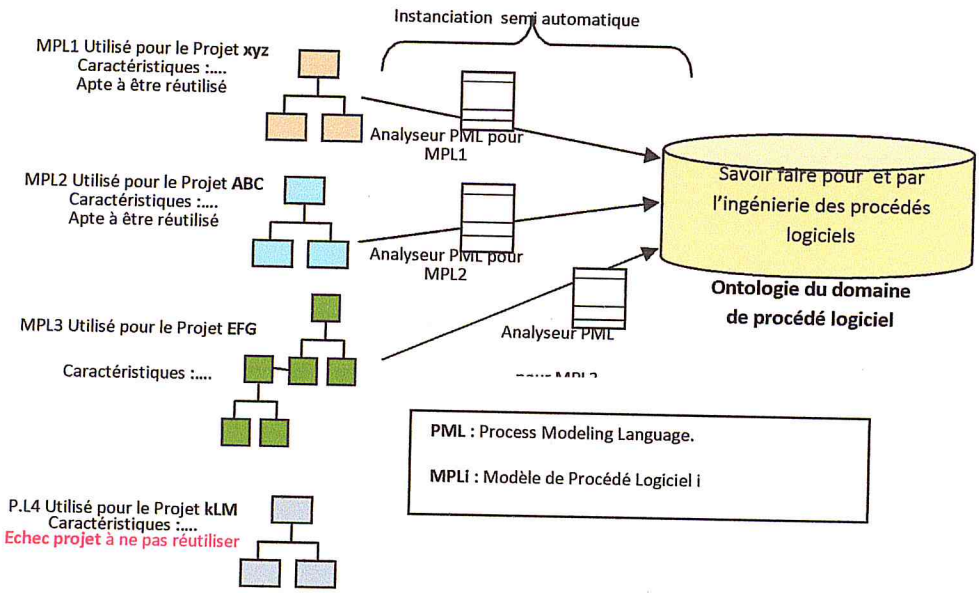

Figure 7 : Acquisition des connaissances éprouvées du domaine de l'ingénierie des procédés logiciels

Afin de capitaliser l'expérience de ce domaine, l'instanciation de cette ontologie se fait à partir de modèles de PLs existants et éprouvés. L'acquisition de ces connaissances passe nécessairement par une phase de réingénierie, des analyseurs de modèles de PLs doivent être développés à cet effet. Chaque langage de modélisation de PL doit avoir son analyseur pour permettre la capture des connaissances réutilisables. Une instanciation pertinente est celle qui permet d'identifier de manière unique chaque instance de l'ontologie. L'instanciation à partir de plusieurs modèles de PLs se heurte au problème de l'hétérogénéité du vocabulaire utilisés, ainsi il est important de définir une stratégie pour gérer ces d'instances synonymes.

1.2 Ingénierie par la réutilisation des PLs :

L'ingénierie par la réutilisation passe par génération de l'architecture de PLs, à partir des connaissances de l'ontologie, Les requêtes doivent prendre en considération la demande du développeur de procédé puis extraire les connaissances qui répondent le mieux à ces exigences.

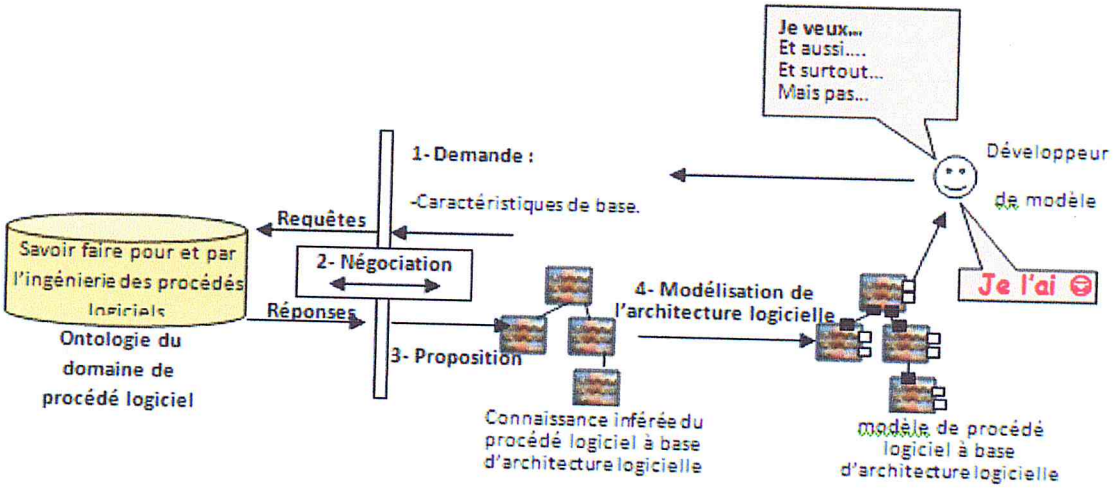

Figure 8 : Etape d'extraction d'architecture de PL

Les requêtes doivent permettre l'extraction deux trois type de connaissances:

1) Connaissances configuration PL : Identifier les éléments architecturaux (composants, connecteurs) et leur assemblage. Les connaissances configuration PL résultat sont décrites en exploitant un ADL générique ACME.
2) Connaissances composants PLs : représente les connaissances composants qui vont être utilisés lors du déploiement de l'architectures PLs. Ces connaissances sont décrient à l'aide de fichier XML.
3) Connaissances connecteurs PLs : représente les connaissances connecteur qui vont être utilisés lors du déploiement de l'architectures PLs. Ces connaissances sont décrient à l'aide de fichier XML.

2- Etape de réalisation de l'approche :

Comme cité précédemment, afin de réaliser de prototype, l'approche a été décomposée en plusieurs projets, chaque projet a été réalisé par un binôme d'étudiants dans le cadre le leur projet de fin d'études (master ou ingéniorat). Chaque projet traite un aspect particulier et réalise une solution souvent proposée par les étudiants eux même et non pas imposée ou définit au préalable.

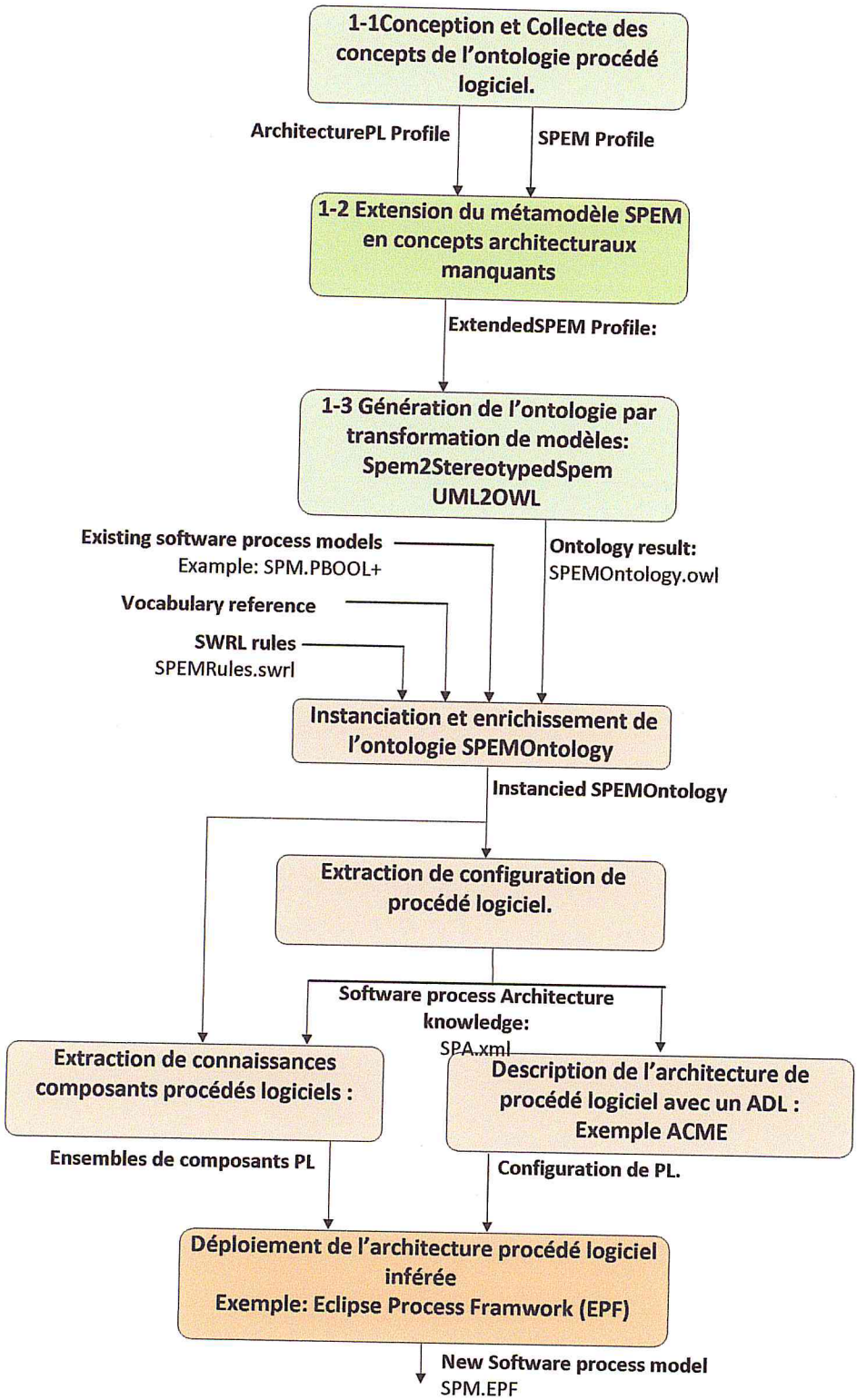

Figure 9 : Etapes suivie pour la réalisation de l'approche

La Figure 9 résume les étapes de réalisation du prototype. Chaque étape représente un projet de fin d'études excepté la première étape qui est plutôt une étude bibliographique et non une réalisation, nous détaillons chaque projet (réalisation) comme suit :

2-1 Conception et collecte des concepts de l'ontologie procédé logiciel :

Cette étape qui est une étude bibliographique avait pour objectif de mettre en place la conceptualisation adéquate de l'ontologie qui permettra de capitaliser les connaissances des Modèles de PLs hétérogènes qui sont déjà réalisé. Les ontologies existantes sont généralement spécifiques à un type de PL, ce qui ne permet pas leur réutilisation. La solution proposée est l'exploitation du méta modèle SPEM, qui est un standard adopté par l'OMG, ce choix est justifiée par :

- SPEM regroupe des concepts concernant plusieurs modèles de procédés logiciels. L'utilisation de SPEM permet de collecter un maximum de concepts pertinents concernant le domaine des procédés logiciels.
- A travers les sept packages de SPEM, plusieurs points de vue concernant les procédés logiciels sont offerts. Ainsi, il est possible d'inférer différents types de fragments procédés logiciels et de répondre non seulement à nos préoccupations (réutilisation à large échelle des modèles de procédés logiciels), mais aussi, de répondre à d'autres préoccupations telles que les préoccupations des méthodes situationnelles **[19]**, d'assistance ou documentation de procédés logiciels.
- SPEM est un standard adopté par l'OMG et un méta modèle référence dans le domaine de la modélisation des procédés logiciels. Notre intérêt est d'exploiter une conceptualisation déjà adoptée et acceptée par la communauté.
- SPEM est un profile UML, donc conforme au méta modèle UML2.0 **[32];** il est possible d'exploiter les différents outils, Framework développés dans le cadre de la MDA pour exploiter les concepts SPEM.
- Des règles de mapping entre différents méta modèles on été définit pas l'OMG et peuvent être exploité pour générer l'ontologie.

Cependant pour la description d'architecture de PLs, SPEM manquent de concepts architecturaux tels que la configuration PL, connecteurs et style PL, d'où la nécessite d'extension du méta modèle SPEM en concepts architecturaux manquant.

2.2 Extension du méta modèle SPEM en concepts architecturaux manquants [27]

Ce projet a permis de mettre en place un méta modèle générique pour la description d'architecture de PLs.

Ce méta modèle a été utilisé pour étendre le profil le SPEM afin de permettre la description d'architecture de PLs.

- **Avantage :** a permis de mettre en place un profil pour la description d'architecture de PL.
- **Difficultés :** Compréhension de différent domaines (architecture logicielles, ontologie, procédé logiciels, …)

2.3 Génération de l'ontologie par transformation de modèles : Spem2StereotypedSpem UML2OWL [29]

Ce projet a pour entré le profil UML décrivant le méta modèle SPEM, le résultat demandé est la génération de l'ontologie SPEMOntology. [29]

L'ontologie est générée automatiquement à partir du méta modèle SPEM en utilisant le langage de transformation de modèles ATL **[5]**. ATL (Atlas Transformation Language) est un langage de transformation de modèles basé sur le langage de contraintes OCL proposé par l'OMG, il est défini pour effectuer des transformations de modèles dans le cadre MDA (Model Driven Architecture). Une transformation ATL est composée de modules ATL. Pour notre génération t deux transformations existantes sont utilisées : UML2OWL **[30]** et UML2COPY **[30].**

UML2OWL et OWL2XML sont des modules qui fournissent des règles permettant la transformation d'un modèle UML en un modèle OWL.

Cependant, cette transformation n'est pas suffisante, car elle ne permet pas la transformation d'un modèle UML « stéréotypé » conforme à un profil UML en un modèle OWL. La transformation UML2OWL ne comporte pas de règles de transformation concernant les profils et leurs constituants (stéréotypes, valeurs marquées et contraintes). En effet, le modèle transformé est un modèle UML (modèle SPEM) conforme à un profil UML (profil SPEM), ainsi, une transformation ATL préalable est nécessaire. Cette transformation doit appliquer le profil SPEM au modèle SPEM ; elle doit appliquer chaque stéréotype à son élément UML (classe ou association) de manière à avoir des éléments stéréotypés avec des valeurs marquées et des contraintes prédéfinies. Une nouvelle transformation (ATL1) qui permet d'appliquer le profile SPEM au modèle SPEM est définit. Cette transformation est constituée de deux modules : UML2COPY qui permet de copier les éléments non stéréotypés et

ApplySPEMProfile2SPEMModel Module qui permet d'appliquer le profil SPEM aux modèle SPEM. Finalement, l'application successive des deux transformations ATL1, ATL2 permettent de générer notre SPEMOntology.

- **Difficultés :**
  - Un grand nombre de stéréotypes, classes, relations, cardinalités et propriétés à traiter.
  - Difficulté de comprendre la logique SPEM. Particulièrement, les différentes visions proposées pour chaque concept procédé.
- **Avantages :** L'ontologie générée est globale cohérente et utilisable directement.
- **Inconvénient :** Lors de la génération, certains nom de classes ont été modifiés ce qui introduit certaines difficultés de compréhension.

2.4 Instanciation l'ontologie SPEMOntology

L'application doit récupérer les connaissances pertinentes à partir de Modèle de PLs existants. Deux projets de fin d'études ont été réalisés : Instanciation de SPEMOntology à partir de modèles PBOOL+ **[26]**. Instanciation de SPEMOntology à partir de modèles EPF **[23]**.

Le problème à résoudre et de gérer l'hétérogénéité des connaissances résultat de la capitalisation des connaissances de sources hétérogènes. L'identification des synonymes et la mise en place d'un vocabulaire standard pour afficher les connaissances extraites.
La structure du méta modèle SPEM (organisée en plusieurs packages), a permis de faire la distinction entre :

- **Le vocabulaire référence** choisi par l'instanciation des concepts de Method Content Package (ce package est dédié à la définition des méthodes de développement indépendamment de leur utilisation).
- **Le vocabulaire utilisé** dans les modèles de procédés logiciels existants par instanciation des concepts Process Structure package et Process With Method package (ces packages sont destinés à la description des utilisations effectives des Méthodes de développement définis dans le package Method Content).
- **Faire la correspondance** entre utilisant les associations existantes dans SPEM. L'instanciation de l'ontologie se fait en utilisant la bibliothèque Jena, Jena est un Framework écrit en Java, qui permet de manipuler des documents RDF, RDFS et

OWL, il fournit en plus un moteur d'inférences permettant des raisonnements sur les ontologies. Nous utilisons ce Framework pour l'instanciation de notre ontologie.

Difficultés :

Comprendre l'ontologie de sa logique.

Identifier les connaissances à capitaliser et les concepts correspondant dans l'ontologie.

Avantages : Une première solution pour gérer l'hétérogénéité des connaissances sans introduire de nouveaux concepts.

Inconvénients : les instanciations sont partielles, les connaissances pertinentes ne sont pas toutes identifiées (prototypes).

2.5 Extraction de la configuration procédé logiciel [22].

Cette étape permet d'extraire des configurations de PL, la configuration peut respecter un style de PL, les algorithmes mis en place favorisent la réutilisation des configurations existantes, permettent de rechercher les assemblages de configuration, comme ils permettent de rechercher des fusions de configurations PLs.

- **Difficultés :** Les parties de l'ontologie utilisées pour l'extraction des configurations sont instanciées manuellement ce qui ajoute une difficulté à la réalisation.
- **Avantages :** La solution traite tous les cas possibles, exploite la théorie des graphes via des algorithmes de parcours de graphes.
  Le résultat est directement décrit dans le langage de description d'architecture ACME, directement utilisable.
- **Inconvénients :** La solution (configuration) est la première trouvé et elle n'est pas forcément l'optimale.

2.6 Extraction des connaissances des composants PL [25]

Cette application permet de générer les connaissances composant nécessaires pour le déploiement de l'architecture des PLs. Le principe est qu'à partir des noms des produits en entré et des noms de produits en sortie, les requêtes extrait un enchainement d'activités. Cet enchainement peut être une fusion d'activités de modèles de PLs hétérogène.

Difficultés : Compréhension de l'ontologie, cerner la différence entre certains concepts de l'ontologie tel que Activity, Task Use, sub activity.

Avantages : Affichage graphique des résultats, résultats sont en fichier XML.

Inconvénients : Algorithmes de recherche FIFO, pas de stratégie de recherche d'enchainement d'activités source de plusieurs modèles de procédés logiciels.

2.7 - Extraction de connecteur PL

Des connecteurs PL ont été identifiés, Ces connecteurs sont explicites et réutilisables, par conséquent, ont une structure connue. Par conséquent les connaissances connecteur PL ne sont pas extraites mais des connaissances sont adaptées.

Ce travail n'est pas encore réalisé et doit être réalisé lors de l'intégration.

2.8 - Déploiement de l'architecture PL extraite [24]:

Exemple: Eclipse Process Framework (EPF)

Ce projet permet de déployer des configurations de PLs et de générer le modèle de PL final, les fichiers en entré sont : le fichier configuration décrit en ADL ACME un langage de description d'architecture ; les fichiers composants PL et les fichiers connecteurs PL qui sont décrit à l'aide de balises XML. Ces fichiers sont les ressauts des applications précédentes.

Plusieurs types de déploiement sont possibles : Déploiement global, ou le modèle de PL est généré globalement. Le déploiement partiel qui permettra par la suite la modélisation de différents types de PLs (distribués, itératifs, incrémentaux…)

Difficultés :

- Compréhension d'EPF (Eclipse Process Framework), de sa structure et son raisonnement
- Traitement de tous les cas possibles avec tous les types de composants, connecteurs existants.

Avantages :

- Traitement de tous les types de connecteurs définis.
- Possibilité de déploiement global et déploiement partiel.

Inconvénients : Les fichiers doivent être corrects en nombre et en forme.

Développement du prototype final et Intégration des applications réalisés :

Les travaux qui doivent être intégré sont de cinq projets et qui sont comme suit :

- Instanciation l'ontologie SPEMOntology avec des modèles PBOOL+.
- Instanciation l'ontologie SPEMOntology avec des modèles EPF.
- Extraction de la configuration procédé logiciel.
- Déploiement d'architecture de PL en environnement EPF.

Les autres sont des étapes de création de l'ontologie donc exécuté une fois (génération de l'ontologie) et ne sont pas ingéré dans le prototype global.

Conclusion :

Dans ce chapitre nous avons présenté l'approche de réutilisation de PL à base d'architecture logicielle, nous avons aussi présenté les applications qui ont été réalisé et qui doivent être intégré pour avoir le premier prototype de l'approche. Dans ce qui suit nous présentons la conception de notre application et la stratégie suivie pour intégrer ces différentes applications.

1- Introduction

Après avoir donné un léger aperçu de l'approche suivie, à présent nous entamons la partie conception de cette dernière. Tout d'abord nous abordons par des définitions l'approche de conception utilisée, on suit on étudiera les besoins fonctionnelles et techniques de notre application, et pour terminer nous allons conclure avec quelques techniques utilisées lors de notre travail.

2- Méthode de conception

Toute méthode de conception est constituée d'un langage de modélisation et d'une démarche de réalisation. Pour ce travail, nous avons opté pour un cycle en vie en cascade, ce choix et due pour la simplicité la facilité avec la quelle il est connue. En ce qui concerne le langage de modélisation notre choix c'est porté sur UML pour ça standardisation dans le milieu des langages de modélisation orientés objets.

3.1 Présentation du cycle de vie en cascade :

Décrit par Royce en 1970, ce dernier à été largement dans le milieu utilisé pour la description des activités liées au logiciels. Il décrit le cycle de vie comme étant une suit de phases qui s'enchainent d'une manière linéaire (Figure 10 : cycle en vie d'un logiciel « modèle en cascade ») [26].

2.1.1 Les activités :

- **Expression des besoins :**

L'expression des besoins comme son nom l'indique, permet de définir et de répertorier les différents besoins :

- Inventorier les besoins principaux et fournir une liste de leurs fonctions
- Recenser les besoins fonctionnelles (du point de vue utilisateur).
- Adhérer les besoins non fonctionnels (techniques), et livrer une liste des exigences.

En UML, Le modèle des cas d'utilisation présent le système du point de vue de l'utilisateur, et présent sous la forme de cas d'utilisation et d'acteur les besoins du client.

- **Analyse :**

L'objectif de l'analyse étant d'accéder à la compréhension des besoins et des exigences du client. Il s'agit de livrer des spécifications pour permettre la mis en place de la conception.

Un modèle d'analyse livre une spécification complète des besoins issus de l'étape précédente et les structure sous une forme qui facilite la compréhension, la préparation et la maintenance future du système.

Il s'écrit dans un langage développeurs, et il peut être considéré comme une première ébauche du modèle de conception.

- **Conception :**

La conception permet d'acquérir une compréhension approfondie des contraintes liées au langage de programmation, à l'utilisation des composants du système d'exploitation.

Elle détermine les principales interfaces et les transcrit à l'aide d'une notation commune.

Elle constitue le point du départ de l'implémentation :

- Elle décompose le système en sous systèmes.
- Elle crée des abstractions de l'implémentation.

- **Implémentation :**

L'implémentation est le résultat de la conception pour implémenter le système sous forme de composants, c'est-à-dire en forme de code source, de scripts, de binaires, d'exécutables et d'autres éléments du même type.

- **Test :**

Les testes permettent de vérifier des résultats de l'implémentation en testant la construction. Pour mener à bien cette tâche il faut bien choisir ces cas de tests, les implémenter, et veuillez à ce que ces derniers respectent bien les besoins décrit par l'utilisateur.

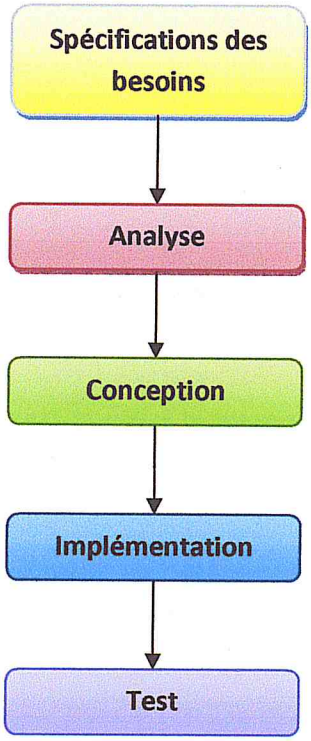

Figure 10 : cycle en vie d'un logiciel « modèle en cascade »

3- Conception de l'application

3-1 Expression des besoins :

3.1.1 Identification des acteurs :

La première étape de cette phase est d'énumérer les acteurs susceptibles d'interagir avec le système. Un Acteur représente l'abstraction d'un rôle joué par des entités externes (utilisateur, dispositif matériel ou autre système), qui interagissent directement avec le système étudié.

- **Les acteurs :**

Acteur	Désignation
Expert PL	L'expert PL est un utilisateur privilégié, il a comme rôle l'enrichissement de notre ontologie de domaine.
Concepteur PL	Le concepteur PL est un utilisateur privilégié, il joue un rôle primordial dans les activités de déploiement et d'extractions d'architectures logiciels.

Tableau 1 : Les acteurs du système

- **Description des besoins fonctionnels :**

Acteur	Description des besoins fonctionnels
Expert PL	L'application doit permettre à l'expert PL : • Sélectionner un fichier de procédée logiciel. • Lancer le processus d'enrichissement de notre ontologie « instanciation ». • Visualiser le résultat de cette dernière.
Concepteur PL	L'application doit permettre au concepteur PL de faire les deux activités suivantes : 1) Le déploiement : • Sélectionner une architecture logicielle à déployer. • Lancer le processus de déploiement. • Visualiser le résultat du déploiement. 2) L'extraction : • Sélectionner un ensemble de ports en entrées et en sortis. • Lancer le processus d'extraction d'architecture logicielle. • Visualiser le résultat de cette dernière.

Tableau 2: Description des besoins fonctionnels

3.1.2 Identification des cas d'utilisation

L'identification des cas d'utilisation, donne un aperçue des fonctionnalités futures que doit implémenter le système.

Dans le cadre de cette étude j'ai définis un diagramme de cas d'utilisation globale suivis par d'autres diagrammes du même genre mais plus détaillées.

La description de chaque cas d'utilisation est va être détaillée de la façon suivante :

Elément	Signification
Cas d'utilisation	Le nom du cas d'utilisation.
Acteur	L'acteur qui réalise le cas d'utilisation.
But	Le but du cas d'utilisation.
Description	Une explication du cas d'utilisation.
Pré condition	Les conditions qui doivent être vérifiés afin d'effectuer les cas d'utilisation.
Post condition	Le résultat attendu du cas d'utilisation.
Exception	Les informations pour les quelles un disfonctionnement pourrez avoir lieu.

Tableau 3: Modèle représentatif des cas d'utilisation

3.1.3 Diagramme de cas d'utilisation global :

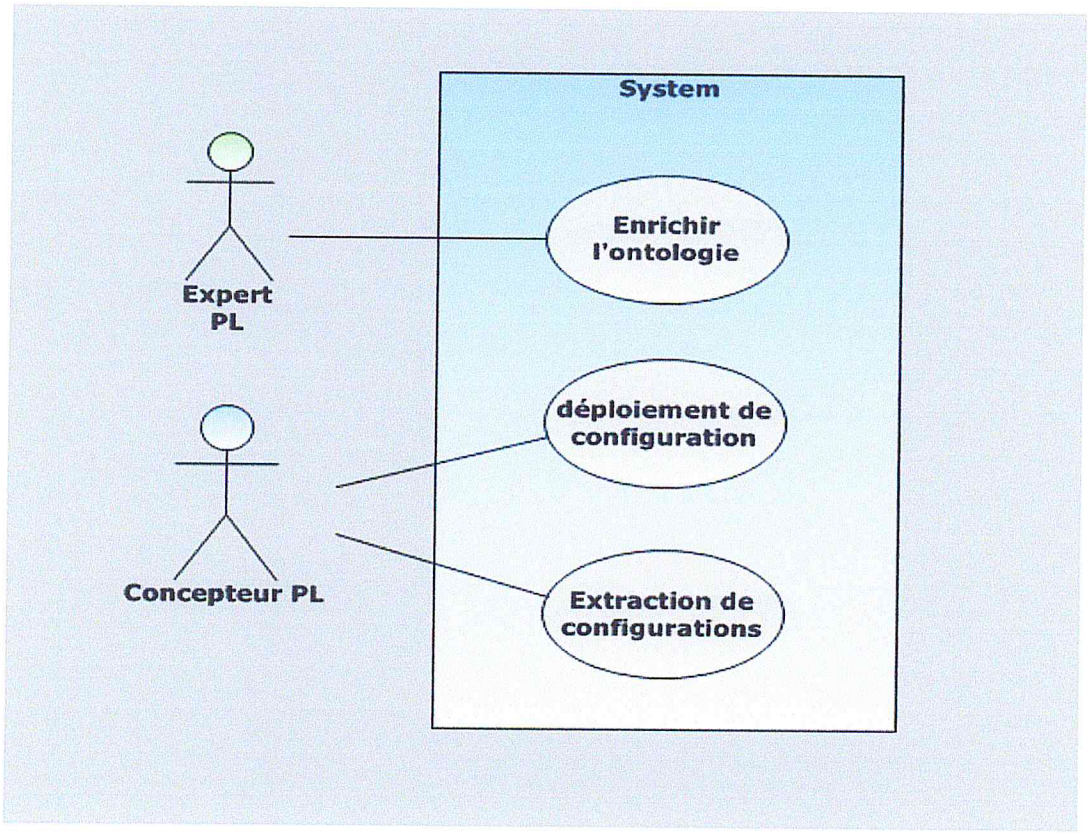

Figure 11: Diagramme de cas d'utilisation global

Enrichir l'ontologie	
Acteur	L'expert PL
But	Ajouter des nouvelles connaissances PL à notre ontologie de domaine.
Description	Cette ne peut être effectué que l'expert PL en personne.
Pré condition	Le fichier PL doit être sélectionné.
Post condition	Aucune.
Exception	Annulation dans l'un des cas suivant : ➢ Le fichier PL ne correspond pas à son PML. ➢ Le fichier PL est inaccessible. ➢ Le fichier PL n'est pas supporté par le système en question.

Tableau 4 : Description détaillé du cas d'utilisation « enrichir une ontologie »

Déploiement d'une configuration	
Acteur	Concepteur PL
But	Déployer une configuration.
Description	Cette étape a pour but de déployer une configuration en ACME sous la forme d'un procédé logicielle.
Pré condition	Une configuration qui suit le méta model ACME doit être sélectionnée.
Post condition	Aucune.
Exception	Annulation dans l'un des cas suivant : ➢ Le fichier de configuration contient des erreurs. ➢ Le fichier de configuration est inaccessible. ➢ Aucune correspondance n'est trouvée.

Tableau 5 : Description détaillé du cas d'utilisation « déploiement d'une configuration »

Extraction d'une configuration	
Acteur	Concepteur PL
But	Extraire une configuration(s).
Description	Cette étape a pour but d'extraire une configuration ou un enchainement de configuration qui respectent l'ADL ACME, ces derniers doivent satisfaire des ports data en entrées et des ports data en sortis.
Pré condition	Un ensemble de ports data en entrées et ensemble de port data en sortis doivent être sélectionnées par le concepteur PL.
Post condition	Aucune.
Exception	Annulation dans l'un des cas suivant : ➢ Le fichier de configuration contient des erreurs. ➢ Le fichier de configuration est inaccessible. ➢ Aucune correspondance n'est trouvée.

Tableau 6 : Description détaillé du cas d'utilisation « extraction d'une configuration »

3.1.4 diagrammes de cas d'utilisations détaillés :

3.1.4.1 Cas d'utilisation enrichir l'ontologie :

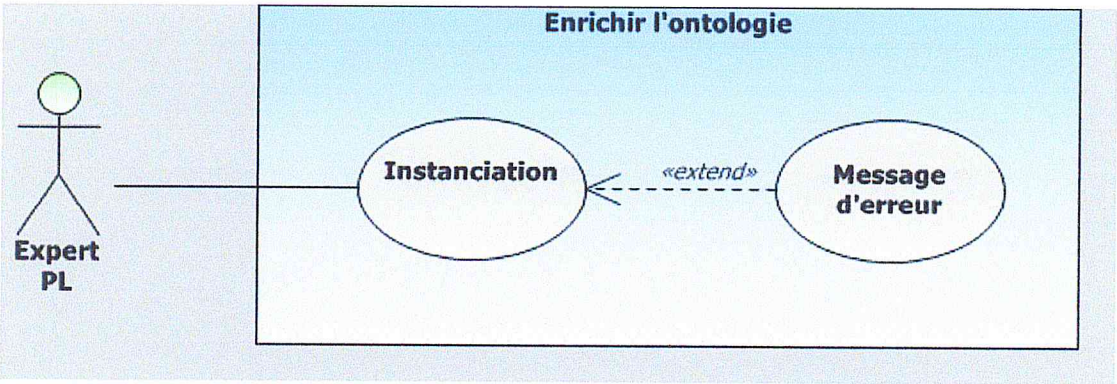

Figure 12 : diagramme de cas d'utilisation détaillé « enrichir l'ontologie »

Acteur	Expert PL
But	Enrichir l'ontologie.
Description	Cette étape a pour but d'enrichir notre ontologie par des connaissances qui proviennent des fichiers de procédées logiciels, ces derniers peuvent respecter de modèles de PL différents.
Pré condition	Le fichier PL doit être sélectionné.
Post condition	Aucune.
Exception	Annulation dans l'un des cas suivant : ➢ Le fichier PL ne correspond pas à son PML. ➢ Le fichier PL est inaccessible. ➢ Le fichier PL n'est pas supporté par le système en question.

Tableau 7 : Description détaillée du cas d'utilisation « instanciation »

3.1.4.2 Cas d'utilisation Déploiement d'une configuration :

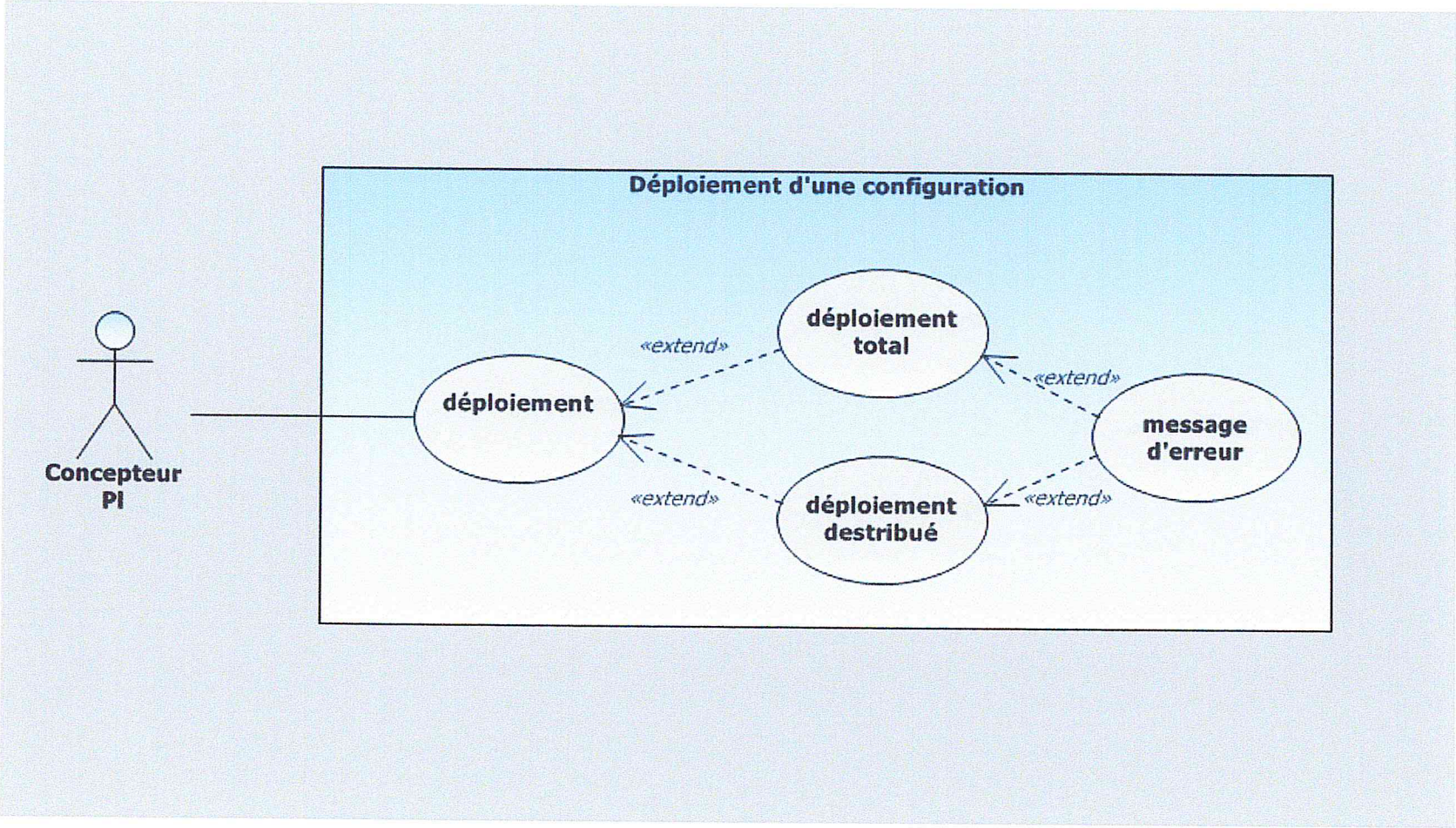

Figure 13: diagramme de cas d'utilisation détaillé « déploiement d'une configuration »

Déploiement total	
Acteur	Concepteur PL
But	Déployer une architecture logicielle écrite suivant l'ADL ACME.
Description	Cette action a pour effet de déployer une configuration vers un fichier procédé logiciel EPF et cela en suivant le mode total.
Pré condition	Le fichier de configuration ACME doit être sélectionné.
Post condition	Aucune.
Exception	Annulation dans l'un des cas suivant : ➢ Le fichier de configuration ne correspond pas aux standards décrits par l'ADL ACME. ➢ Le fichier de configuration est inaccessible.

Tableau 8 : Description détaillé du cas d'utilisation «Déploiement total »

Déploiement distribué	
Acteur	Concepteur PL
But	Déployer une architecture logicielle écrite suivant l'ADL ACME.
Description	Cette action a pour effet de déployer une configuration vers un fichier procédé logiciel EPF et cela en suivant le mode distribué.
Pré condition	Le fichier de configuration ACME doit être sélectionné.
Post condition	Aucune.
Exception	Annulation dans l'un des cas suivant : ➢ Le fichier de configuration ne correspond pas aux standards décrits par l'ADL ACME. ➢ Le fichier de configuration est inaccessible.

Tableau 9 : Description détaillé du cas d'utilisation «Déploiement distribué»

3.1.4.3 Cas d'utilisation extraction de configuration :

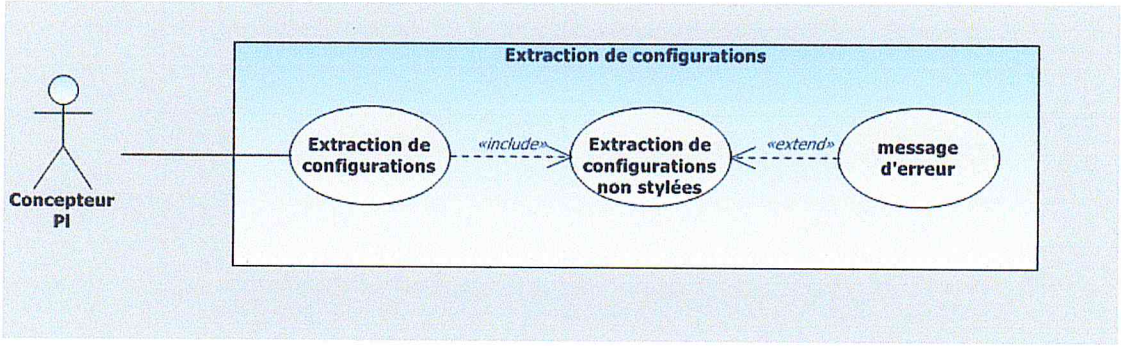

Figure 14: diagramme de cas d'utilisation détaillée « extraction d'une configuration »

Extraction d'une configuration	
Acteur	Concepteur PL
But	Extraire une configuration.
Description	Cette étape a pour but d'extraire une configuration ou un enchainement de configuration qui respectent l'ADL ACME, ces derniers doivent satisfaire des ports data en entrées et des ports data en sortis. L'extraction doit ce faire en utilisant l'algorithme « extraction de configuration non stylé » d'où le cas d'utilisation **utilisée** tire son nom.
Pré condition	Un ensemble de ports data en entrées et ensemble de port data en sortis doivent être sélectionnées par le concepteur PL.
Post condition	Aucune.
Exception	Annulation dans l'un des cas suivant : ➢ Le fichier de configuration contient des erreurs. ➢ Le fichier de configuration est inaccessible. ➢ Aucune correspondance n'est trouvée.

Tableau 10 : Description détaillé du cas d'utilisation «Déploiement distribué»

4- Analyse du système

4.1 Diagramme global de composants

Ce diagramme permet de cerner l'architecture globale de l'application, en effet la grande taille de l'application, nous oblige à identifier la structure globale de l'application avant de détailler sa conception.

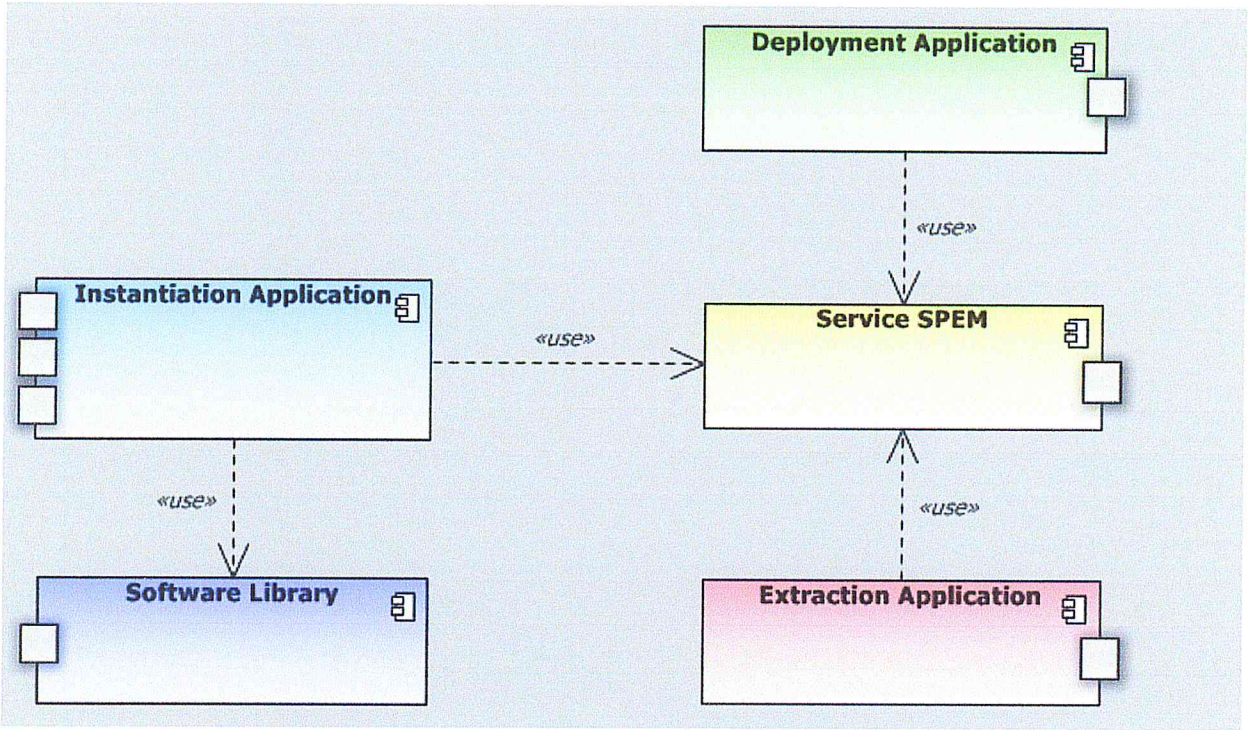

Figure 15: diagramme global des composants

Description :

Dans le diagramme de composants décrit précédemment, le système dans sa globalité a été découpé en un ensemble de composants, ces derniers jouent le même rôle décris par les différents cas d'utilisation du diagramme de cas d'utilisation global, a part deux nouveaux composants Service SPEM et Software Library qui sont des composant essentiel pour la mise en œuvre de notre système et qui ont été ajoutés pour satisfaire les besoins fonctionnels de notre application.

Nous détaillerons chaque composant, son rôle ainsi que sa structure interne dans la partie dédié à la conception.

4.2 Les diagrammes de séquences :

4.2.1 Le diagramme de séquence enrichir l'ontologie :

Scénarios des diagrammes de séquences :

4.2.1.1 le diagramme de séquences Instanciation :

1) L'expert PL sélectionne un fichier de procédés logiciel.
2) L'expert PL lance le processus d'instanciation.
3) L'application d'instanciation charge le composant « service instanciation ».
4) L'application d'instanciation met à disposition du composant « service instanciation » la l'ontologie.
5) L'application d'instanciation met à disposition du composant « service instanciation » la librairie des logiciels.
6) L'application de l'instanciation lance le processus depuis le composant « service instanciation ».
7) Le service d'instanciation se charge de trouver un instantiateur approprié au type du fichier de procédé logiciel.

Si les résultats de recherche sont fructueux alors :

- Le composant « service instanciation » charge l'instantiateur dédié pour le type de fichier que l'expert PL à sélectionné.
- Le composant « service instanciation » met à disposition de l'instantiateur l'ontologie de domaine.
- Lancer l'algorithme d'instanciation local « propre à chaque type d'instantiateur ».

Sinon l'état « fichier PL non pris en charge » sera retourné.

Tous les autres cas sont considérés comme étant des cas erreurs, et dans cette situation le composant « service instanciation » retourne l'état « erreur interne ».

8) Le composant « service instanciation » retourne l'état de l'opération.
9) Le composant « service instanciation » est détruit.
10) L'application d'instanciation affiche à l'utilisateur un message, ce dernier peut varier selon l'état de l'opération que le composant « service instanciation » lui a retourné.

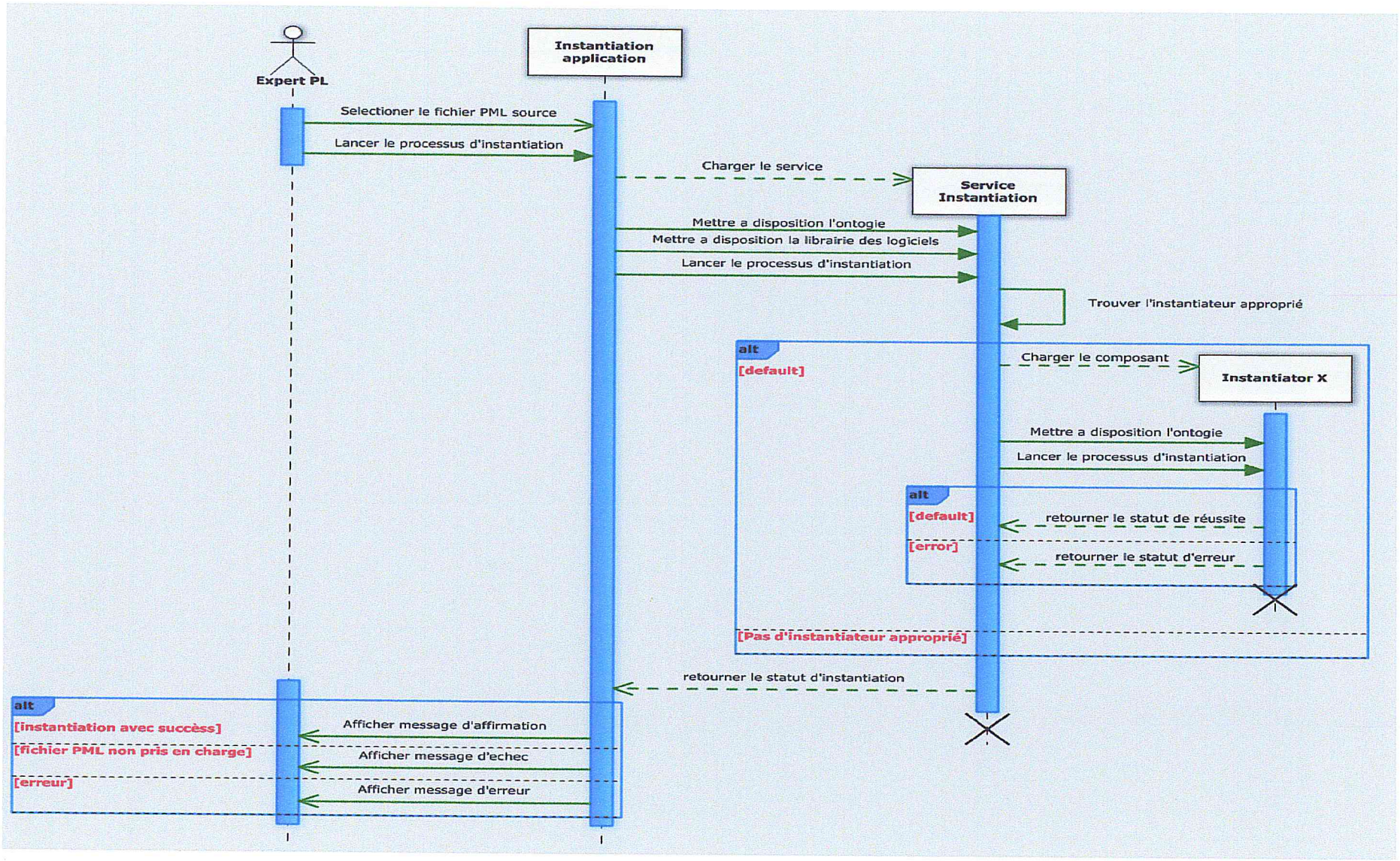

Figure 16 : Diagrammes de séquences « Instanciation »

4.2.2 Le diagramme de séquences déploiement de configurations :

4.2.2.1 Le diagramme de séquences déploiement total :

1) Le concepteur PL sélection la configuration ACME à déployer.

2) Le concepteur PL lance le déploiement total.

3) L'application du déploiement charge le composant « Deployment Tool ».

4) L'application du déploiement met à disposition du composant « Deployment Tool » l'ontologie de domaine.

5) L'application du déploiement met à disposition du composant « Deployment Tool » un dossier de déploiement « output folder ».

6) L'application du déploiement lance la méthode du déploiement total.

7) Le composant « Deployment Tool » retourne le résultat de l'opération.

8) Le composant « Deployment Tool » est détruit.

9) Un message est affiché à l'utilisateur indiquant l'état de l'opération.

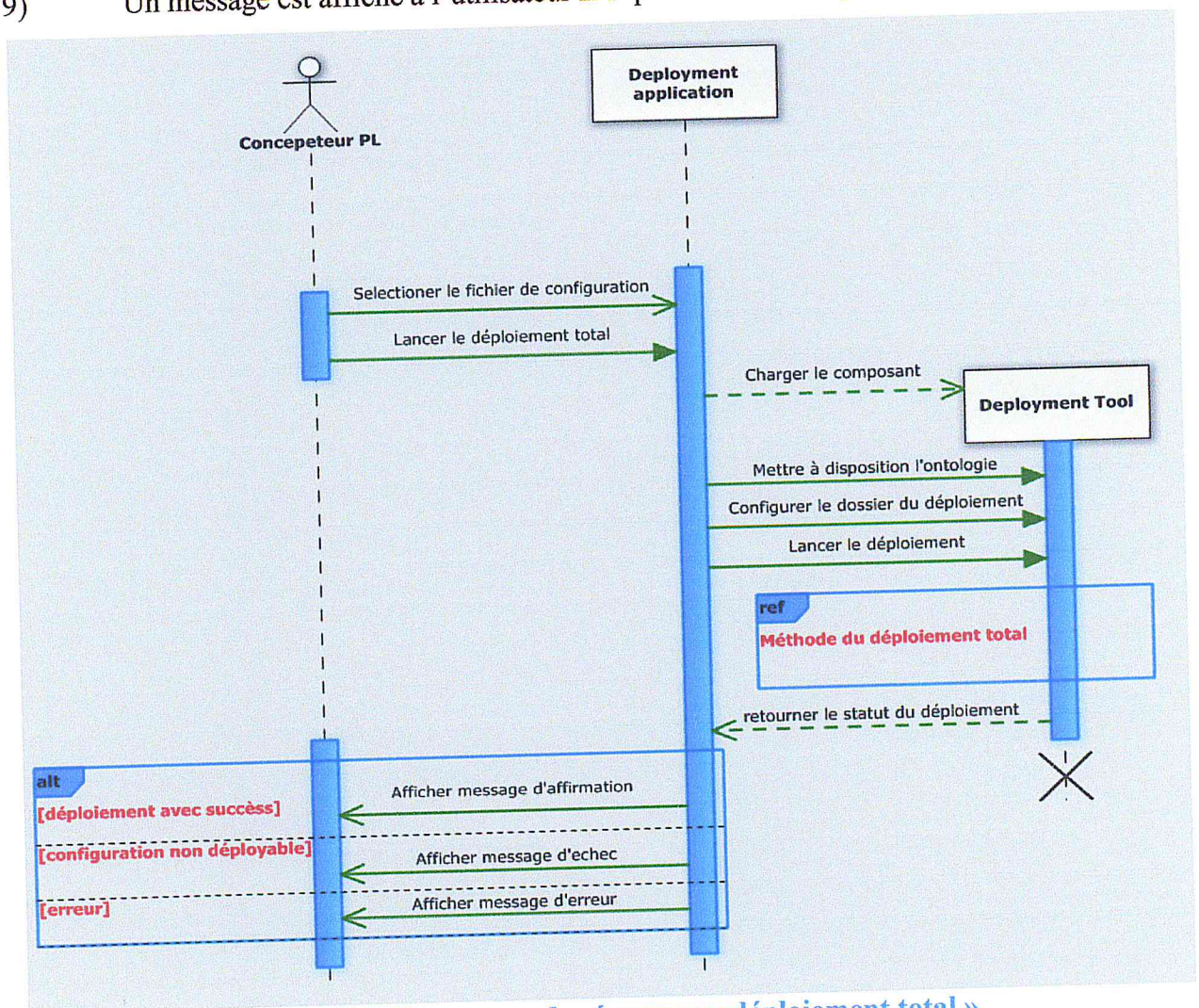

Figure 17 : diagramme de séquences «déploiement total »

4.2.2.1.1 Diagramme de séquences « méthode de déploiement total » :

1) Le composant « Deployment Tool » charge le composant « Full Deployment ».

2) Le composant « Deployment Tool » met à disposition du composant « Full Deployment» l'ontologie de domaine.

3) Le composant « Deployment Tool » met à disposition du composant « Full Deployment » un dossier de déploiement « output folder ».

4) Le composant « Full Deployment » lit la configuration ACME.

5) Le composant « Full Deployment » recherche les connaissances des composants.

6) Le composant « Full Deployment » recherche les connaissances des connecteurs.

7) Le composant « Full Deployment » crée un document EPF vide.

8) Le composant « Full Deployment » crée les informations de déploiement pour les composants.

9) Le composant « Full Deployment » crée les informations de déploiement pour les connecteurs.

10) Le composant « Full Deployment » injecte les informations de déploiement des connecteurs et composants dans le document EPF.

11) Le composant « Full Deployment » retourne l'état de réussite de l'opération.

12) Le composant « Full Deployment » est détruit.

Si les connaissances composant ou connecteurs sont manquantes :

- La méthode est arrêtée.
- Un état échec de l'opération est retourné.
- Le composant « Full Deployment » est détruit.

S'il y a des erreurs dans la phase lecture du fichier ACME ou l'écriture du fichier EPF :

- La méthode est arrêtée.
- Un état d'erreur est retourné.
- Le composant « Full Deployment » est détruit.

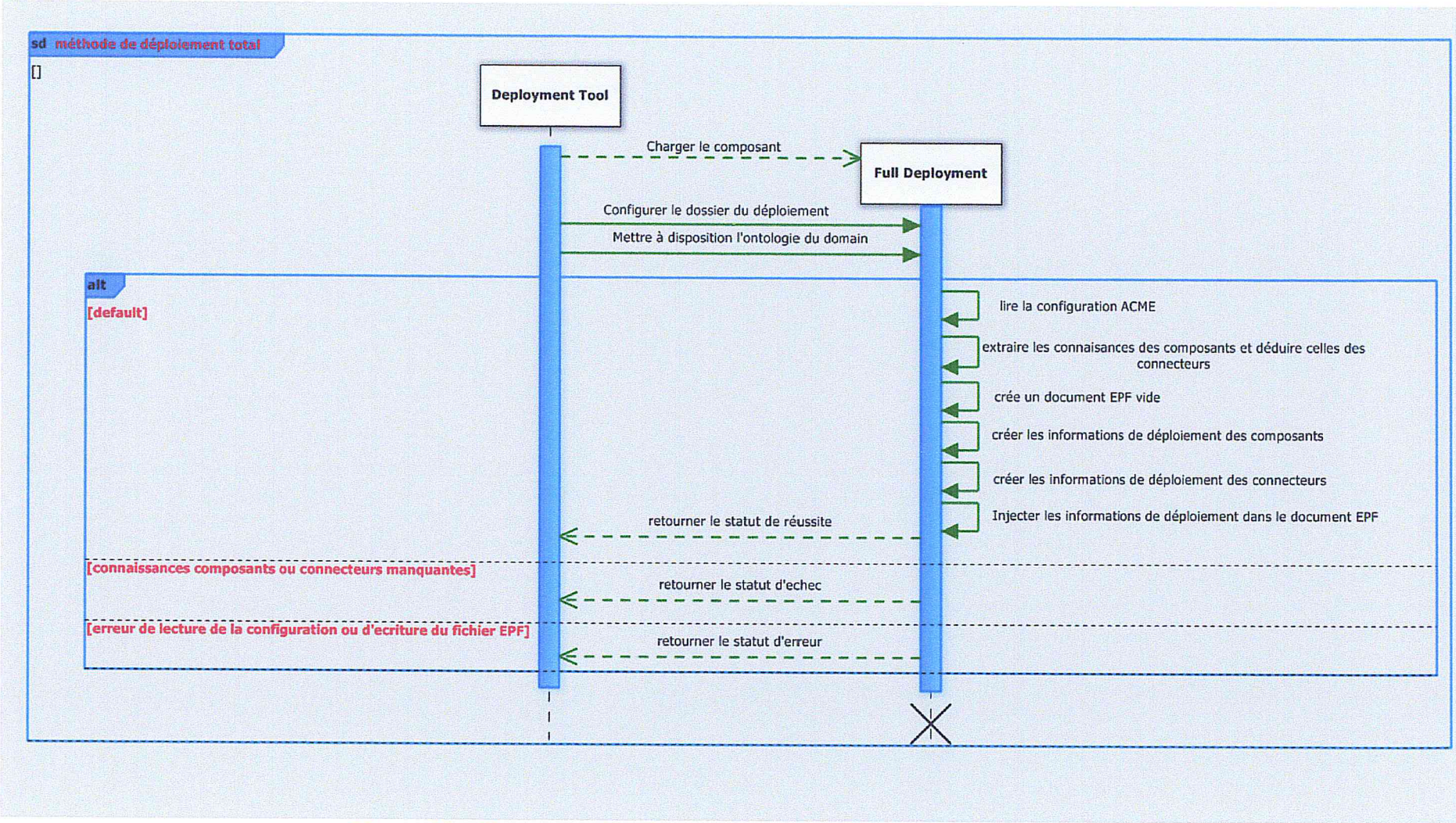

Figure 18 : diagramme de séquences « méthode de déploiement total »

4.2.2.2 Le diagramme de séquences déploiement distribué :

1) Le concepteur PL sélectionne la configuration ACME à déployer.

2) Le concepteur PL lance le déploiement distribué.

3) L'application du déploiement charge le composant « Deployment Tool ».

4) L'application du déploiement met à disposition du composant « Deployment Tool » l'ontologie de domaine.

5) L'application du déploiement met à disposition du composant « Deployment Tool » un dossier de déploiement « output folder ».

6) L'application du déploiement lance la méthode du déploiement distribué.

7) Le composant « Deployment Tool » retourne le résultat de l'opération.

8) Le composant « Deployment Tool » est détruit.

9) Un message est affiché à l'utilisateur indiquant l'état de l'opération.

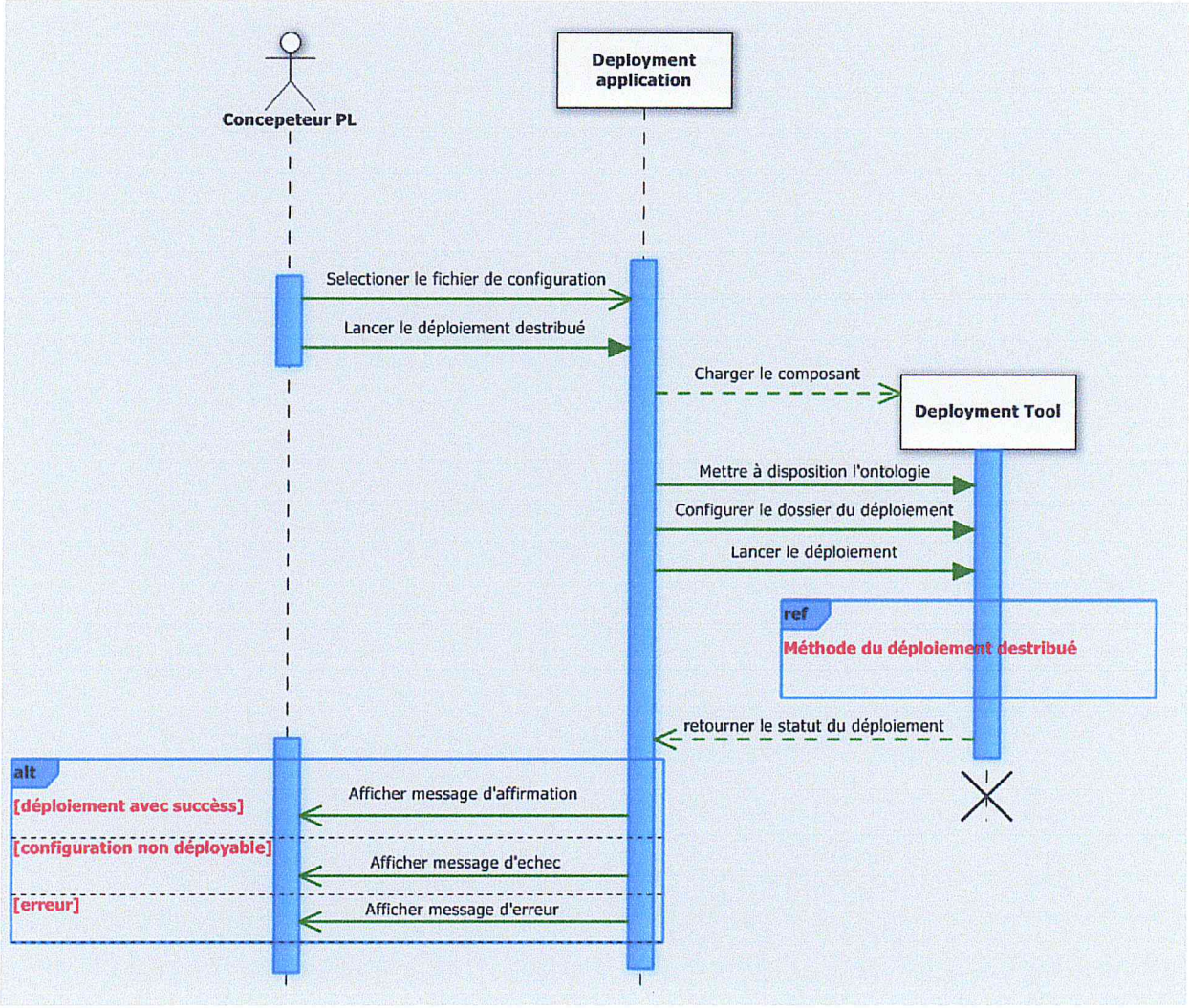

Figure 19 : Diagramme de séquences « déploiement distribué »

4.2.2.2.1 Diagramme de séquence référence « méthode du déploiement total » :

1) Le composant « Deployment Tool » charge le composant « Distributed Deployment ».

2) Le composant « Deployment Tool » met à disposition du composant « Distributed Deployment» l'ontologie de domaine.

3) Le composant « Deployment Tool » met à disposition du composant «Distributed Deployment» un dossier de déploiement « output folder ».

4) Le composant « Distributed Deployment » lit la configuration ACME.

Pour chaque composant faire:

5) Le composant « Distributed Deployment » recherche les connaissances des composants.

6) Le composant « Distributed Deployment » crée un document EPF vide.

7) Le composant « Distributed Deployment » crée les informations de déploiement pour les composants.

8) Le composant « Distributed Deployment » injecte les informations de déploiement dans le document EPF.

9) Le composant « Distributed Deployment » retourne l'état de réussite de l'opération.

10) Le composant « Distributed Deployment » est détruit.

Si les connaissances composant sont manquantes :

- La méthode est arrêtée.
- Un état échec de l'opération est retourné.
- Le composant « Distributed Deployment » est détruit.

S'il y a des erreurs dans la phase lecture du fichier ACME ou l'écriture du fichier EPF :

- La méthode est arrêtée.
- Un état d'erreur est retourné.
- Le composant « Distributed Deployment » est détruit.

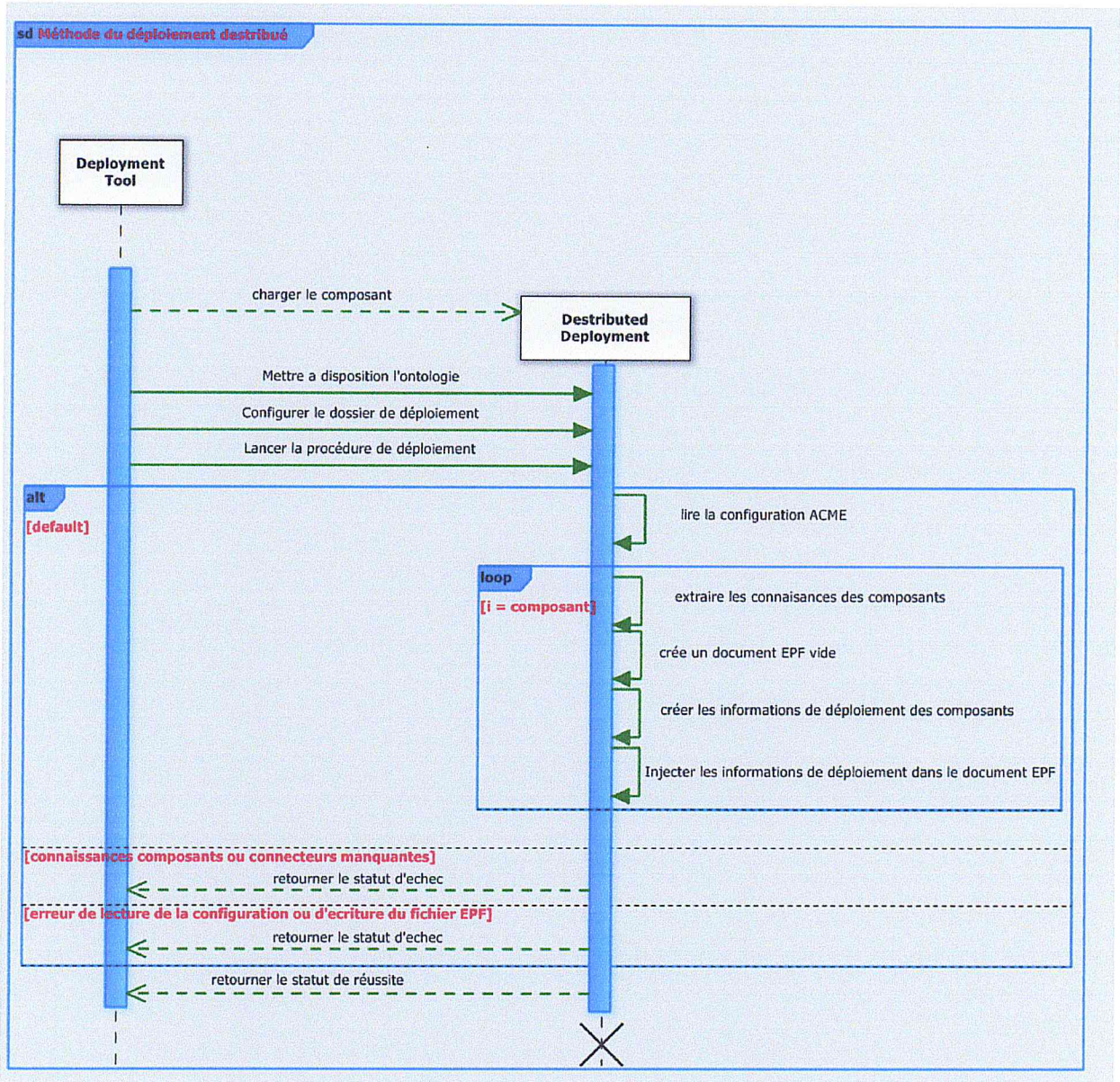

Figure 20 : diagramme de séquence « méthode de déploiement distribué »

4.2.3 Diagramme de séquences Extraction de configurations :

4.2.3.1 Le diagramme de séquences Extraction de configurations non stylées :

1) Le concepteur PL sélectionne les ports data in et out.

2) Le concepteur PL lance le processus d'extraction.

3) L'application de l'extraction charge le composant « Extraction Tool ».

4) L'application de l'extraction met à disposition du composant « Extraction Tool » l'ontologie du domaine.

5) L'application de l'extraction met à disposition du composant « Extraction Tool » un dossier de sortis.

6) L'application de l'extraction lance le processus d'extraction.

7) Le composant « Extraction Tool » charge le composant « Unstyled Configuration Extractor ».

8) Le composant « Extraction Tool » met à disposition du composant « Unstyled Configuration Extractor » l'ontologie du domaine.

9) Le composant « Extraction Tool » met à disposition du composant « Unstyled Configuration Extractor » un dossier de sortis.

10) Le composant « Extraction Tool » lance le processus d'extraction.

11) Le composant « Unstyled Configuration Extractor » :

- Cherche une configuration déjà existante.
- Sinon, un enchainement de configurations.
- Sinon, un enchainement de composants, que ces derniers vont servir pour la construction de la configuration personnalisée.

12) Le composant « Unstyled Configuration Extractor » retourne l'état de succès.

13) Le composant « Unstyled Configuration Extractor »est détruit.

14) Le composant « Extraction Tool » retourne l'état de l'opération.

15) Le composant « Extraction Tool » est détruit.

16) L'application de l'extraction affiche un message à l'utilisateur selon l'état du processus d'extraction de configurations.

Si aucune configuration n'est trouvée :

- Le composant « Unstyled Configuration Extractor » retourne l'état d'échec.
- Le composant « Unstyled Configuration Extractor »est détruit.

Si une erreur apparait :

- Le composant « Unstyled Configuration Extractor » retourne l'état d'erreur.
- Le composant « Unstyled Configuration Extractor » est détruit.

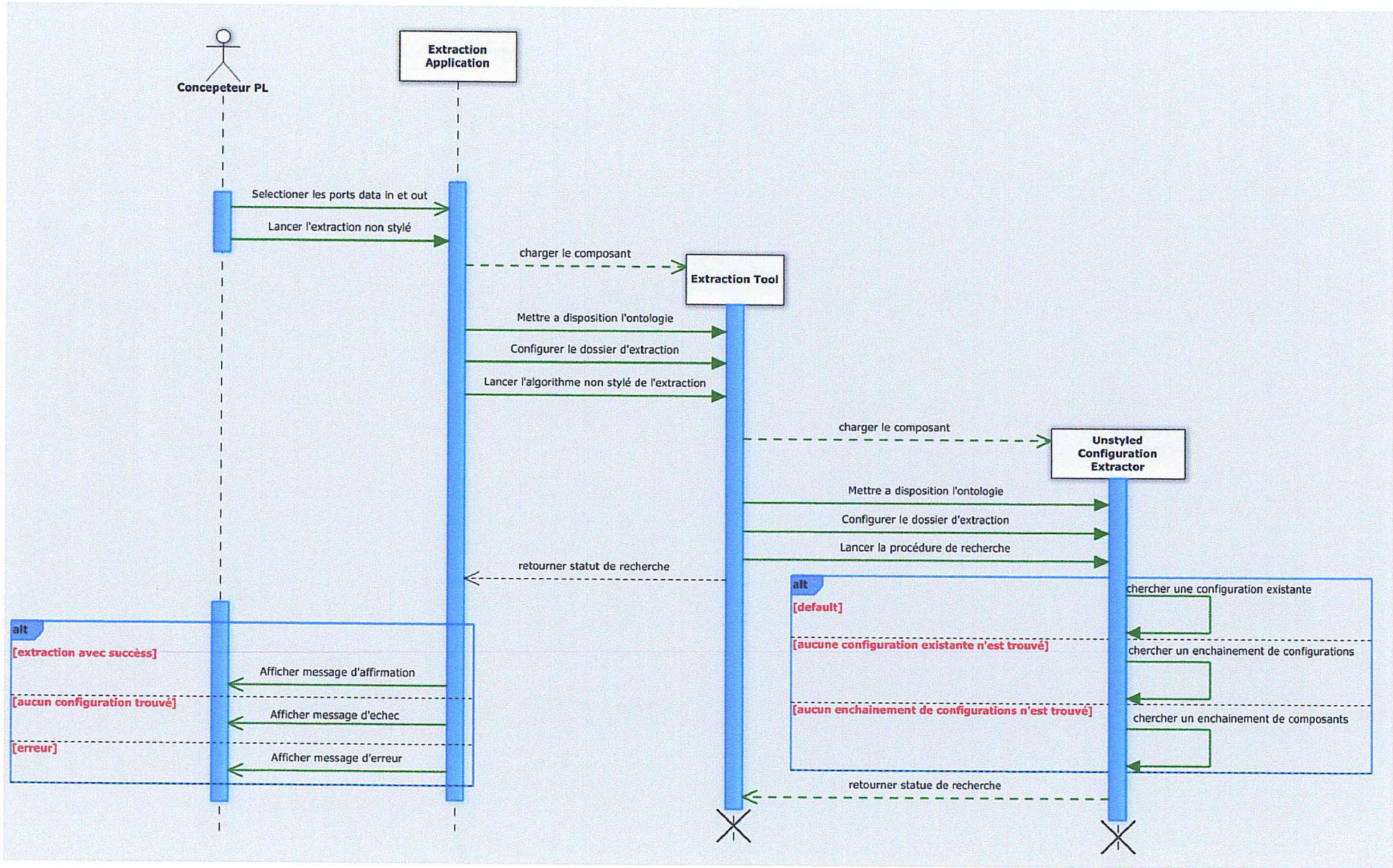

Figure 21 : diagramme de séquences « Extraction de configurations non stylées »

5 Conception du system :

Dans cette section nous allons aborder la conception par un ensemble de diagrammes de composants qui vont décrire l'aspect structurelle et fonctionnelle de notre système, pour chaque diagramme de composants nous allons évoquer son rôle, ces objectifs, suivis du diagramme de classes associé.

Nous complétons cette section par un ensemble d'algorithmes que nous avons spécialement crée pour l'accomplissement des objectifs de notre système.

Notre application est composée d'une dizaine de composants qui regroupent une dizaine de classes. En prenant en compte le nombre considérable des classes de l'application, nous détaillons jusqu'au niveau élémentaire les composants : Service SPEM et Instantiation Service.

5.1 Service SPEM:

- **Diagramme de classes :**

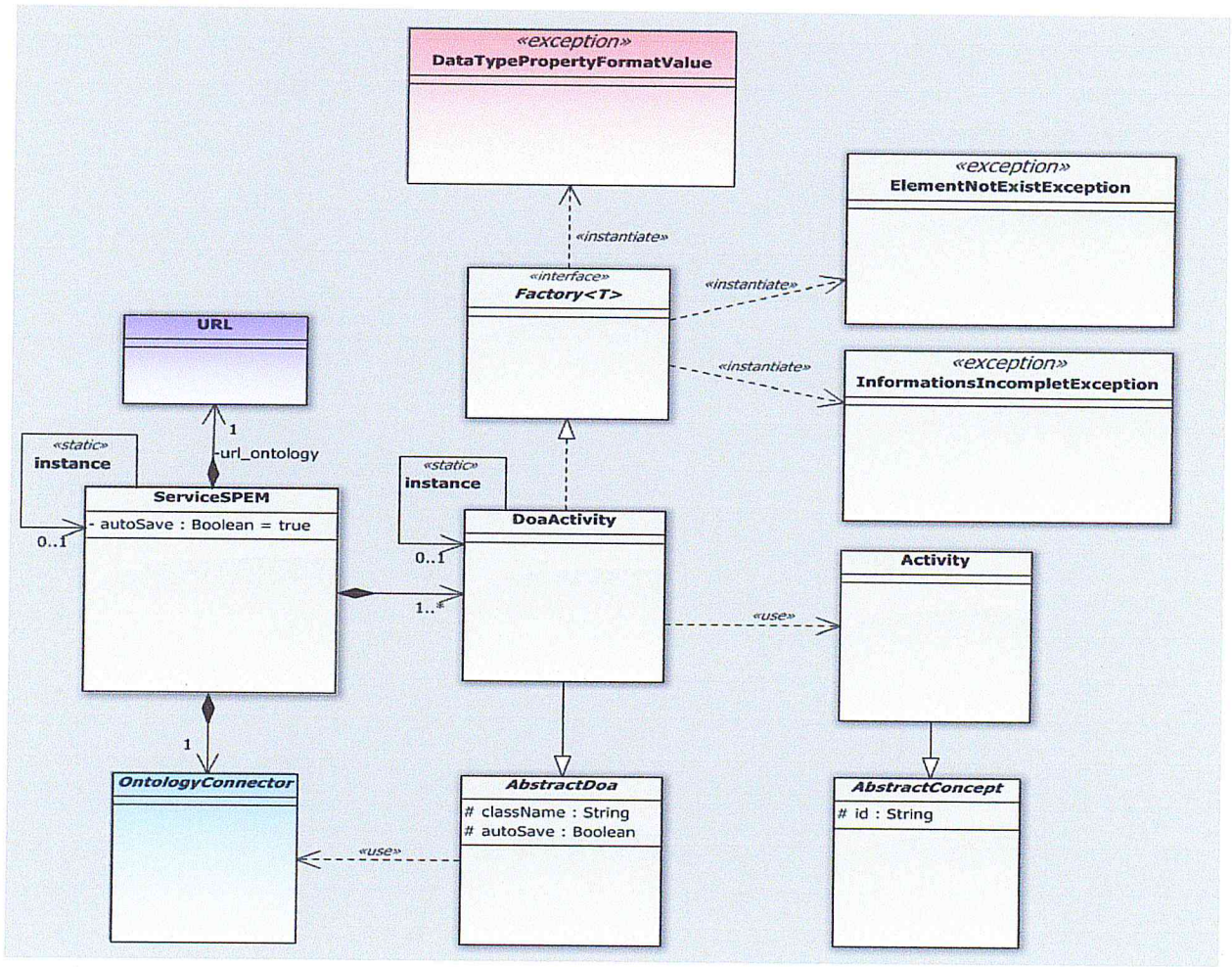

Figure 22 : diagramme de classes « Service SPEM »

- **Description diagramme de classes :**

Factory	
Type de classe	Classe abstraite, générique, interface.
Super classe	Aucune.
Méthode	
Nom	**Description**
+create(arg : T)	
+save(arg : T)	
+modify(arg : T)	
+remove(arg : String)	
+find(arg : String)	
+getAllReference(arg : String) : String[]	

AbstractDoa	
Type de classe	Classe abstraite.
Super classe	Aucune.
Attributs	
Nom	**Description**
#id : String	Cet attribut renferme le nom de la classe que le Doa gère.

AbstractConcept	
Type de classe	Classe abstraite.
Super classe	Aucune.
Attributs	
Nom	**Description**
#className : String	Cet attribut renferme le nom de la classe que le Doa gère.
#autoSave : Boolean	Si la valeur de cette attribue vaut « True » alors toutes opération sauvegarde, modification et de suppression a un effet immédiat sur notre ontologie de domaine.

ElementNotExistException	
Type de classe	Classe concrète, exception.
Super classe	Exception.
Description	Cette classe représente un type d'exception, cette dernière peut être déclenché « instancié » lorsque ont essaye de modifier une instance qui n'existe pas.

InformationIncompletException	
Type de classe	Classe concrète, exception.
Super classe	Exception.
Description	Cette classe représente un type d'exception, cette dernière peut être déclenché « **instancié** » lorsque ont essaye de d'ajouter « **create, save** » ou de modifier « **modify**» une instance dont on n'indique pas l'URI ou bien lorsque des informations nécessaires sont manquantes « **les propriétés qui possède une cardinalité >=1 ou bien =1».**

DoaActivity	
Type de classe	Classe concrète.
Super classe	AbstractDoa.
Classes implémentées	Factory< Pro_Activity >
Attributs	**Description**
_-instance : DoaActivity	**Cet objet renferme la référence vers l'instance de l'objet lui-même.**
Méthode	
Nom	**Description**
+create(arg : Pro_Activity)	Cette méthode à pour effet de crée une instance vide de la classe OWL Pro_Activity.
+save(arg : Pro_Activity)	Cette méthode à pour effet de crée une instance complète de la

	classe OWL Pro_Activity.
+modify(arg : Pro_Activity)	Cette méthode à pour effet de modifier une instance de la classe OWL Pro_Activity.
+remove(arg : String)	Cette méthode à pour effet de supprimer une instance référencée par son URI de la classe OWL Pro_Activity.
+find(arg : String)	Cette méthode à pour effet de supprimer une instance référencée par son URI de la classe OWL Pro_Activity.
+getAllReference (arg : String) : String[]	Cette méthode à pour effet d'acquérir tout les référence d'instances de la classe Pro_Activity.
_+ newInstance ( ) : DoaActivity	Cette méthode a le même effet qu'un constructeur, elle est spécialement dédié à l'instanciation des objets de type DoaActivity.

Dans la figure 20 nous avons un petit aperçu de la classe OWL pro_Activity, les liens qui sont en vert indiquent les data types, et ceux qui sont en bleu représentent les object type.

Pour chaque classe OWL qui peut être instanciée j'ai définit son équivalent dans UML de la manière suivante :

1) Chaque data type a été représenté par un String qui indique son contenue.

2) Chaque object type a été représenté par un String qui indique son URI pointé par l'object type.

3) Chaque object et data type possède son propre getter et setter.

4) Chaque object ou data type qui na pas la restriction de cardinalité (cardinalité max <=1) ou bien (cardinalité max =1) a été représenté par une collection de String, plus deux méthode pour ajouter et supprimer un élément du contenue de la propriété.

5) La classe UML hérite obligatoirement de la classe AbstractConcept.

6) Les liens d'héritage ainsi que le type des relations entre relations (subsomption, …) non pas été abordé dans ces travaux, les classes UML se content juste de suivre leurs conjoints OWL dans ce que ces dernières peuvent contenir.

Remarque :

Dans le diagramme de classes précédent et par souci de clarté de ce dernier seul la classe Activity et son Doa respective « DoaActivity » ont été définit, mais dans le cas réel nous avons utilisé plus d'une dizaine de classes OWL dans notre système.

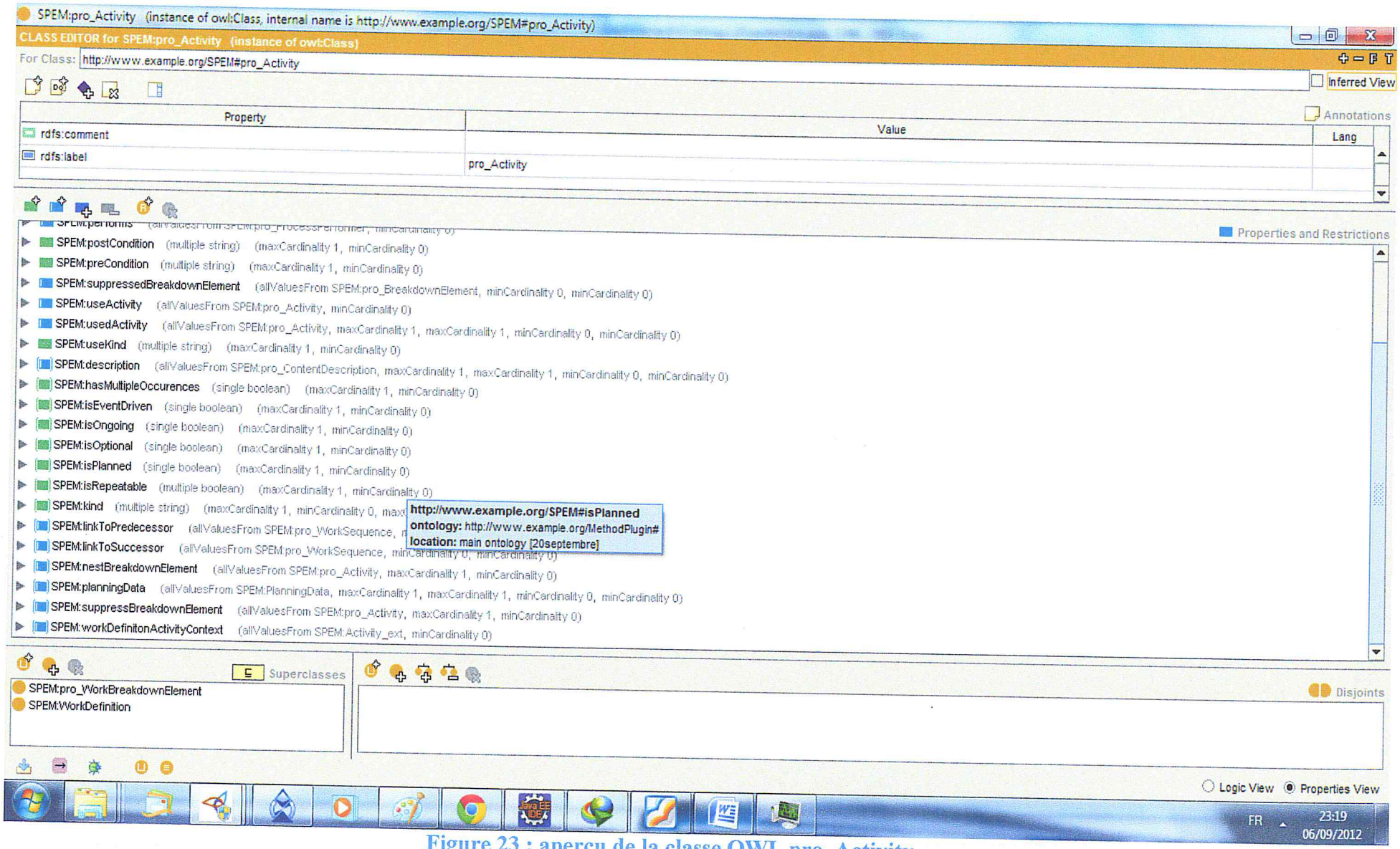

Figure 23 : aperçu de la classe OWL pro_Activity

ServiceSPEM	
Type de classe	Classe concrète.
Super classe	AbstractDoa.
Classes implémentées	Factory< Pro_Activity >
Attributs	**Description**
_-instance : DoaActivity	**Cet attribut renferme la référence vers l'instance de l'objet lui-même.**
-autoSave : Boolean	**Si cet attribut est à « True » alors toute modification sur l'ontologie sera sauvegardée instantanément.**
-connector :OntologyConnector	
-url_ontology : URL	**Cet attribut indique l'URL ou ce trouve l'ontologie.**
Attributs	**Description**
_+ newInstance ( ) : ServiceSPEM	Cette méthode a le même effet qu'un constructeur, elle est spécialement dédiée à l'instanciation des objets du type ServiceSPEM.
+ start ( )	Cette méthode à pour effet de charger notre ontologie en mémoire et d'initialiser nos Doa.
+ close ( )	Lorsque cette méthode est invoquée l'accès à notre ontologie de domaine est coupé.
+ save ( )	Cette méthode oblige l'ontologie à enregistrer les modifications.
+ clear ( )	Cette méthode a pour but de vider notre ontologie de domaine de toutes ces instances, les effets de cette action sont irréversibles.

De plus de ça, la classe ServiceSPEM possède pour chaque classe OWL qui peut être instancié un Doa unique vers elle.

Pour les méthodes, tous les attributs possèdes un getter et un setter exception faite pour l'attribut connector qui ne possède ni l'un ni l'autre.

- **Diagramme de composants:**

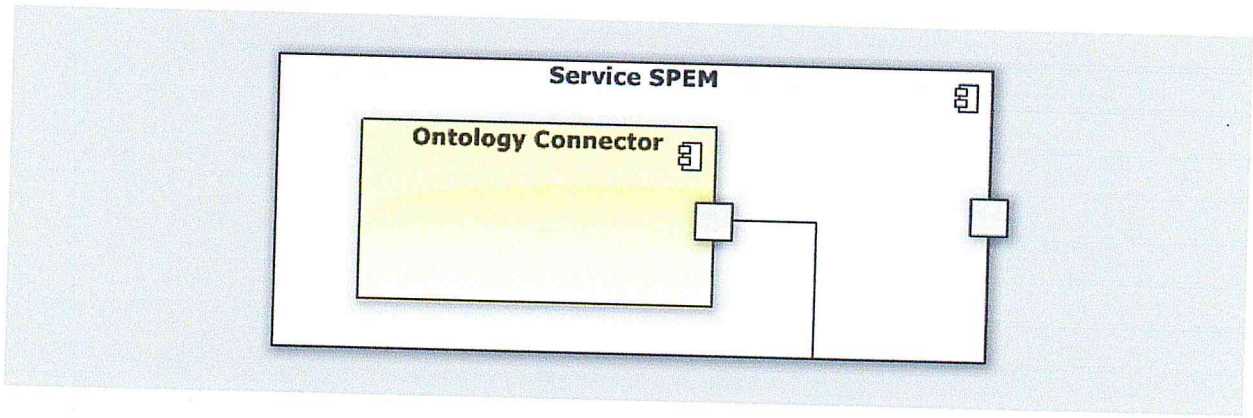

Figure 24 : Diagramme de composants« Service SPEM »

- **Objectifs :**

Ce composant est à la base de notre système, il permet à nos différentes applications d'interagir avec notre ontologie de domaine.

5.1.1 Ontology Connector :

- **Objectifs :**

Ce composant est la brique de base de notre système, grâce à lui ont peut communiquer avec n'importe quel ontologie, il a été optimisé pour être utilisé d'une manière simple et intuitifs.
Ce composant dans ça constitution est un Framework, et grâce à ça nous pourrons adapter notre système à n'importe quel type de technologie sans que cela puisse avoir la moindre influence sur la conception ou l'implémentation de ce dernier.

- **Diagramme de composants :**

Figure 25 : Diagramme de composants« Ontology Connector »

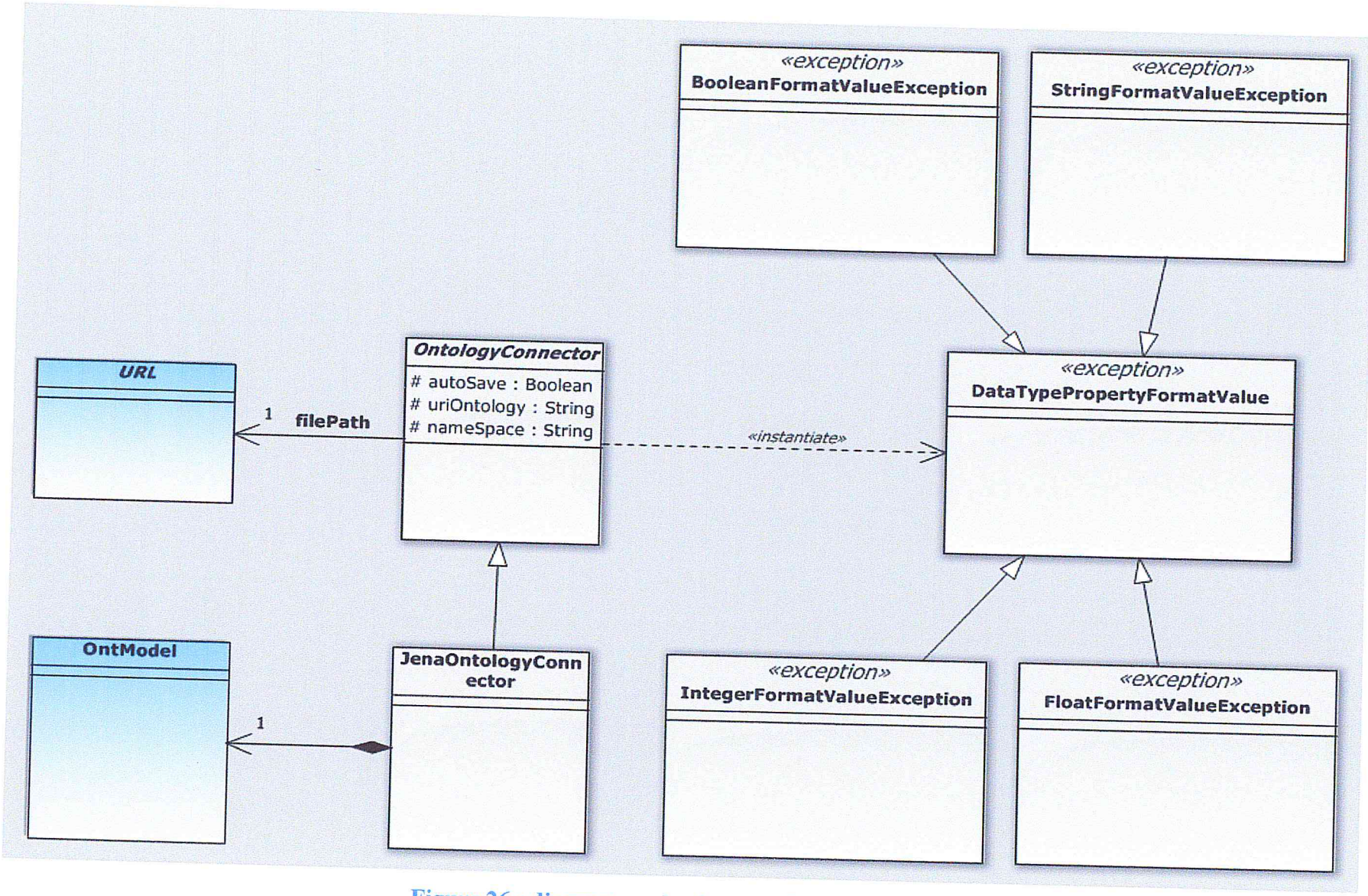

Figure 26 : diagramme de classes « Ontology Connector »

- **Description diagramme de classes :**

DataFormatValueException	
Type de classe	Classe abstraite, exception.
Super classe	Exception.
Description	Cette classe représente un modèle pour les exceptions qui se déclenche lors de l'insertion d'une valeur qui ne correspond pas au type du data type.

IntegerFormatValueException	
Type de classe	Classe concrète, exception.
Super classe	DataFormatValueException.
Description	Cette classe représente un type d'exception, cette dernière peut être déclenché « instancié » lorsque ont essaye d'insérer une valeur qui ne correspond pas data type Integer.

BooleanFormatValueException	
Type de classe	Classe concrète, exception.
Super classe	DataFormatValueException.
Description	Cette classe représente un type d'exception, cette dernière peut être déclenché « instancié » lorsque ont essaye d'insérer une valeur qui ne correspond pas data type Boolean.

FloatFormatValueException	
Type de classe	Classe concrète, exception.
Super classe	DataFormatValueException.
Description	Cette classe représente un type d'exception, cette dernière peut être déclenché « instancié » lorsque ont essaye d'insérer une valeur qui ne correspond pas data type Float.

StringFormatValueException	
Type de classe	Classe concrète, exception.
Super classe	DataFormatValueException.
Description	Cette classe représente un type d'exception, cette dernière peut être déclenché « instancié » lorsque ont essaye d'insérer une valeur qui ne correspond pas data type String.

JenaOntologyConnector	
Type de classe	Classe concrète.
Super classe	OntologyConnector.

Attribut	**Description**
#model : OntModel	Cet attribut renferme le model OWL décrit par l'API Jena.

Méthode	**Description**
+JenaOntologyConnector (uri : String, location : String)	
+JenaOntologyConnector (uri : String, location : URL)	

OntologyConnector	
Type de classe	Classe abstraite.
Super classe	Aucune.

Attribut	**Description**
#autoSave : Boolean	Cet indique si les modifications doivent être prisent en compte immédiatement ou pas.
#uriOntology : String	Cet attribut indique l'URI de l'ontologie, chaque ontologie possède son propre URI.
#namespace : String	Ceci indique l'espace de nommage de l'ontologie, cet attribut nous forte utile pour le référencement des instances de

	l'ontologie.
#filePath : URL	Cet attribut indique l'emplacement physique de notre ontologie.

Méthode	**Description**
	Getters et setters
+setAutoSave(arg : Boolean)	Setter pour l'attribut autoSave.
+getAutoSave() : String	Getter pour l'attribut autoSave.
+setURIOntology(arg : String)	Setter pour l'attribut uriOntology.
+getURIOntology() : String	Getter pour l'attribut uriOntology.
+getFilePath() : URL	Retourne l'URL de l'ontologie.
	Autres
+getURI(think : String)	Retourne l'URI complet d'une classe ou bien d'une instance.
+save( )	Applique les changements sur l'ontologie.
	Les individus
+getNumberOfIndividualByClass (className: String) : Integer	Méthode abstraite, son implémentation permet de récupérer le nombre d'individus d'une classe.
+createIndividual (individualName : String, className : String)	Méthode abstraite, son implémentation permet de crée un d'individus d'une classe.
+getIndividualsOfClass (className: String) : String[ ]	Méthode abstraite, son implémentation permet de récupérer toutes les références d'individus d'une classe.
+getClassOfIndividual (individualName : String) : String	Méthode abstraite, son implémentation permet de récupérer le nom de classe de l'individu.
+removeIndividual (instance : String)	Méthode abstraite, son implémentation permet de récupérer de supprimer un individu.
+clearAllIndividuals()	Méthode abstraite, son implémentation permet de vider l'ontologie de tous ces individus.
+containInstanceClass (className : String , individualName : String) : Boolean	Méthode abstraite, son implémentation retourne vrai si l'individu est un membre existant de la classe.

Les classes	
+getClassesList(): String[ ]	Méthode abstraite, son implémentation permet de récupérer tout les références de classes que l'ontologie contienne.
+getSuperClasses (className : String) : String[ ]	Méthode abstraite, son implémentation permet de récupérer les super classes d'une classe donnée.
+getSubClasses (className : String) : String[ ]	Méthode abstraite, son implémentation permet de récupérer les classes filles d'une classe donnée.
+getHierarchyClasses() : String[ ]	Méthode abstraite, son implémentation permet de récupérer les classes qui se trouvent au sommet de l'hiérarchie.
Les propriétés « objet »	
+getObjectTypePropertiesList() : String []	Son implémentation permet de récupérer tout la liste des propriétés objet que l'ontologie contienne.
+getClassObjectTypePropertiesList (className: String) : String[ ]	Son implémentation permet de récupérer la liste des propriétés objet d'une classe donnée.
+getObjectPropertiesByClassDomain (className : String) : String[ ]	Son implémentation permet de récupérer la liste des propriétés objet par la classe qui les définissent.
+createInstanceObjectProperty (objectPropertyName : String, individualName : String, propertyValue: String)	Son implémentation permet de remplir une propriété de type objet.
+getObjectPropertyDomain (objectPropertyName: String) : String[ ]	Son implémentation permet de récupérer les la liste des domaines d'une propriété objet donnée.
+getObjectPropertyRange (objectPropertyName : String) : String[ ]	Son implémentation permet de récupérer les la liste des rangs d'une propriété objet donnée.
+getObjectPropertyValue (individualName: String , objectPropertyName : String) : String	Méthode abstraite, son implémentation permet de récupérer la valeur d'une propriété objet. Si la propriété en question est multivaluée alors la première valeur sera retournée.

+getObjectPropertyValues (individualName : String , objectPropertyName : String) : String[ ]	Méthode abstraite, son implémentation permet de récupérer la l'ensemble des valeurs d'une propriété objet.
+removeInstanceObjectProperty (individualName : String, propertyValue : String, objectPropertyName : String)	Méthode abstraite, son implémentation permet de supprimer une valeur d'une propriété objet.
+containInstanceObjectProperty (individualName : String, propertyValue : String, objectPropertyName : String) : Boolean	Méthode abstraite, elle retourne vrai si la propriété objet possède la valeur citée en référence
Les propriétés « data »	
+getDataTypePropertiesList() : String []	Son implémentation permet de récupérer tout la liste des propriétés « data » que l'ontologie contienne.
+getClassDataTypePropertiesList (className: String) : String[ ]	Son implémentation permet de récupérer la liste des propriétés « data » d'une classe donnée.
+getDataPropertiesByClassDomain (className : String) : String[ ]	Son implémentation permet de récupérer la liste des propriétés « data » par la classe qui les définissent.
+createInstanceDataProperty (dataPropertyName : String, individualName : String, propertyValue: String)	Son implémentation permet d'attribuer une valeur à une propriété data référencé.
+getDataPropertyDomain (DataPropertyName: String) : String[ ]	Son implémentation permet de récupérer les la liste des domaines d'une propriété data donnée.
+getDataPropertyRange (DataPropertyName : String) : String	Son implémentation permet de récupérer le rang d'une propriété data donnée.

+getDataPropertyValue (individualName: String , dataPropertyName : String) : String	Méthode abstraite, son implémentation permet de récupérer la valeur d'une propriété data. Si la propriété en question est multivaluée alors la première valeur sera retournée.
+getDataPropertyValues (individualName : String , dataPropertyName : String) : String[ ]	Méthode abstraite, son implémentation permet de récupérer la l'ensemble des valeurs d'une propriété data.
+removeInstanceDataProperty (individualName : String, propertyValue : String, DataPropertyName : String)	Méthode abstraite, son implémentation permet de supprimer une valeur d'une propriété data.
+containInstanceDataProperty (individualName : String, propertyValue : String, DataPropertyName : String) : Boolean	Méthode abstraite, elle retourne vrai si la propriété data possède la valeur citée.
Cardinalité	
+minCardinality (objectPropertyName : String) : Integer	Méthode abstraite, son implémentation retourne la cardinalité minimum pour une propriété objet donnée.
+maxCardinality (objectPropertyName : String) : Integer	Méthode abstraite, son implémentation retourne la cardinalité maximum pour une propriété objet donnée.
+cardinalityIsOne (objectPropertyName : String) : Boolean	Méthode abstraite, elle retourne vrai si la cardinalité de la propriété objet vaut un.
Connexion	
+closeConnection()	Méthode abstraite, son implémentation permet de clôturer la connexion avec l'ontologie.
+resetConnection()	Méthode abstraite, son implémentation permet de ré initier la connexion avec l'ontologie.

+changeConnection (uri : String, filePath : URL)	Méthode abstraite, son implémentation permet de se connecter à une autre ontologie.

5.2 Software Library :

Vue que nos applications utilisent des composants dynamique, il s'avérait plus qu'utile de concevoir un outil qui nous permettrai la prise en charge de cette collection de composants.

Pour se faire, ce composant regroupe les différents composants dans un répertoire et les classifie dans deux catégories, les composant autonome et les composants non autonomes ou comme nous les avons appelés « composants exécutables ».

Les composants autonomes constituent une part essentiel du système dans ça globalité, contrairement à cela les exécutables son des composants utilisés par d'autres composants d'une manière dynamique, ils apportent un plus au système mais ils ne constituent pas le système ou une partie de ce dernier.

Chaque composant possède un id, nom, description et une version et de plus les composants exécutables possède un luncher qui indique le nom de la classe qui permet le chargement du composant.

Les composants ne sont pas toujours élémentaires, les composant peuvent s'interagir entre eux, de plus certains composants ont la possibilité d'utiliser des composants qui sont de nature tout à fait différent de notre système, tout ces cas ont été traité dont le moindre petit détaille dans notre conception.

Bien que le composant Software Library utilise des informations pour gérer et organiser notre collection de composants, ce dernier na aucune information et aucune idée sur leurs organisation interne et sur la manière dont ils sont conçue.

- **Diagramme de composants :**

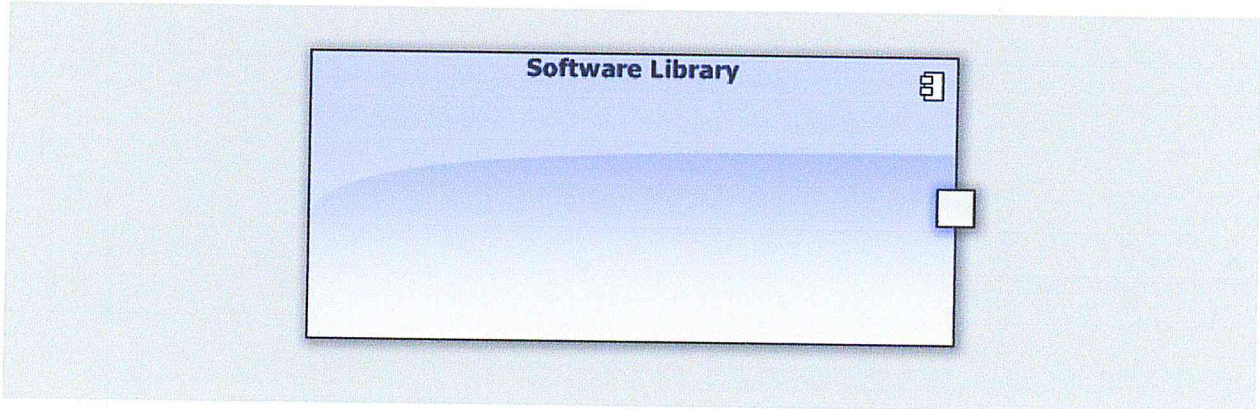

Figure 27 : diagramme de composants « Software Library »

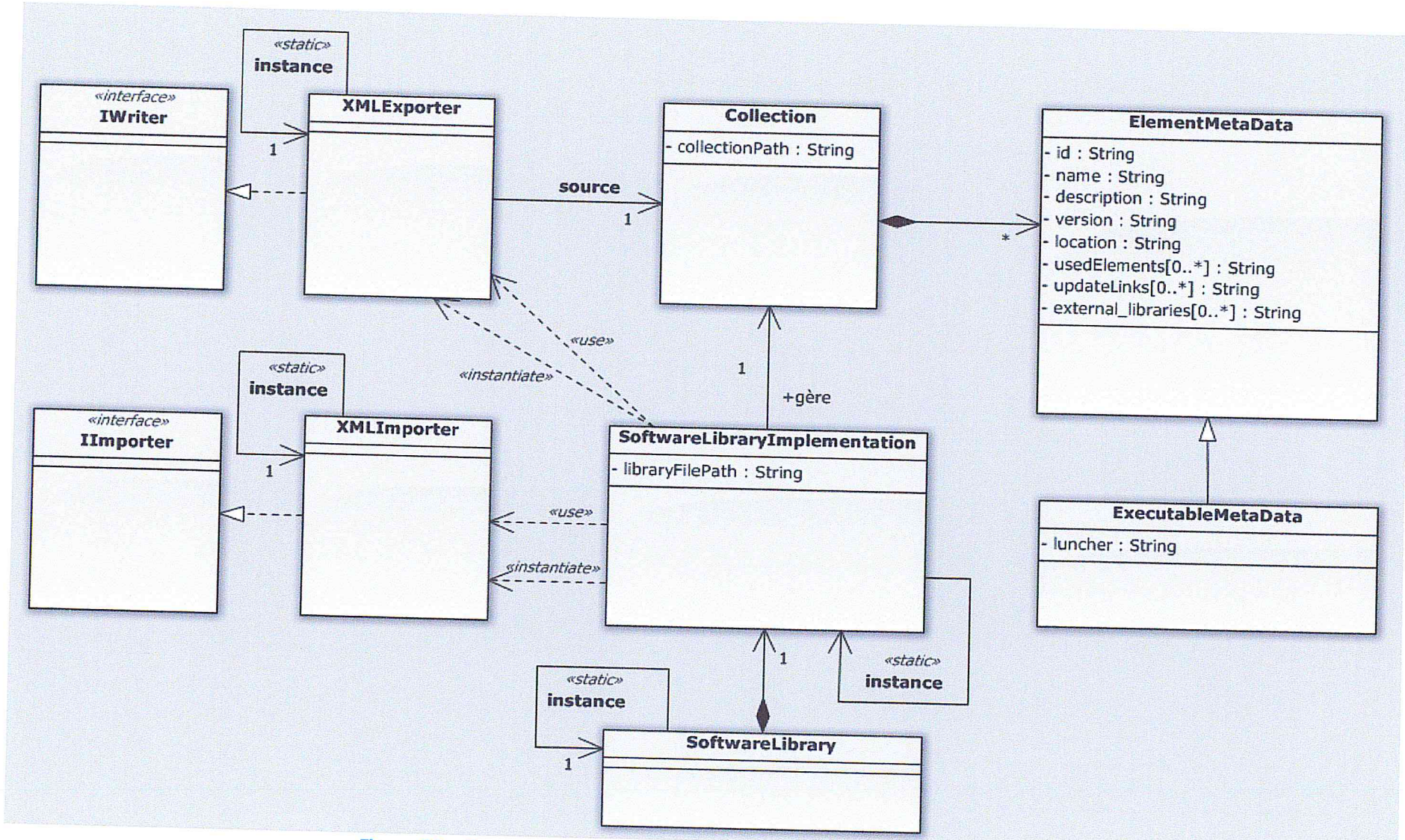

Figure 28 : diagramme de classes pour le composant « software Library »

ElementMetaData	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
-id : String	Cet attribut renferme l'id du nos composant, chaque composant possède un id unique.
-name : String	Le nom du composant.
-description : String	Description du composant.
-version : String	Indique la version du composant.
-location : String	Le nom du fichier qui abrite le composant.
-usedElements : String [ ]	Indique l'ensemble des composant que ce dernier à besoin pour fonctionner.
-updateLinks : String [ ]	Indique les URL des liens de téléchargement des mises à jour.
-external_libraries : String [ ]	Indique l'ensemble éléments « bibliothèque, fichiers, … » nécessaire pour le bon fonctionnement du composant.
Méthode	
Des getters et des setters sur l'ensemble des attributs.	

Collection	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
- luncher : ElementMetaData [ ]	Cet attribut indique la collection des métas data géré par notre librairie.
- collectionPath : String	Cet attribut indique l'emplacement de nos fameux composants.

Méthode	Description
+ getElementsList ( ) : String []	Cette méthode retourne la liste de tout les id, autrement dit la liste de tous les composants que notre bibliothèque abrite.
+ getExecutablesList() : String [ ]	Récupère la liste de tout les exécutable
+getElementMetaData (id: String) : ElementMetaData	Permet de récupérer les métas data d'un composant.
+addElementMetaData (x : ElementMetaData)	Ajout des métas data d'un composant à notre collection.
+removeElementMetaData (id : String)	Permet de supprimer les métas data d'un composant.
+isExecutableMetaData (id : String) : Boolean	Retourne vrai si le composant référencé est un composant exécutable.
+ getCollectionPath() : String	Getter pour l'attribut collectionPath.
+ setCollectionPath(path : String)	Setter pour l'attribut collectionPath.

ExecutableMetaData	
Type de classe	Classe concrète.
Super classe	ElementMetaData.
Attribut	**Description**
- collection : String	Cet attribut indique le nom de la classe qui permet au composant de se charger en mémoire.
Méthode	
Des getters et des setters sur l'ensemble des attributs.	

IWriter	
Type de classe	Classe abstraite, interface.
Super classe	Aucune.
Méthode	**Description**

+ write(filePath : String)	L'implémentation de cette méthode permet l'écriture du fichier en format cible
+ load( )	L'implémentation de cette méthode permet le chargement des données dans le format cible.

XMLExporter	
Type de classe	Classe concrète.
Super classe	Aucune.
Classes implémentées	IWriter
Attribut	**Description**
-source : Collection	
_- instance : XMLExporter	
Méthode	**Description**
+ write(filePath : String)	
+ load( )	
+ setCollection(collection : Collection)	
_+ newInstance : XMLExporter	

IImporter	
Type de classe	Classe abstraite, interface.
Super classe	Aucune.
Méthode	**Description**
+ load(String filePath) : Collection	L'implémentation de cette méthode permet charger une collection depuis un fichier.

XMLImporter	
Type de classe	Classe concrète.
Super classe	Aucune.
Classes implémentées	IImporter

Attribut	**Description**
_- instance : XMLExporter	
Méthode	**Description**
+ load(String filePath) : Collection	
_+ newInstance : XMLExporter	

SoftwareLibraryImplementation	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
-collection : Collection	Cet attribut indique la collection géré par notre bibliothèque.
- libraryFilePath : String	Cet attribut indique l'emplacement du fichier ou toutes nos données sont sauvegardées.
_- instance : SoftwareLibraryImplementation	
Méthode	**Description**
_+ newInstance : SoftwareLibraryImplementation	
setLocation(location : String)	
getLocation() : String	
+getFonctionalPath(id : String) : String	Retourne l'emplacement du composant.
+getInstalledElements() : String [ ]	Retourne la liste des références des composants présents dans la bibliothèque.
+addElement(ElementMetaData x)	Permet d'ajouter des métas data d'un composant à librairie.
+removeElement(id : String)	Supprime les métas data d'un composant référencé.

+getElement(id : String) : ElementMetaData	Retourne les métas data d'un composant référencé.

SoftwareLibraryImplementation	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
-implementation : SoftwareLibraryImplementation	
_- instance : SoftwareLibraryImplementation	
Méthode	**Description**
_+ newInstance( ) : SoftwareLibrary	
setLocation(location : String)	
setLocation() : String	
+getFonctionalPath(id : String) : String	Retourne l'emplacement du composant.
+getInstalledElements() : String [ ]	Retourne la liste des références des composants présents dans la bibliothèque.
+addElement(ElementMetaData x)	Permet d'ajouter des métas data d'un composant à librairie.
+removeElement(id : String)	Supprime les métas data d'un composant référencé.
+getElement(id : String) : ElementMetaData	Retourne les métas data d'un composant référencé.

5.3 Instantiation application :

- **Diagramme de composants:**

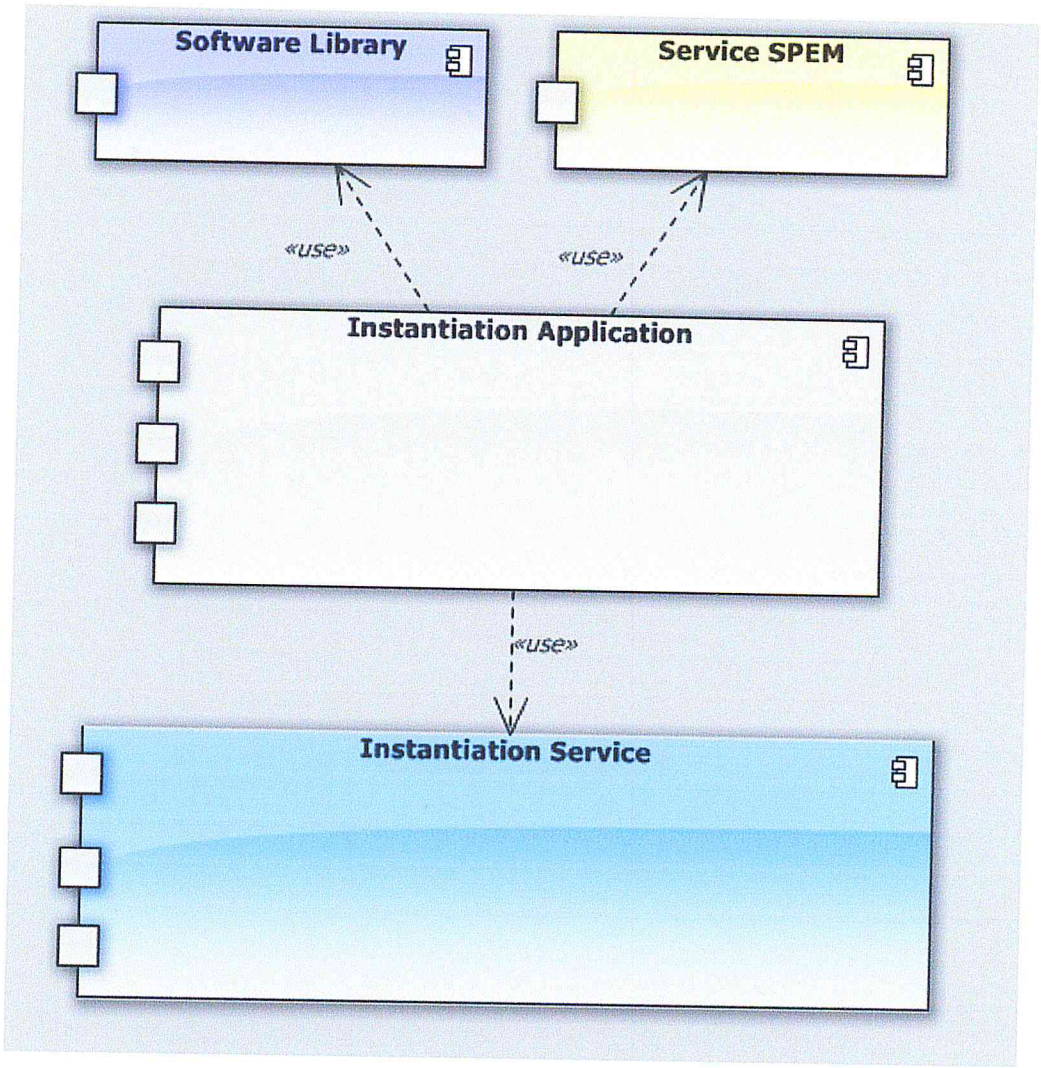

Figure 29 : Diagramme de composants« Instantiation Application »

- **Diagramme de classes :**

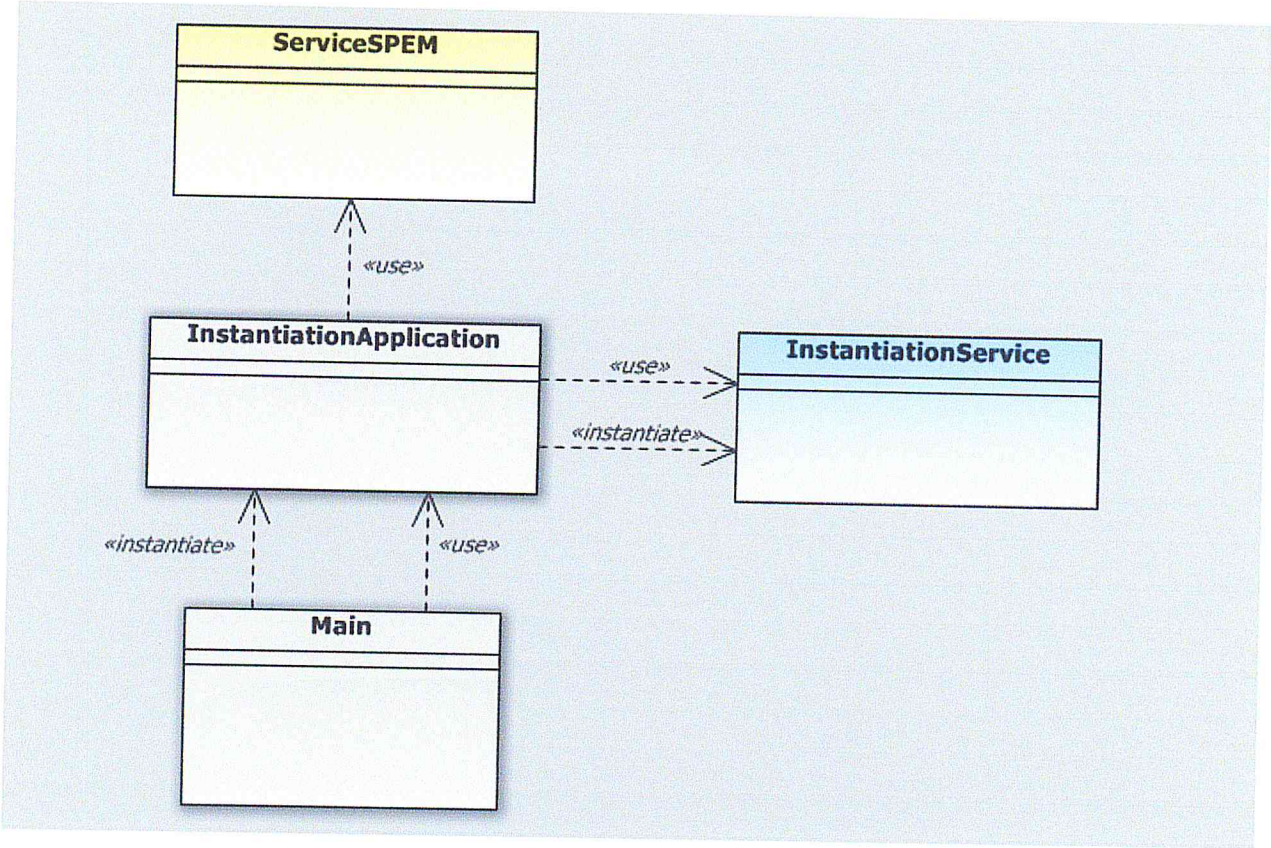

Figure 30 : diagramme de classes « Instantiation Application »

- **Description diagramme de classes :**

InstantiationApplication	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
_- library : SoftwareLibrary	
_- service : ServiceSPEM	
_- instance : InstantiationApplication	
Méthode	**Description**
_+ newInstance( ) : InstantiationApplication	
_+ setServiceSPEM(service : ServiceSPEM)	
_+ getServiceSPEM( ) : ServiceSPEM	

_+ setSoftwareLibrary(library : SoftwareLibrary)	
_+ getSoftwareLibrary( ) : SoftwareLibrary	
+ getIDList () : String [ ]	
+ getMetaData (id : String) : ElementMetaData	
+ installe(filePath : String) : String	
+ uninstalle(id : String)	
+ instanciate(filePath : String) : String	

Main	
Type de classe	Classe concrète.
Super classe	Aucune.
Méthode	**Description**
_+ main(arg : String[])	Cette méthode configure l'application de l'instanciation d'une manière générale puis elle la lance.

5.3.1 Service Instantiation :

Le composant que nous allons étudier maintenant consiste dans son intégralité le module qui nous permettra d'exécuter le processus d'instanciation.

Ce composant définit 4 sous composants, « Instantiation Executioner » pour le processus d'instanciation, « Manager » pour la gestion du composant, « Instantiator Library » pour la gestion de librairie des instantiateurs et « Provider » pour l'acquisition de différentes informations depuis l'extérieur.

Les sous composants du composant « Instantiation Service » ont été construit d'une manière statique ce qui signifie que leur implémentation est contenue dans le plus grand composant, exception faite pour le composant Instantiator Library.

- **Diagramme de composants :**

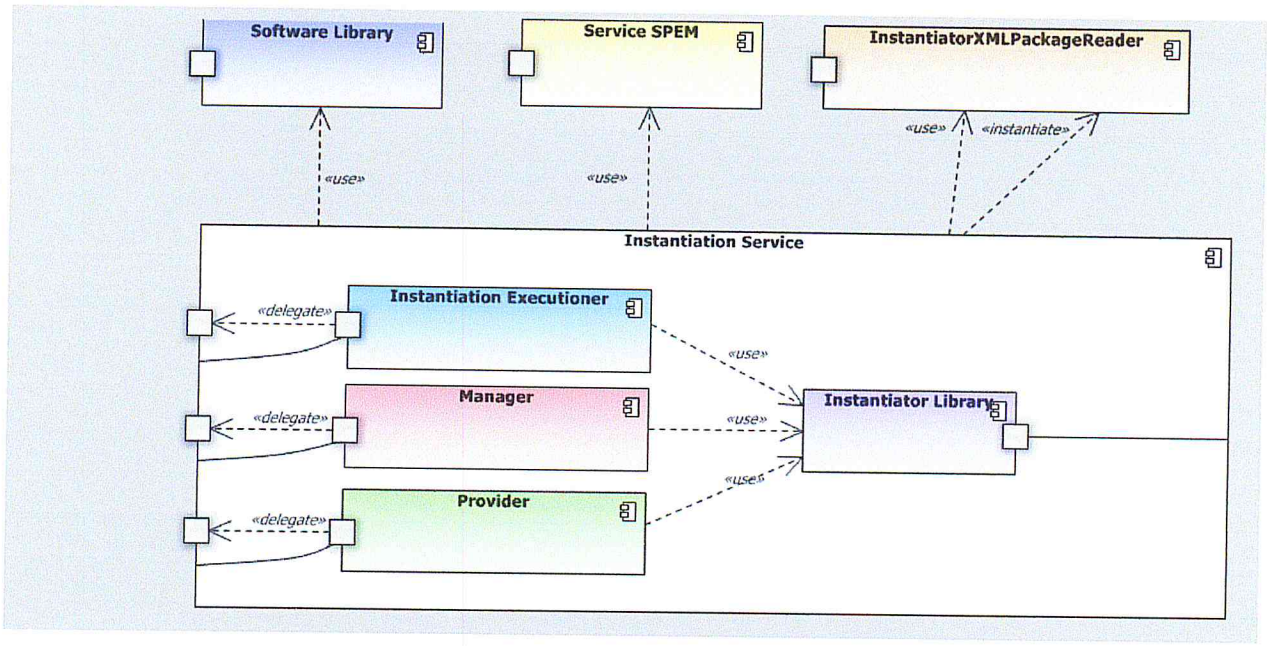

Figure 31 : Diagramme de composants« Instantiation Service »

- **Diagramme de classes :**

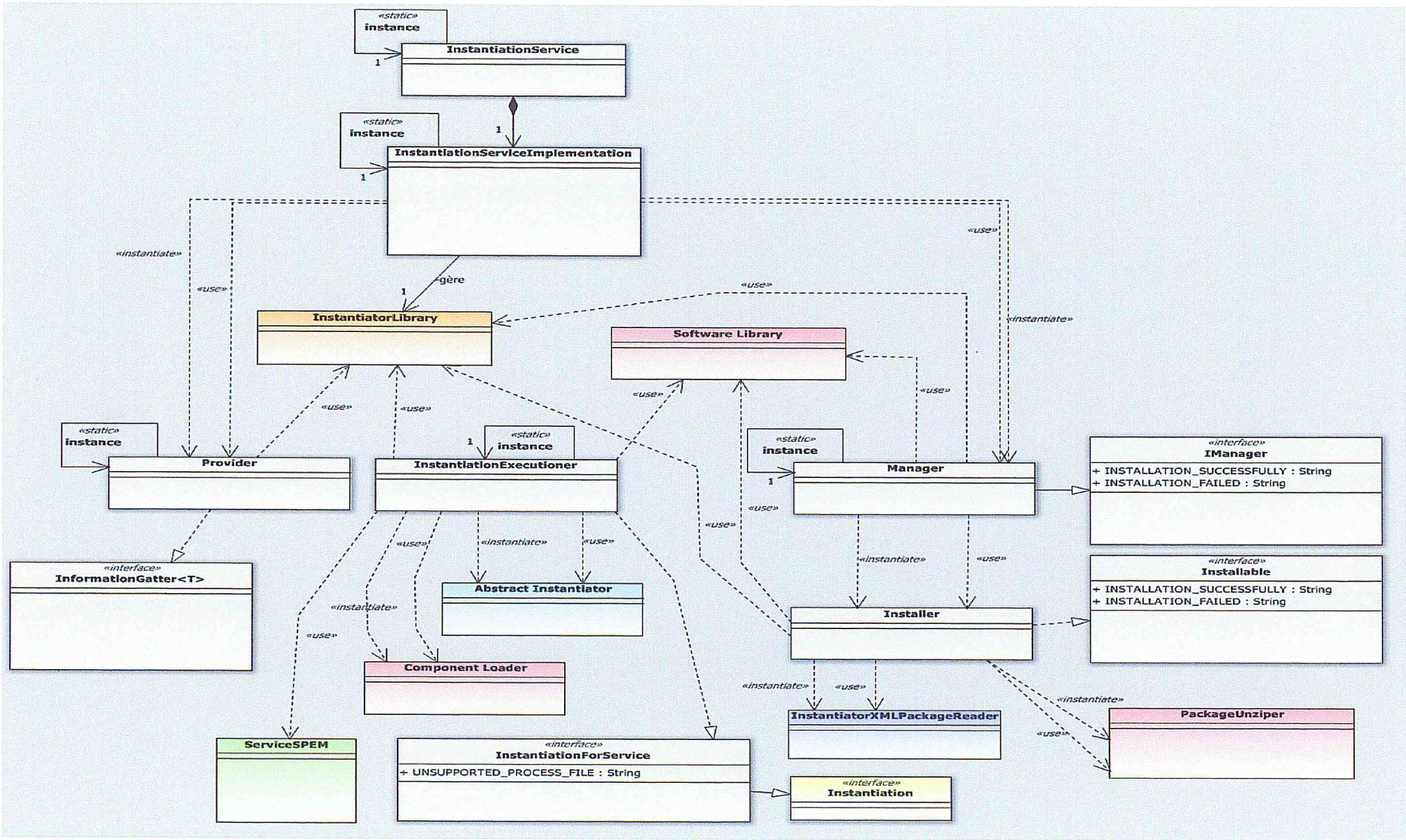

Figure 32 : diagramme de classes « Instantiation Service »

- **Description diagramme de classes :**

InstantiationService	
Type de classe	Classe concrète.
Super classe	Aucune.
Interfaces implémentées	InstantiationForService, Installable, InformationGatter<InstantiatorMetaData>
Attribut	**Description**
_-implementation : InstantiationServiceImplementation	
_- instance : InstantiationApplication	
Méthode	**Description**
_+ newInstance( ) : InstantiationApplication	
+setServiceSPEM(service : ServiceSPEM)	
+ setSoftwareLibrary(library : SoftwareLibrary)	
+ getIDList () : String [ ]	
+ getMetaData (id : String) : ElementMetaData	
+ installe(filePath : String) : String	
+ uninstalle(id : String)	
+ manage_extensions (idInstantiator : String , extensions: String)	
+ instanciate(filePath : String) : String	

InstantiationServiceImplementation	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
- library : SoftwareLibrary	

- service : ServiceSPEM	
_- instance : InstantiationServiceImplementation	
Méthode	**Description**
_+ newInstance() : InstantiationServiceImplementation	
+ setServiceSPEM(service : ServiceSPEM)	
+ setSoftwareLibrary(library : SoftwareLibrary)	
+ getIDList () : String [ ]	
+ getMetaData (id : String) : ElementMetaData	
+ installe(filePath : String) : String	
+ uninstalle(id : String)	
+ manage_extensions (idInstantiator : String , extensions: String)	
+ instanciate(filePath : String) : String	

Installable	
Type de classe	Classe abstraite, interface
Super classe	Aucune.
Attribut	**Description**
+INSTALLATION_SUCCESSFULLY : String = "Installation successfully"	Indique l'état de succès de l'installation.
+ INSTALLATION_FAILED : String = "Installation failed"	Indique l'état d'échec de l'installation.
Méthode	**Description**
+ installe(filePath : String) : String	Son implémentation permettra l'installation d'un système.
+ uninstalle(id : String)	Son implémentation permettra la désinstallation d'un système.

Installer	
Type de classe	Classe concrète.
Super classe	Aucune.
Interfaces implémentées	Installable
Attribut	**Description**
-library : SoftwareLibrary	
-instantiatorLibrary : InstantiatorLibrary	
- instance : Installer	
Méthode	**Description**
_+ newInstance( ) : Installer	
+setInstantiatorLibrary (library : InstantiatorLibrary)	
+getInstantiatorLibrary ( ) : InstantiatorLibrary	
+ setSoftwareLibrary(library : SoftwareLibrary)	
+ getSoftwareLibrary( ) : SoftwareLibrary	
+ uninstalle(id : String)	
+ installe (filePath : String) : String	

IManage	
Type de classe	Classe abstraite, interface
Super classe	Aucune.
Attribut	**Description**
+INSTALLATION_SUCCESSFULLY : String = "Installation successfully"	Indique l'état de succès de l'installation.
+ INSTALLATION_FAILED : String = "Installation failed"	Indique l'état d'échec de l'installation.

Méthode	Description
+ installe(filePath : String) : String	Son implémentation permettra l'installation d'un instantiateur.
+ uninstalle(id : String)	Son implémentation permettra la désinstallation d'un instantiateur.
+ manage_extensions (idInstantiator : String , extensions: String)	Son implémentation permet d'affecter une extension à un instantiateur.

Manager	
Type de classe	Classe concrète.
Super classe	Aucune.
Interfaces implémentées	IManage
Attribut	**Description**
-library : SoftwareLibrary	
-instantiatorLibrary : InstantiatorLibrary	
- instance : Manager	
Méthode	**Description**
_+ newInstance( ) : Installer	
+setInstantiatorLibrary (library : InstantiatorLibrary)	
+getInstantiatorLibrary ( ) : InstantiatorLibrary	
+ setSoftwareLibrary(library : SoftwareLibrary)	
+ getSoftwareLibrary( ) : SoftwareLibrary	
+ uninstalle(id : String)	
+ installe (filePath : String) : String	
+ manage_extensions (idInstantiator : String , extensions: String)	

InformationGatter<T>	
Type de classe	Classe abstraite, interface générique
Super classe	Aucune.
Méthode	**Description**
+ getIDList() : String [ ]	Permet de récupérer la liste de toutes les références des métas data.
+ getMetaData(id : String) : T	Permet de récupérer un méta data particulier.

Provider	
Type de classe	Classe concrète.
Super classe	Aucune.
Interfaces implémentées	InformationGatter <InstantiatorMetaData>
Attribut	**Description**
-instantiatorLibrary : InstantiatorLibrary	
- instance : Provider	
Méthode	**Description**
_+ newInstance( ) : Installer	
+set InstantiatorLibrary (library : InstantiatorLibrary)	
+get InstantiatorLibrary ( ) : InstantiatorLibrary	
+ getIDList() : String [ ]	
+ getMetaData(id : String) : InstantiatorMetaData	

InstantiationExecutioner	
Type de classe	Classe concrète.
Super classe	Aucune.
Interfaces implémentés	InstantiationForService

Attribut	Description
- library : SoftwareLibrary	
- instantiatorLibrary : InstantiatorLibrary	
- service : ServiceSPEM	
_- instance : InstantiationExecutioner	
Méthode	**Description**
_+ newInstance( ) : InstantiationExecutioner	
+ setInstantiatorLibrary (library : InstantiatorLibrary)	
+ getInstantiatorLibrary ( ) : InstantiatorLibrary	
+ setSoftwareLibrary(library : SoftwareLibrary)	
+ getSoftwareLibrary( ) : SoftwareLibrary	
+ setServiceSPEM(service : ServiceSPEM)	
+ instanciate(filePath : String) : String	

5.3.2 Composant Manager :

Ce composant à été crée dans le but de gérer nos différents instantiateurs, pour cela se composant dispose d'une panoplie de méthodes et d'outils qu'ils vont l'aider à le faire.

Ce composant permet donc d'installer, de désinstaller des instantiateurs, il permet aussi la gestion de ces derniers en offrant à l'utilisateur la possibilité de changer l'étendue du domaine d'extension PL. Le domaine d'extension PL définit l'ensemble des extensions qu'un instantiateur est apte pour travailler avec, chaque extension réfère un méta modèle de procédé logiciel unique.

Connaitre le domaine d'extensions PL associé à chaque instantiateur est d'une importance vitale, car cela va nous permettre de choisir un instantiateur par défaut à charger lors de l'étape de l'instanciation.

- **Diagramme de composants :**

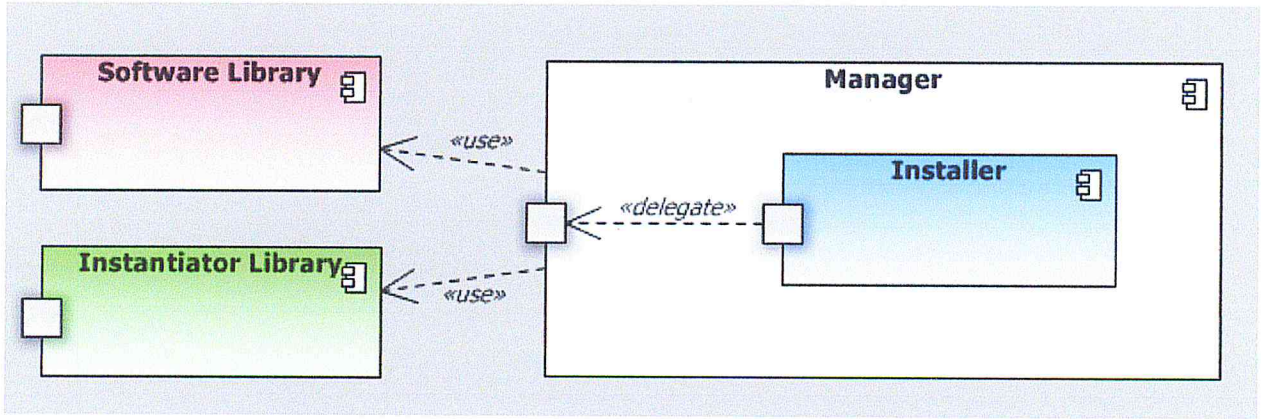

Figure 33 : diagramme de composants « Manager »

- **Diagramme de classes :**

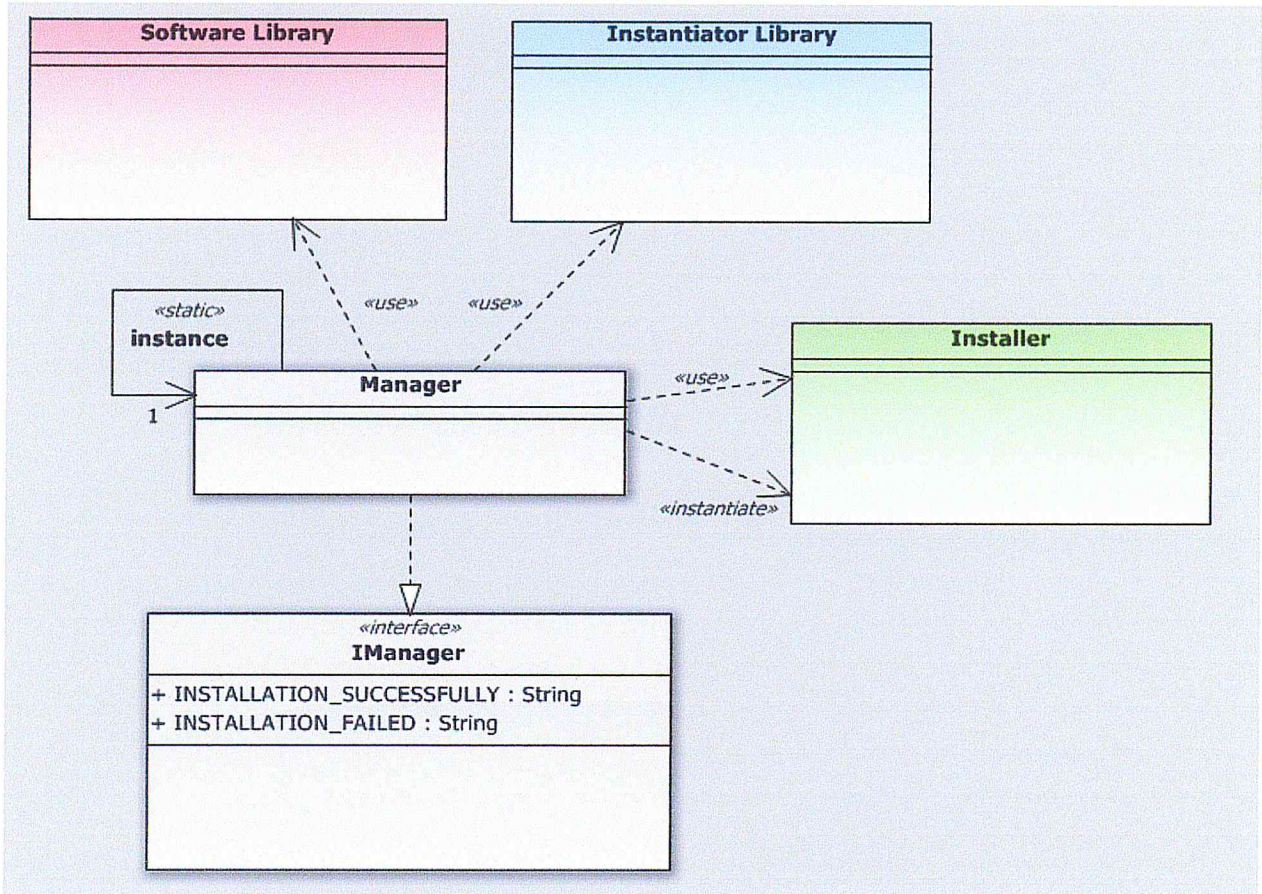

Figure 34 : diagramme de classes

- **Diagramme de composants :**

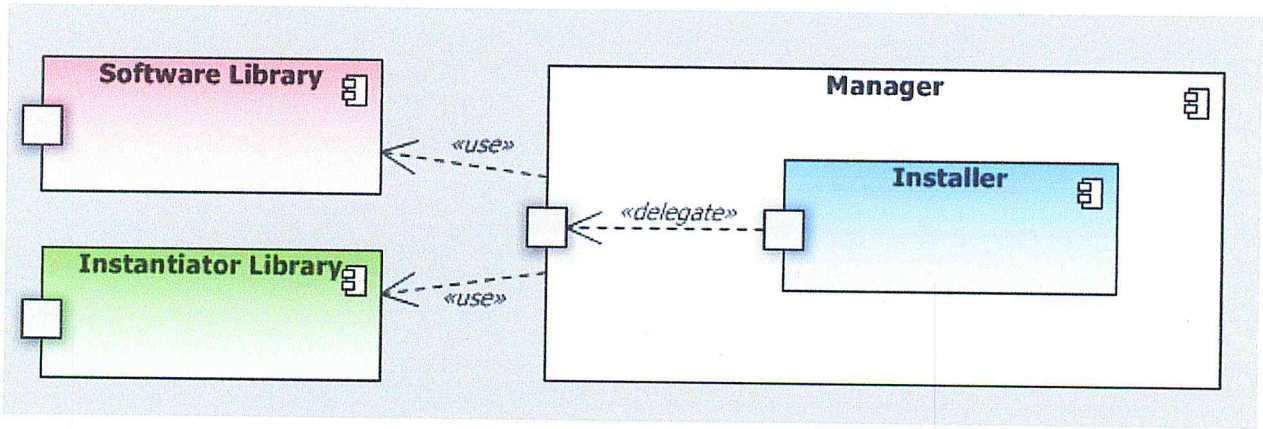

Figure 33 : diagramme de composants « Manager »

- **Diagramme de classes :**

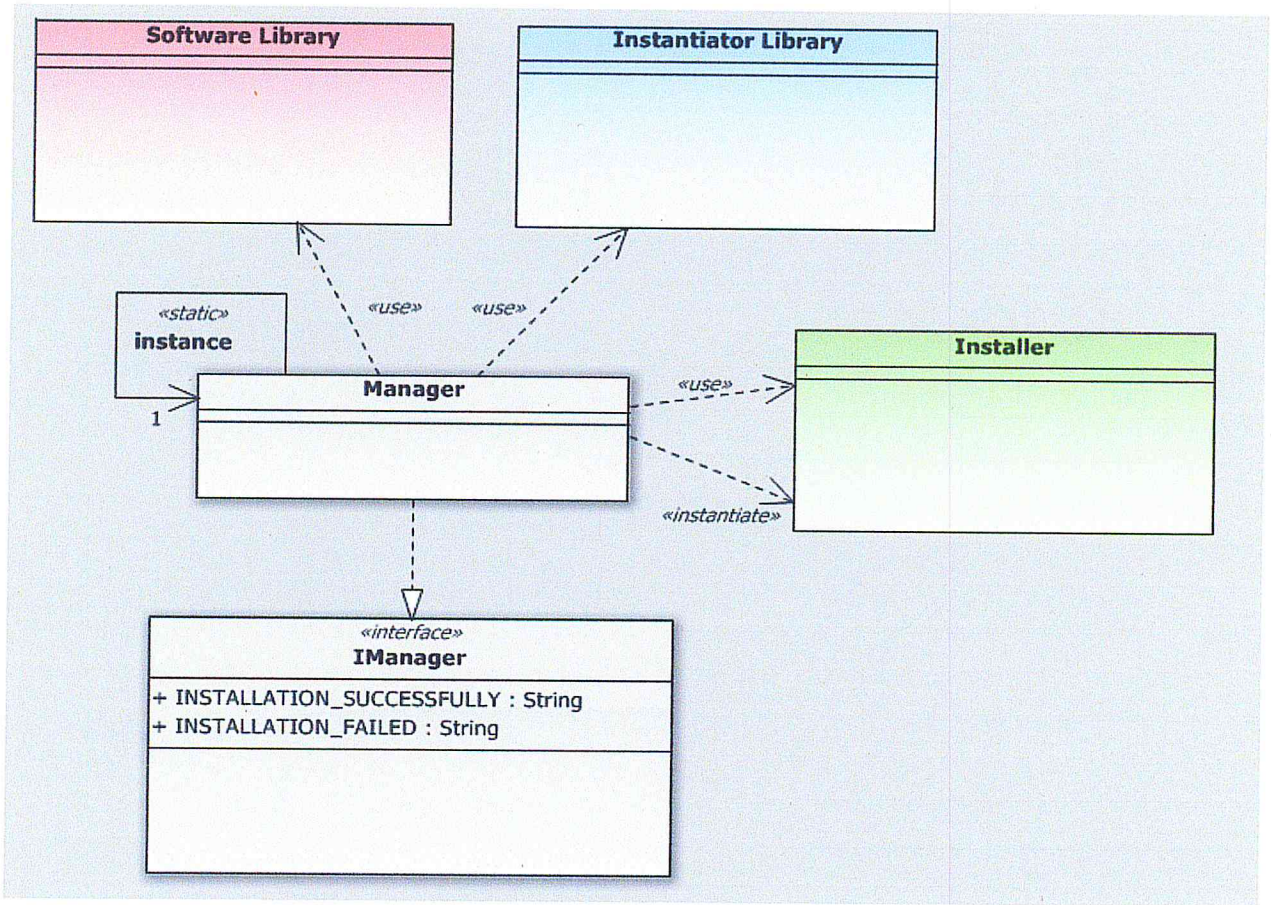

Figure 34 : diagramme de classes

5.3.3 Le composant Installer :

Ce composant est fait partie intégrante du composant « Manager », comme son nom l'indique ça tache principale consiste à installer et désinstaller des instantiateurs.

Dans cette section nous allons aborder en plus en détail les étapes de l'installation, mais tout juste avant ça nous allons étudier nos fichiers d'installation et la façon dont ils sont fabriqués plus en détaille.

Un fichier d'installation est structuré physiquement par un ensemble de fichiers organisés dans un ensemble de dossier, pour faciliter la tâche et pour plus de cohérence nous avons accompagné l'ensemble avec un fichier supplémentaire ce fichier n'est rien d'autre que le fichier descripteur.

Le fichier descripteur contient tout les métas données concernant l'élément qui va être installé comme son nom, ça version, son id, ça description, … etc.

Finalement l'ensemble des dossiers et des fichiers vont être compressés en un fichier Zip pour fournir la forme final de notre fichier d'installation.

L'installation consiste à copier le contenue du fichier de l'installation vers notre bibliothèque logiciel et de mettre à jour nos librairies, pour cela et pour une installation cohérence nous devons choisir l'algorithme suivant :

1) Décompresser le fichier de l'installation dans un emplacement temporaire.
2) Lire le fichier descripteur.

Si la version de l'ontologie est la même version que l'instantiateur utilise alors :

3) Copier tout les dossiers et les fichiers dans notre bibliothèque de logiciels.
4) Mettre à jours les informations de la librairie logicielles et celle des instantiateurs.

Et pour terminer :

5) Supprimer tout les fichiers et dossier temporaires.
6) Retourner un message indiquant l'état de l'installation.

L'installation peut se solder soit par état de succès soit par état un d'échec.

- **Diagramme de composants:**

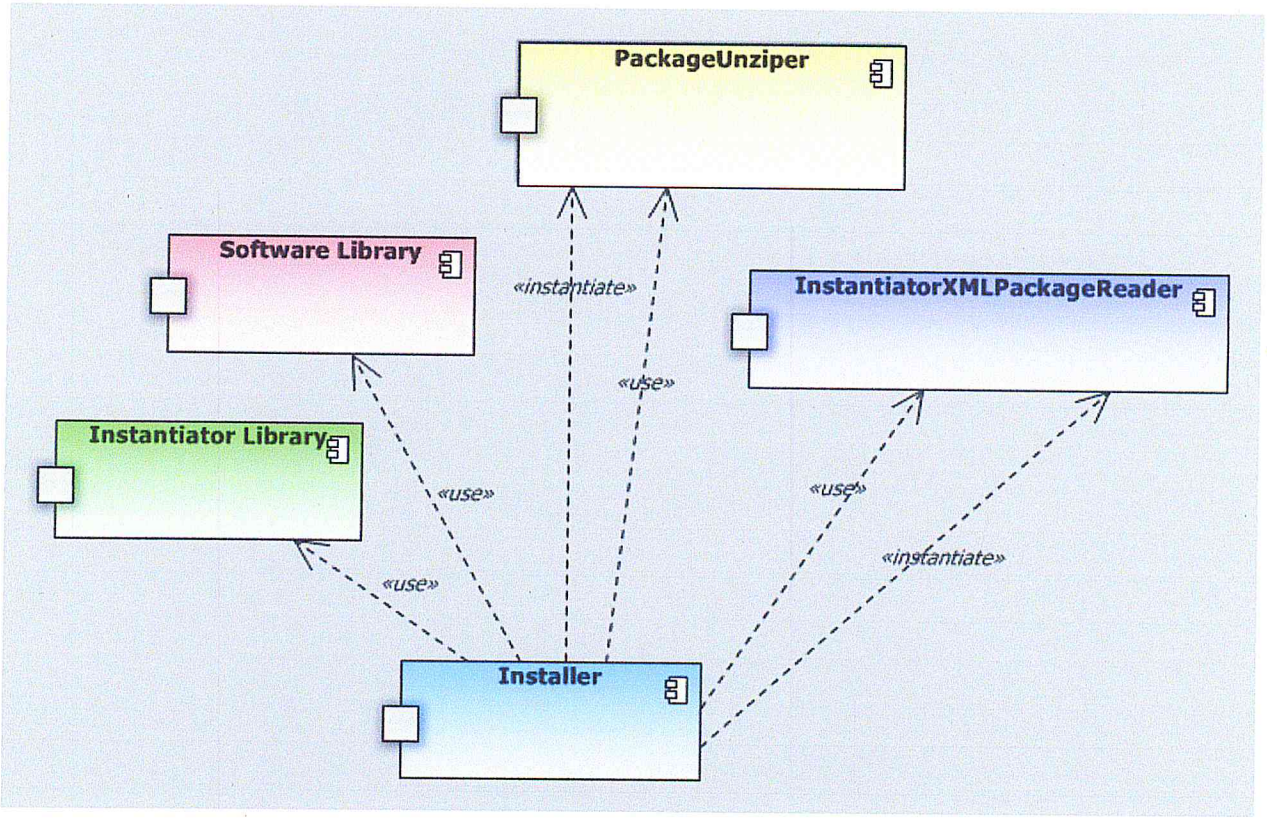

Figure 35 : diagramme de composants « Installer »

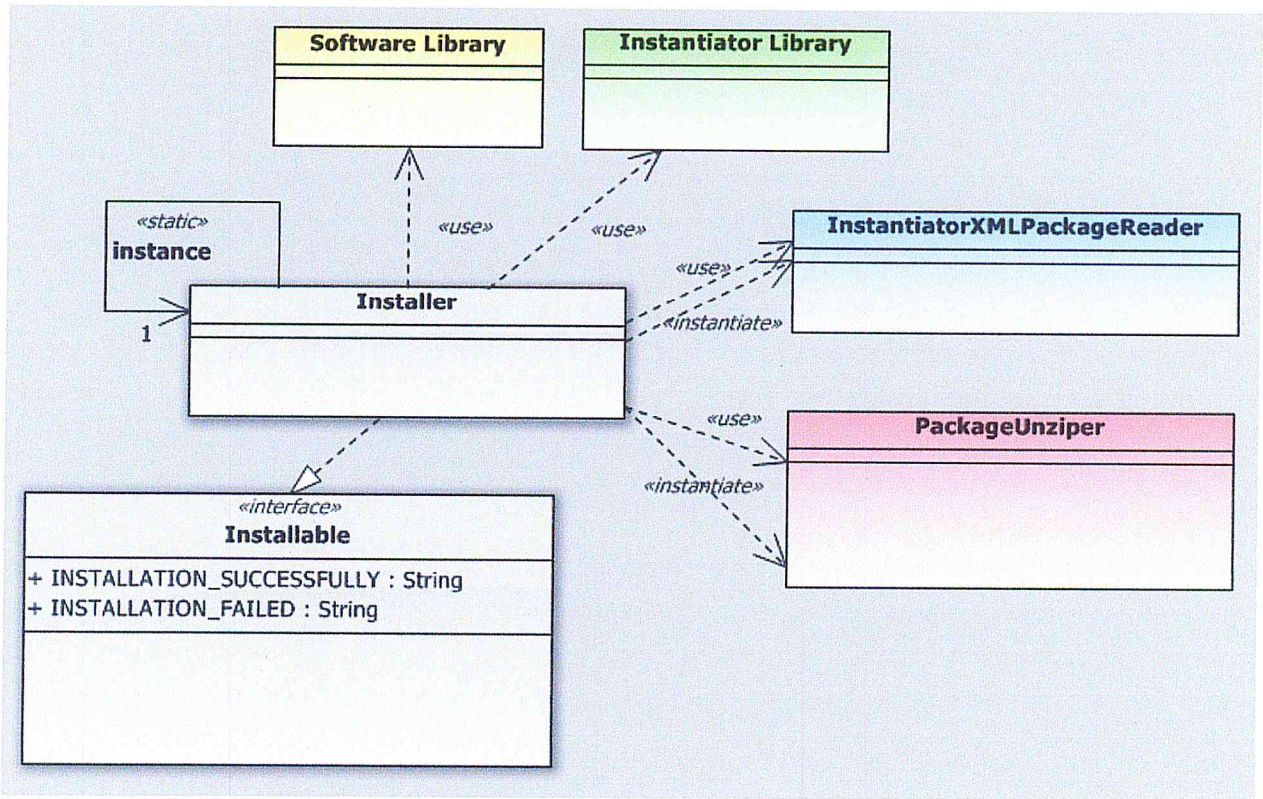

Figure 36 : diagramme de classes « Installer »

5.3.4 Package Unziper :

Ce composant comme Service SPEM est un Framework, son utilité est de permettre la décompression des fichiers, pour ça spécification ont a choisie le format zip qui est un format standard et gratuit.

- **Diagramme de composants:**

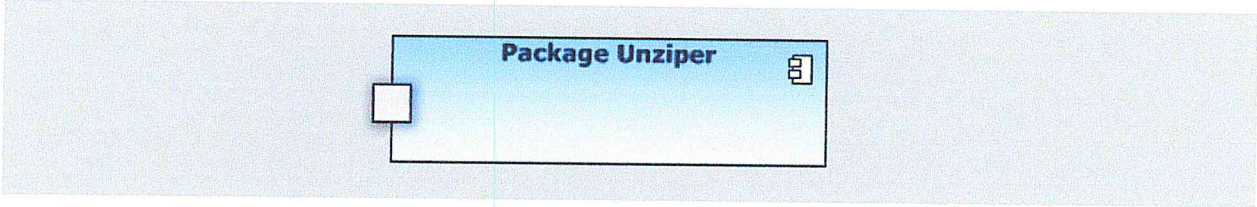

Figure 37 : diagramme de composants « Package Unziper »

- **Diagramme de classes :**

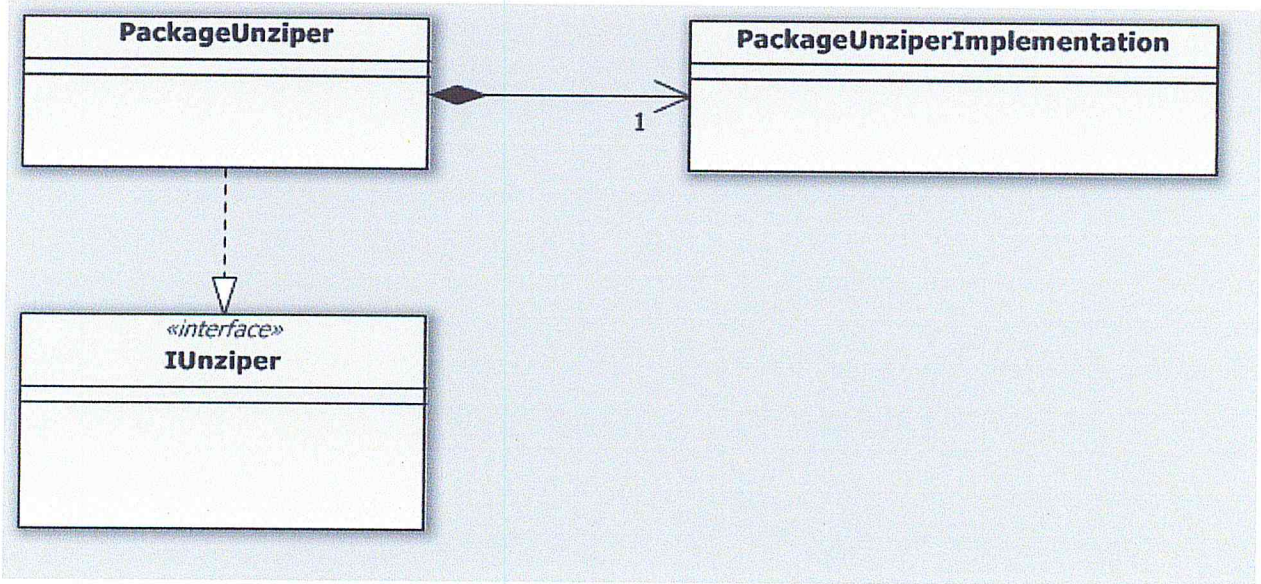

Figure 38 : diagramme de classes « Package Unziper »

- **Description diagramme de classes :**

IUnziper	
Type de classe	Classe abstraite, interface
Super classe	Aucune.
Méthode	**Description**
+ unzip(zipFilePath : String)	Son implémentation permettra de décompresser le archive dans le même emplacement ou que ce dernier.
+ unzip(zipFilePath : String, directory : String)	Son implémentation permettra de décompresser un archive dans un répertoire.

PackageUnziper	
Type de classe	Classe concrète.
Super classe	Aucune.
Interfaces implémentées	IUnziper
Attribut	**Description**
- implementation : PackageUnziperImplementation	
Méthode	**Description**
+ unzip(zipFilePath : String)	
+ unzip(zipFilePath : String, directory : String)	

PackageUnziperImplementation	
Type de classe	Classe concrète.
Super classe	Aucune.
Méthode	**Description**
+ unzip(zipFilePath : String)	
+ unzip(zipFilePath : String, directory : String)	

5.3.5 InstantiatorPackageReader :

Ce composant est destiné à lire le fichier descripteur des fichiers d'installation, le fichier descripteur que nous avons concocté est comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<Element        id="instantiator.pbool"
                name="PBOOL Instantiator"
                description="used to instantiat PBOOL software process in SPEM meta-model"
                luncher="pbool_instantiator.luncher.PBOOLInstantiator"
                fileName="PBOOL Instantiator.jar"
                version="1.02">

    <supported_extensions>
        <extension>pbool</extension>
    </supported_extensions>

    <external_libraries>
        <lib>lib/PBOOL Instantiator Engine.jar</lib>
        <lib>lib/PBOOL_Model.jar</lib>
        <lib>lib/PBOOL_Reader.jar</lib>
        <lib>lib/PBOOL_Source.jar</lib>
    </external_libraries>

    <used_element>
        <element id="service_spem" version="09.20.2011" />
    </used_element>

    <update_sites>
        <site>www.university-x.com/?level=master/?year=2011/?update="reuse_software_process"</site>
    </update_sites>
</Element>
```

Figure 39 : exemple fichier descripteur

Comme nous le pouvons le voir il fichier descripteur contient différents types d'information, chaque un d'eux décrit d'une manière précisent l'élément à installer.

- **Diagramme de composants:**

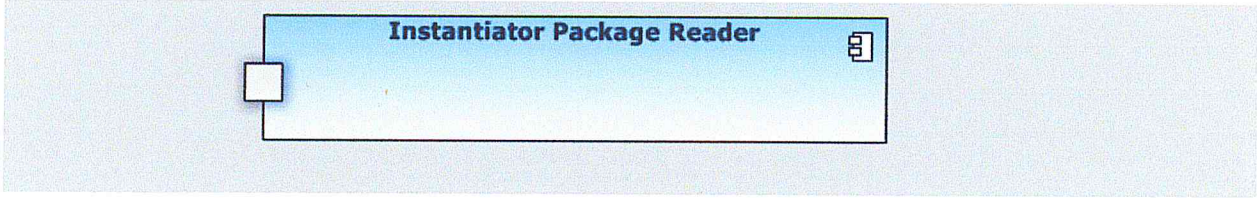

Figure 40 : diagramme de composants : « Instantiator Package Reader »

- **Diagramme de classes :**

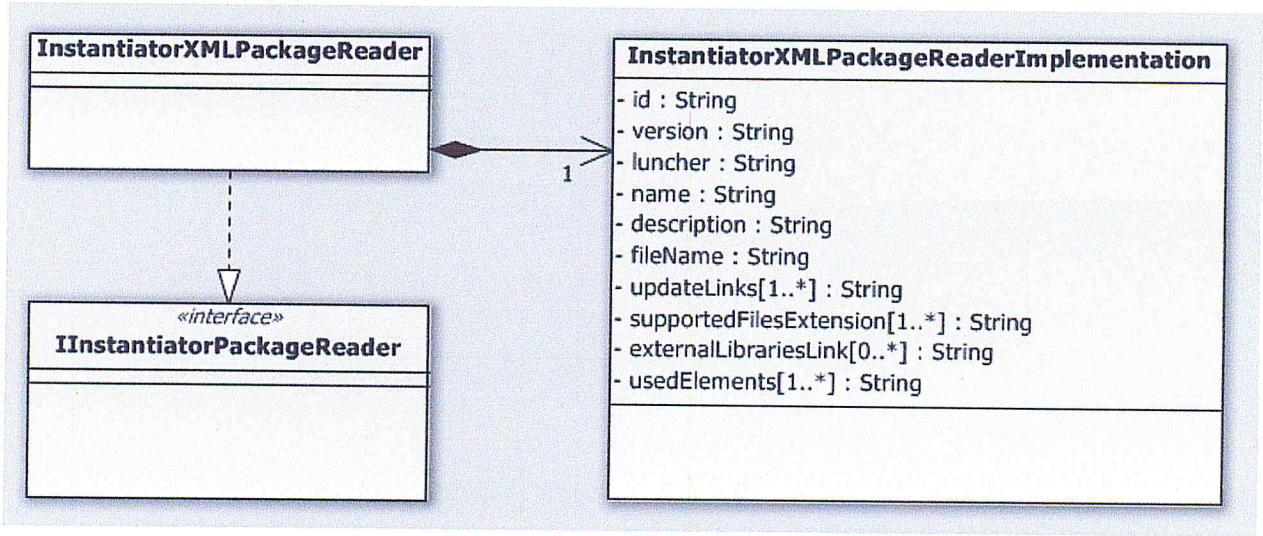

Figure 41 : diagramme de classes « Instantiator Package Reader »

- **Description diagramme de classes :**

IInstantiatorPackageReader	
Type de classe	Classe abstraite, interface
Super classe	Aucune.
Méthode	**Description**
+ getId ( ) : String	
+ getVersion ( ) : String	
+ getLuncher ( ) : String	
+ getName ( ) : String	
+ getDescription ( ) : String	
+ getFileName ( ) : String	
+ getUsedElements ( ) : String [ ]	
+ getUpdateLinks ( ) : String [ ]	
+ getSupportedFilesExtension ( ) : String [ ]	
+ getExternalLibrariesLink ( ) : String [ ]	

+ read (filePath : String)	Son implémentation permet de charger le contenue du fichier descripteur.

InstantiatorXMLPackageReaderImplementation	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
- id : String	Cet attribut indique l'id de l'instantiateur, chaque composant comporte un id unique.
- version : String	Cet attribut indique la version de l'instantiateur.
- luncher : String	Cet attribut indique la classe qui permet le chargement du composant en mémoire.
- name : String	Cet attribut indique le nom de l'instantiateur.
- description : String	Cet attribut indique la description de l'instantiateur.
- fileName : String	Cet attribut indique le nom du fichier qui contient l'implémentation de l'instantiateur.
- updateLinks : String [ ]	Cet attribut indique les différents liens de mises à jour.
- supportedFilesExtension : String [ ]	Cet attribut indique les différentes extensions que l'instantiateur gère.
-externalLibrariesLink : String [ ]	Cet attribut indique les différentes bibliothèques externes que l'instantiateur utilise.
- usedElements : String [ ]	Cet attribut indique les différents éléments que l'instantiateur utilise pour son fonctionnement. Exemple : Service SPEM
Méthode	**Description**
+getId ( ) : String	
+getVersion ( ) : String	
+getLuncher ( ) : String	

+getName ( ) : String	
+getDescription ( ) : String	
+getFileName ( ) : String	
+getUsedElements ( ) : String [ ]	
+getUpdateLinks ( ) : String [ ]	
+getSupportedFilesExtension ( ) : String [ ]	
+getExternalLibrariesLink ( ) : String [ ]	
+ read (filePath : String)	

InstantiatorXMLPackageReader	
Type de classe	Classe concrète.
Super classe	Aucune.
Interface implémentées	IInstantiatorPackageReader
Attribut	**Description**
-implementation : InstantiatorXMLPackageReaderImplementation	
Méthode	**Description**
+getId ( ) : String	
+getVersion ( ) : String	
+getLuncher ( ) : String	
+getName ( ) : String	
+getDescription ( ) : String	
+getFileName ( ) : String	
+getUsedElements ( ) : String [ ]	
+getUpdateLinks ( ) : String [ ]	
+getSupportedFilesExtension ( ) : String [ ]	
+getExternalLibrariesLink ( ) : String [ ]	
+ read (filePath : String)	

5.3.6 Le composant Provider :

Ce composant « Provider » a pour rôle d'extraire les informations relatives à nos instantiateurs comme le nom, id, description, …

Ce genre de composant est forte utile dans le cas ou l'utilisateur interagit avec un IHM qui offre la possibilité de manipuler et de configurer des composants ou des systèmes, ce qui est le cas pour nous.

- **Diagrammes de composants :**

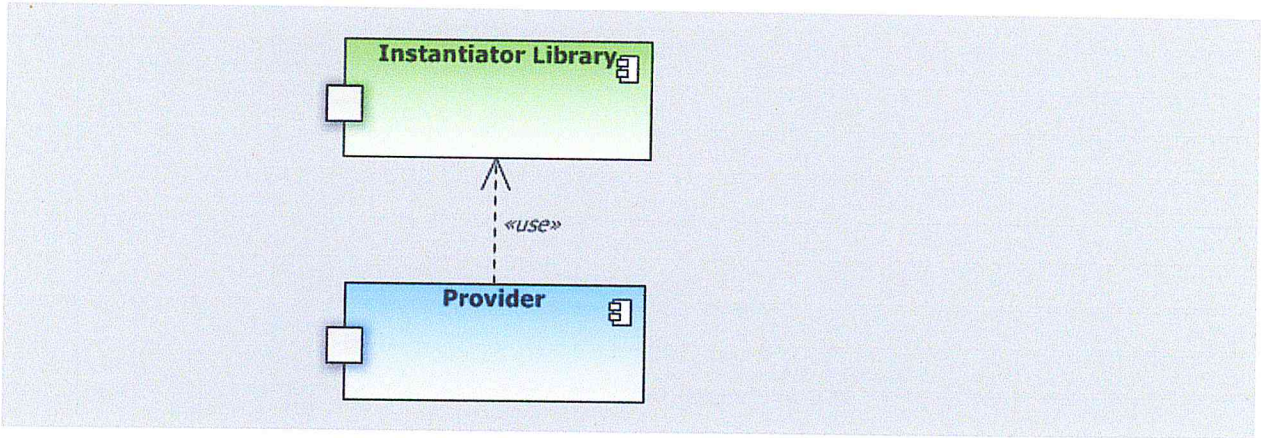

Figure 42 : diagramme de composants « Provider »

- **Diagramme de classes :**

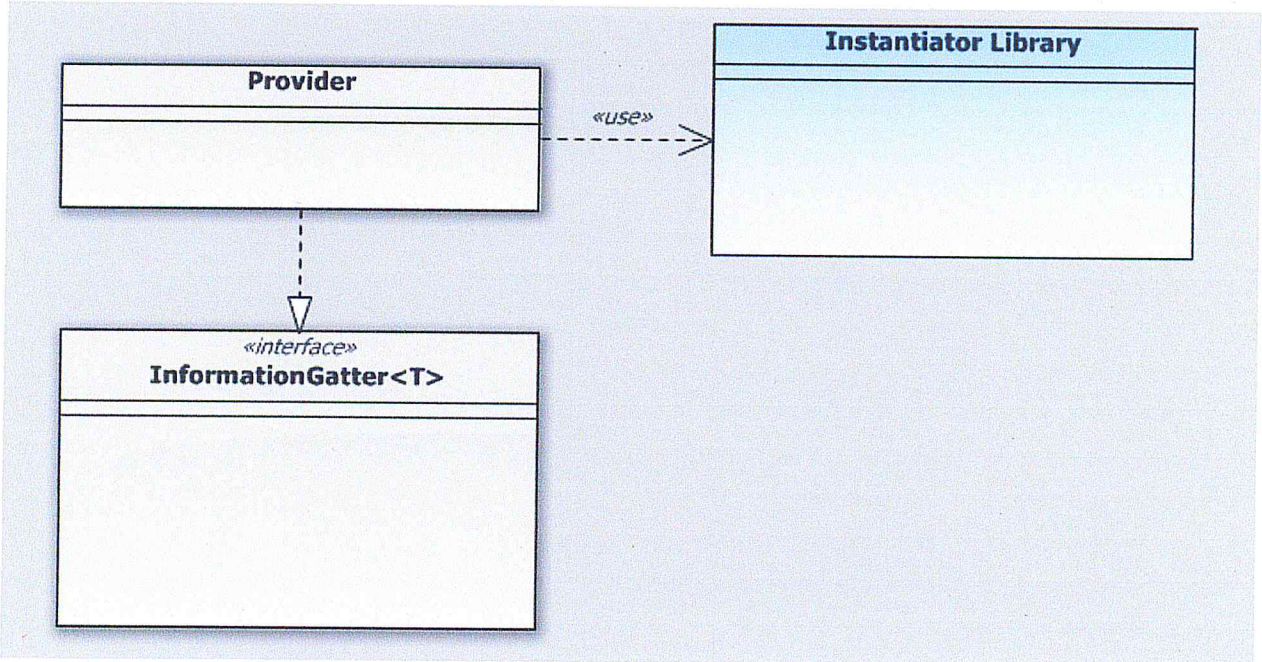

Figure 43 : diagramme de classe « Provider »

5.3.7 Le composant Instantiator Library :

Afin de garantir une bonne gestion de nos instantiateurs, nous avons eu besoin de crée un système qui nous permettrai d'utiliser l'ensemble des informations sur ces derniers d'une manière simple, intuitif et indépendamment de la façon dont ces informations vont être enregistrées, ce sont les objectifs visées par ce composant.

- **Diagramme de composants :**

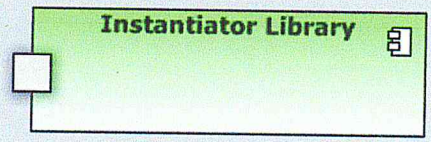

Figure 44 : Diagramme de composants« Instantiator Library »

- **Diagramme de classes :**

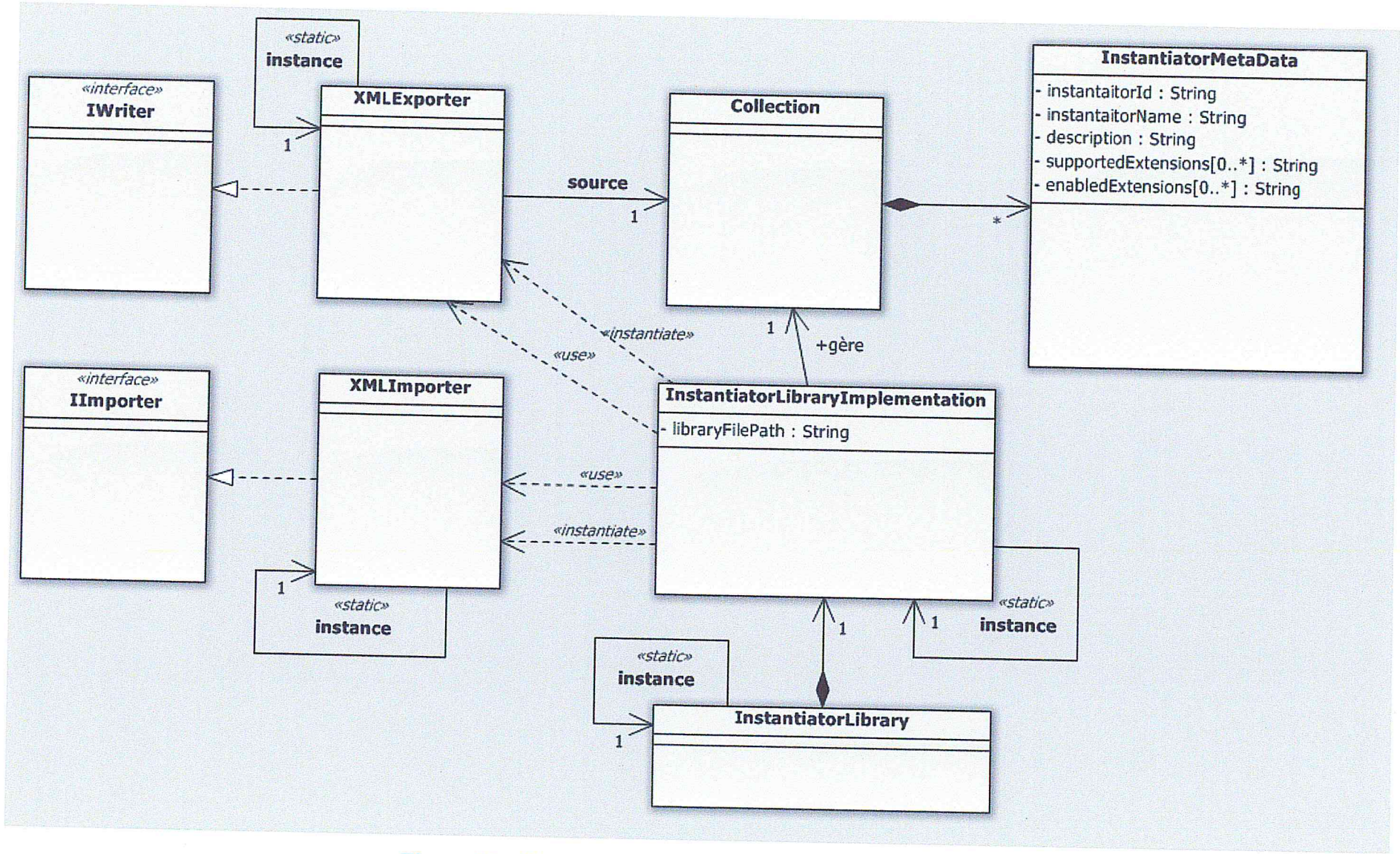

Figure 45 : diagramme de classes « Instantiator Library »

InstantiatorMetaData	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
-instantiatorId : String	Cet attribut renferme l'id de l'instantiateur.
-instantiatorName : String	Le nom de l'instantiateur.
-description : String	La description de l'instantiateur.
-supportedExtensions: String [ ]	L'ensemble des extensions faites pour l'instantiateur.
- enabledExtensions : String [ ]	L'ensemble des extensions que l'instantiateur peut travailler avec.
Méthode	
Des getters et des setters sur l'ensemble des attributs.	

Collection	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
- luncher : InstantiatorMetaData [ ]	Cet attribut indique la collection des métas data géré par notre librairie.
Méthode	**Description**
+ getInstantiatorList ( ) : String []	Cette méthode retourne la liste de tout les id, autrement dit la liste de tous nos instantiateurs.
+ getInstantiatorMetaData (id: String) : InstantiatorMetaData	Permet de récupérer les métas data d'un instantiateur.
+ addInstantiator (x : InstantiatorMetaData)	Permet de d'ajouter les métas data d'un instantiateur.
+ removeInstantiator (id : String)	Permet de supprimer les métas data d'un instantiateur.

IWriter	
Type de classe	Classe abstraite, interface.
Super classe	Aucune.
Méthode	**Description**
+ write(filePath : String)	L'implémentation de cette méthode permet l'écriture du fichier en format cible
+ load( )	L'implémentation de cette méthode permet le chargement des données dans le format cible.

XMLExporter	
Type de classe	Classe concrète.
Super classe	Aucune.
Classes implémentées	IWriter
Attribut	**Description**
-source : Collection	
_- instance : XMLExporter	
Méthode	**Description**
+ write(filePath : String)	
+ load( )	
+ setSource(collection : Collection)	
_+ newInstance : XMLExporter	

IImporter	
Type de classe	Classe abstraite, interface.
Super classe	Aucune.

Méthode	Description
+ load(String filePath) : Collection	L'implémentation de cette méthode permet charger une collection depuis un fichier.

XMLImporter	
Type de classe	Classe concrète.
Super classe	Aucune.
Classes implémentées	IImporter
Attribut	**Description**
_- instance : XMLExporter	
Méthode	**Description**
+ load(String filePath) : Collection	
_+ newInstance : XMLExporter	

InstantiatorLibraryImplemenation	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
-collection : Collection	Cet attribut indique la collection géré par notre bibliothèque.
- libraryFilePath : String	Cet attribut indique l'emplacement du fichier ou toutes nos données sont sauvegardées.
_- instance : InstantiatorLibraryImplemenation	
Méthode	**Description**
_+ newInstance : InstantiatorLibraryImplemenation	

+ getInstantiatorList ( ) : String []	Cette méthode retourne la liste de tout les id, autrement dit la liste de tous nos instantiateurs.
+ getInstantiatorMetaData (id: String) : InstantiatorMetaData	Permet de récupérer les métas data d'un instantiateur.
+ addInstantiator (x : InstantiatorMetaData)	Permet de d'ajouter les métas data d'un instantiateur.
+ removeInstantiator (id : String)	Permet de supprimer les métas data d'un instantiateur.
+associateDefaultInstantiator (id : String, extension : String)	Permet d'associer « activer» une extension à un instantiateur.
+associateDefaultInstantiator (id : String, extensions : String[ ])	Permet d'associer « activer» un ensemble d'extensions à un instantiateur.
+getAssociatedInstantiator (fileExtension : String)	Permet de récupérer l'id du l'instantiateur responsable sur l'extension définit.

InstantiatorLibrary	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
-implementation : InstantiatorLibraryImplementation	
_- instance : InstantiatorLibrary	
Méthode	**Description**
_+ newInstance : InstantiatorLibrary	
+ getInstantiatorList ( ) : String []	Cette méthode retourne la liste de tout les id, autrement dit la liste de tous nos instantiateurs.
+ getInstantiatorMetaData (id: String) : InstantiatorMetaData	Permet de récupérer les métas data d'un instantiateur.

+ addInstantiator (x : InstantiatorMetaData)	Permet de d'ajouter les métas data d'un instantiateur.
+ removeInstantiator (id : String)	Permet de supprimer les métas data d'un instantiateur.
+associateDefaultInstantiator (id : String, extension : String)	Permet d'associer « activer» une extension à un instantiateur.
+associateDefaultInstantiator (id : String, extensions : String[ ])	Permet d'associer « activer» un ensemble d'extensions à un instantiateur.
+getAssociatedInstantiator (fileExtension : String)	Permet de récupérer l'id du l'instantiateur responsable sur l'extension définit.

5.3.8 Instantiation Executioner :

Ce composant est dédié pour le spécialement pour processus d'instanciation, instancier revient à lire un fichier de procédé logiciel, charger ces connaissances, et enfin les écrire dans la forme décrite par notre ontologie de domaine.

Vue la diversité des PML, nous avons opté pour une solution à deux nivaux de vision, la vision micro et la vision macro. Dans cette section nous allons voir l'algorithme général d'instanciation « la vision macro », et au fur et à mesure que nous détaillons nous allons arriver aux algorithmes appropriés pour chaque type de PML « vision micro » et cela dans les sections suivantes.

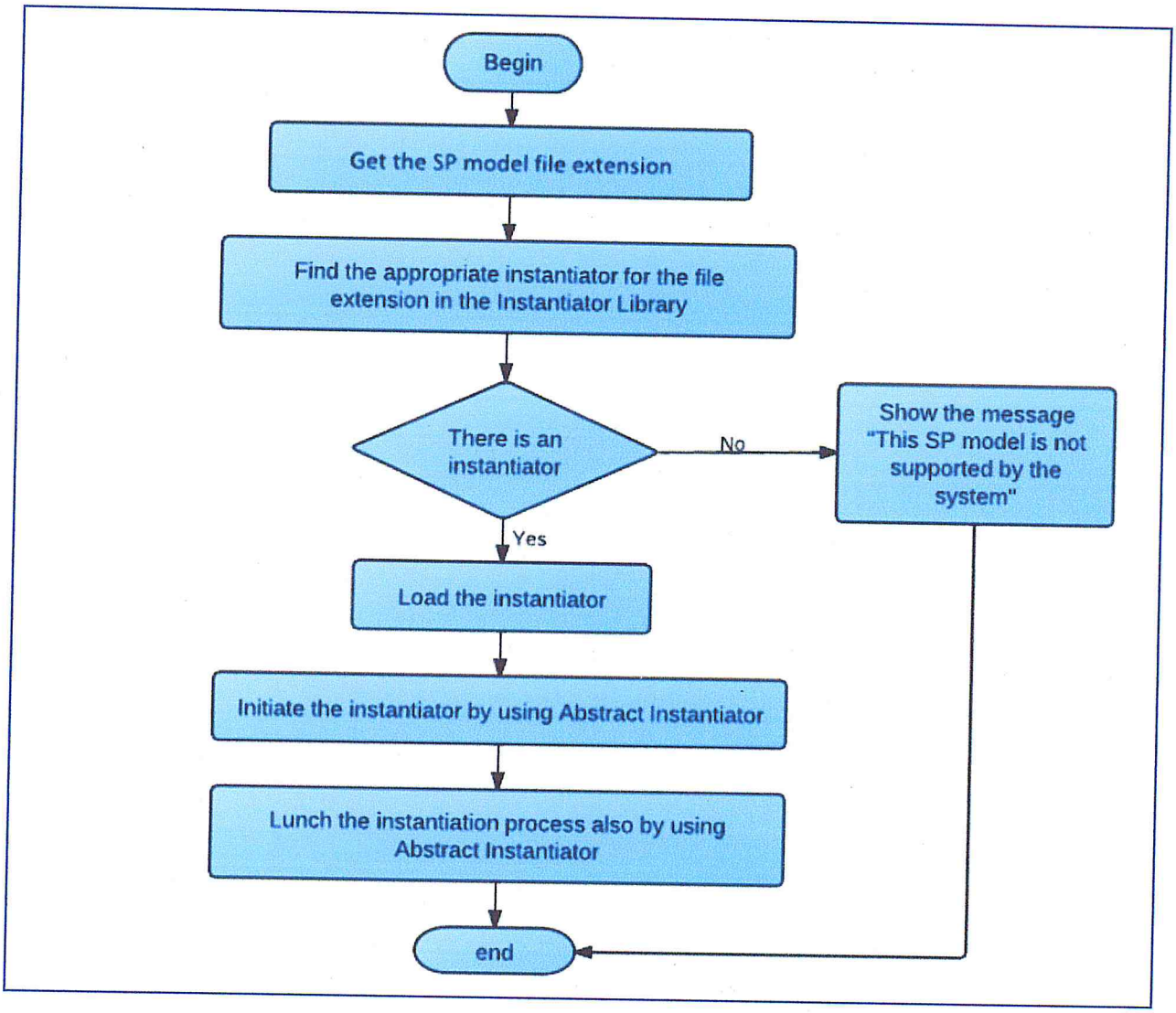

Figure 46 : Algorithme général d'instanciation

- **Diagramme de composants :**

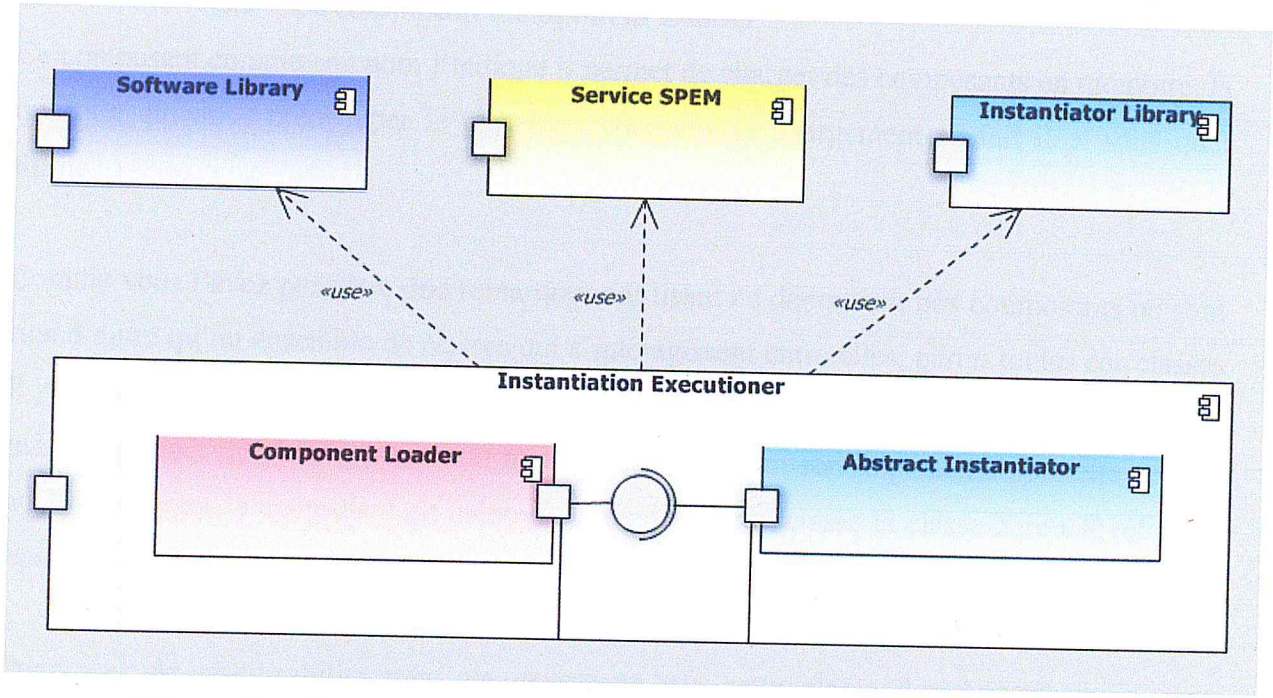

Figure 47 : diagramme de composants « Instantiation Executioner »

- **Diagramme de classes :**

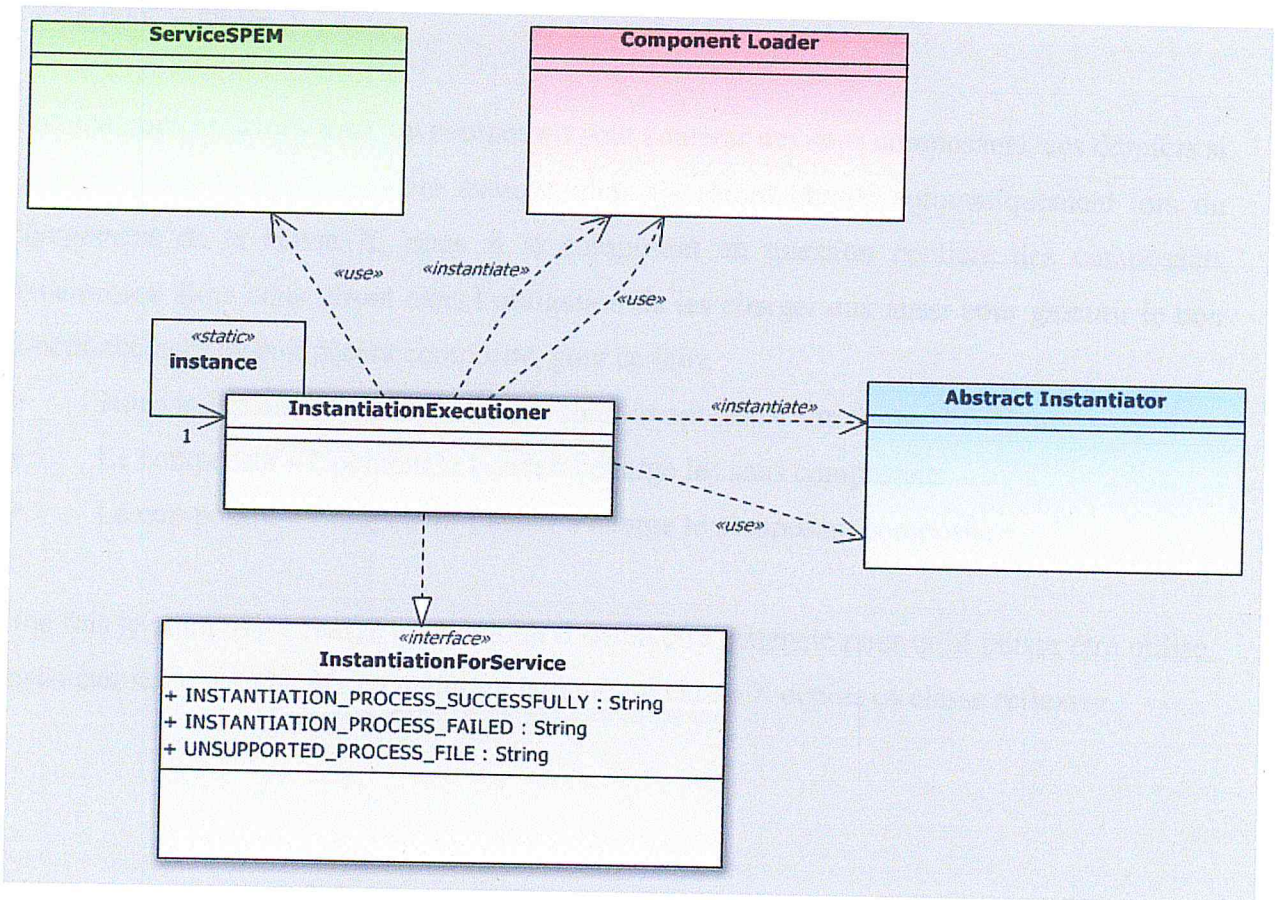

Figure 48 : diagramme de classes « Instantiation Executioner »

- **Diagramme de composants :**

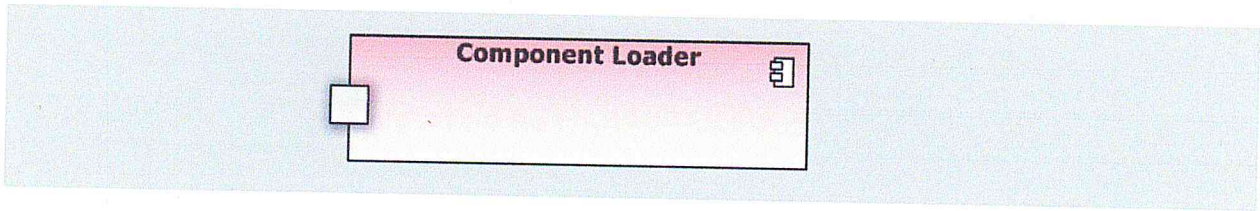

Figure 49 : diagramme de composants « Component Loader »

- **Diagramme de classes :**

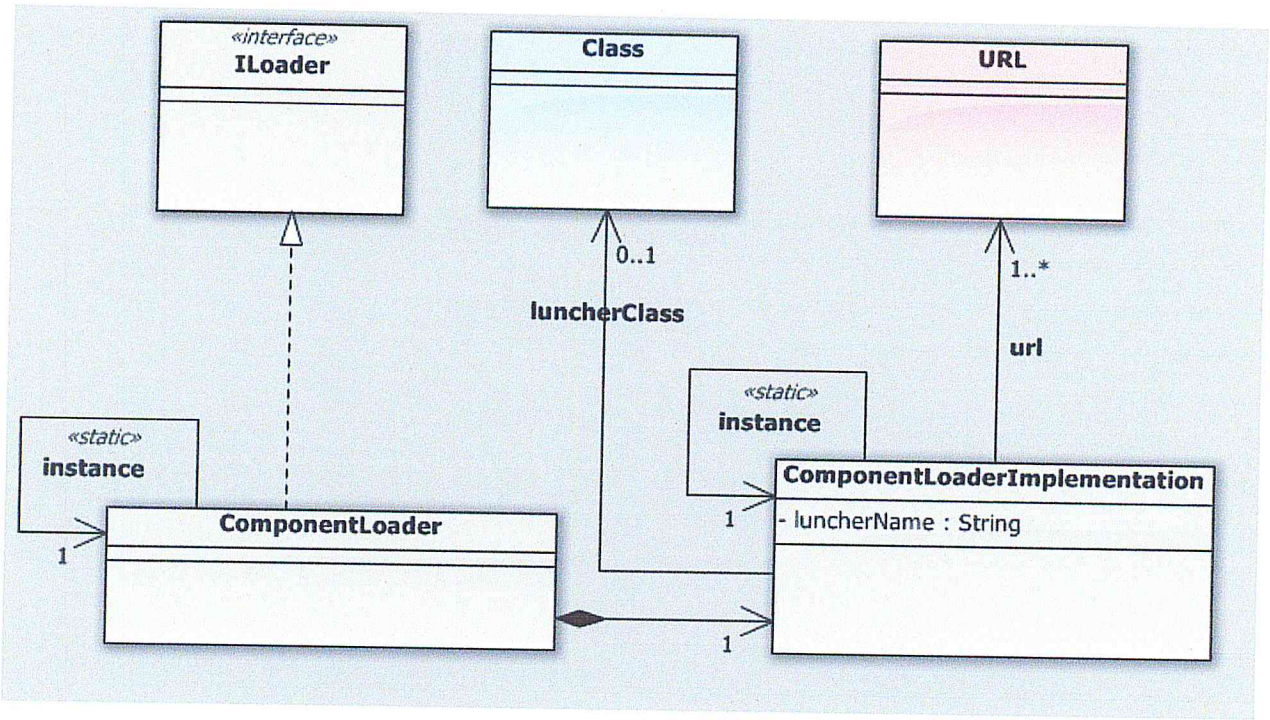

Figure 50 : diagrammes de classes «Component Loader »

- **Description diagramme de classes :**

ILoader	
Type de classe	Classe abstraite, interface.
Super classe	Aucune.
Méthode	**Description**
+load( )	Son implémentation permet le chargement d'un composant.

ComponentLoader	
Type de classe	Classe concrète.
Super classe	Aucune.
Interface implémentées	ILoader
Attribut	**Description**
- implementation : ComponentLoaderImplementation	
_- instance : ComponentLoader	
Méthode	**Description**
_+ newInstance ( ) : ComponentLoaderImplementation	
+ load( )	
+ setLuncherName(arg: String)	
+ setURL(arg : URL[ ])	
+ getLuncherClass ( ) : Class	

ComponentLoaderImplementation	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
- luncherName : String	Cet attribut indique le nom de la classe référencé
- url : URL [ ]	Cet attribut indique l'URL du composant et de ses composants.
- luncherClass : Class	Cet attribut indique la class réflexive de la classe référencé.
_- instance : ComponentLoaderImplementation	

Méthode	Description
_+ newInstance ( ) : ComponentLoaderImplementation	
+ load( )	
+ setLuncherName(arg: String)	
+ setURL(arg : URL[ ])	
+ getLuncherClass ( ) : Class	

5.3.10 Le composant Abstract Instantiator :

Ce composant est un instantiateur dans ça structure et dans son comportement, cependant celui là ne se limite pas à un seul PML particulier mais bien au-delà, il permet d'exécuter le processus d'instanciation sur tous les type de PML comme s'il s'agissait d'un seul PML.

Pour instancier ce composant suit les étapes suivantes :

- Le composant instancie la classe référencé par la classe réflexive, cette dernière doit être mise à disposition avant de commencer la procédure d'instanciation.

La classe référencé représente un instantiateur local, ce dernier peut être élémentaire ou multi PML.

- Le composant configure l'instantiateur locale.
- Le composant lance le processus d'instanciation depuis l'instantiateur local.
- Le composant retourne le résultat du processus.

- **Diagramme de composants :**

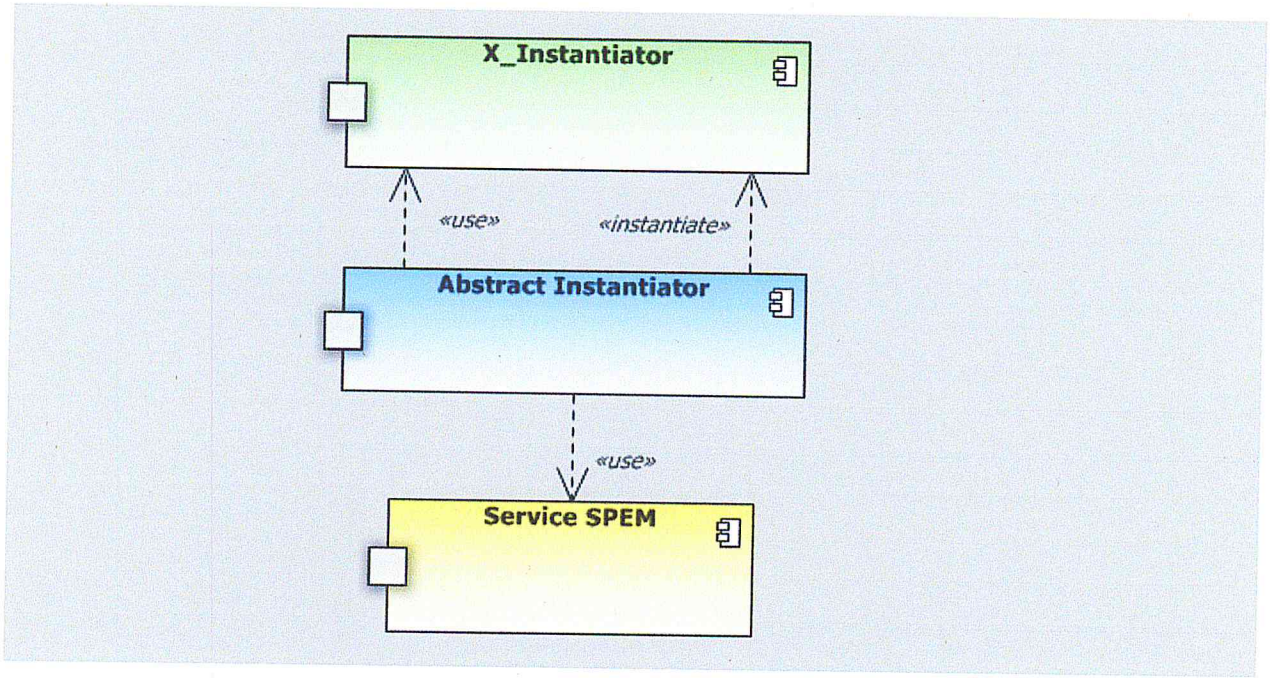

Figure 51 : diagramme de composants « Abstract Instantiator »

- **Diagramme de classes :**

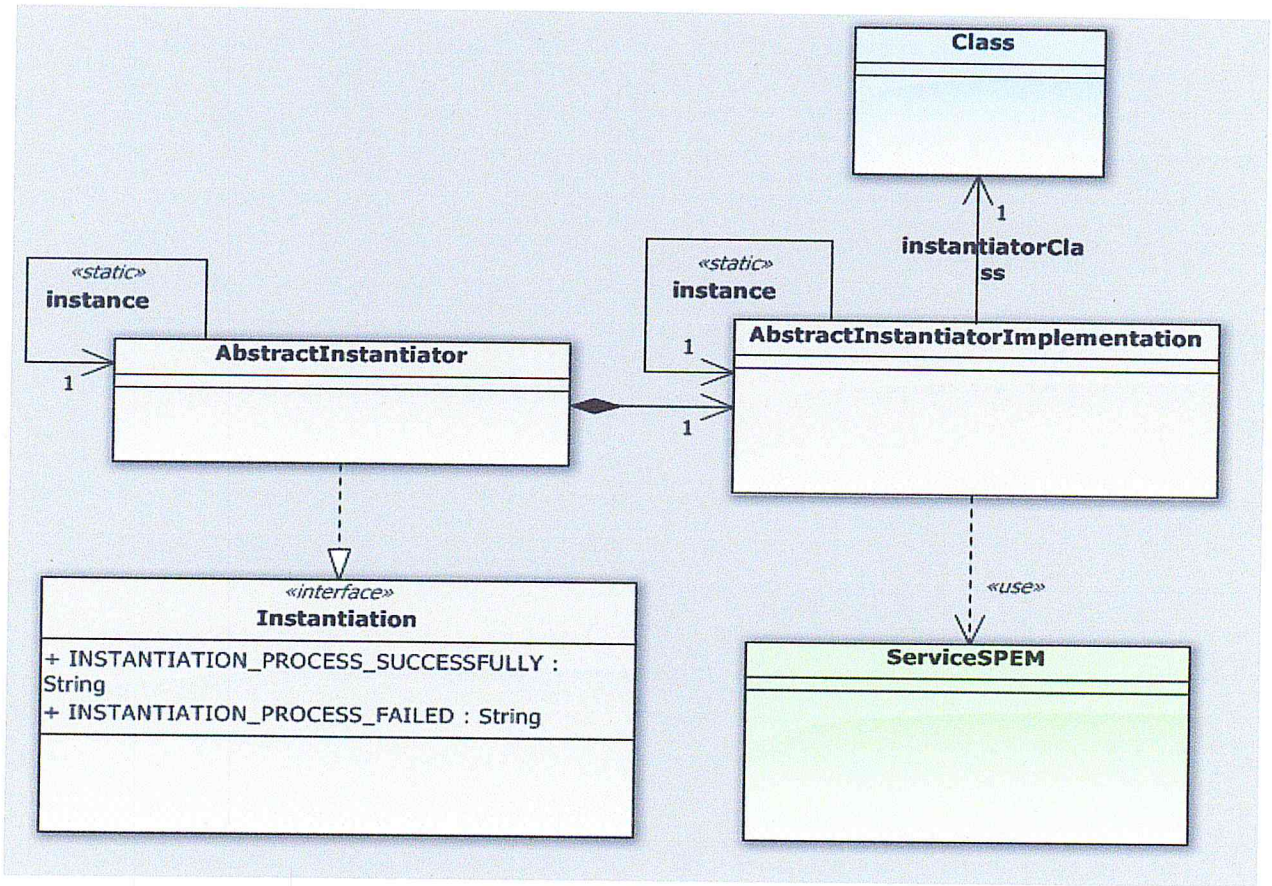

Figure 52 : diagramme de classes « Abstract Instantiator »

- **Description diagramme de classes :**

Instantiation	
Type de classe	Classe abstraite, interface.
Super classe	Aucune.
Attribut	**Description**
_+INSTANTIATION_PROCESS_SUCCESSFULLY : String = "instantiation process successfully"	Indique que le processus d'instanciation c'est terminé avec succès.
_+INSTANTIATION_PROCESS_FAILED : String = "instantiation process failed"	Indique un échec dans le processus d'instanciation.
Méthode	**Description**
+instantiate(filePath : String) : String	
+setServiceSPEM(service : ServiceSPEM)	

AbstractInstantiator	
Type de classe	Classe concrète.
Super classe	Aucune.
Interface implémentées	Instantiation
Attribut	**Description**
- implementation : AbstractInstantiatorImplementation	
_- instance : AbstractInstantiator	
Méthode	**Description**
_+ newInstance ( ) : AbstractInstantiator	
+instantiate(filePath : String) : String	
+setServiceSPEM(service : ServiceSPEM)	
+ setRealInstantiator(instantiatorClass : Class)	

AbstractInstantiatorImplementation	
Type de classe	Classe concrète.
Super classe	Aucune.
Interface implémentées	Instantiation
Attribut	**Description**
- instantiatorClass : Class	
_- instance : AbstractInstantiatorImplementation	
Méthode	**Description**
_+ newInstance ( ) : AbstractInstantiatorImplementation	
+instantiate(filePath : String) : String	
+setServiceSPEM(service : ServiceSPEM)	
+ setRealInstantiator(instantiatorClass : Class)	

5.3.11 Le composant X_Instantiator :

Ce composant se charge du processus d'instanciation, il est la brique de base pour les composants « Instantiation Service » et « Instantiation Application ».

Ce composant n'existe pas dans la réalité, nous l'avons appelé ainsi pour référencer une conception générique de tout nos instantiateur.

Les instantiateurs utilisent tous le composant « Service SPEM », respectent l'interface « Instantiation » et lance le processus d'instanciation. Les instantiateurs peuvent avoir une conception qui diffère d'un instantiateur à un autre mais ils s'entendent tous sur les trois points cités précédemment.

Dans le cadre de notre étude nous avons travaillé uniquement avec l'instantiateur PBOOL et l'instantiateur EPF, d'ailleurs c'est ce que nous allons voir juste après.

5.3.12 PBOOL Instantiator :

Ce composant est dédié spécialement pour instancier des procédés logiciels qui respectent le méta mode PBOOL.

Pour y remédier nous avons décomposé ce travail en deux parties :

- La lecture et le chargement des connaissances dans le format PBOOL.

- La traduction des connaissances PBOOL en connaissances SPEM, puis écriture de ces derniers dans notre ontologie de domaine.
- **Diagramme de composants :**

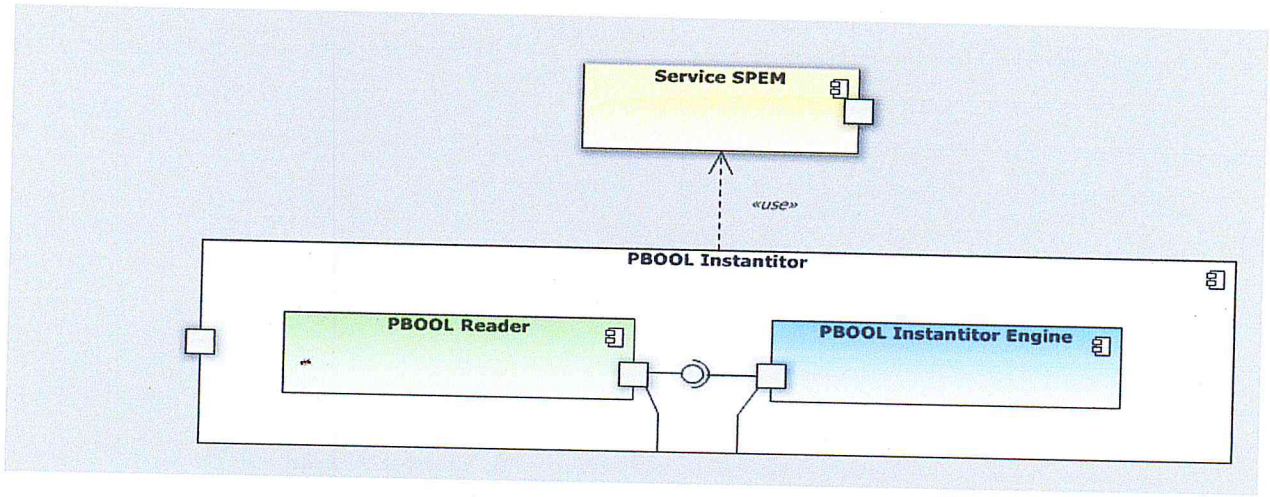

Figure 53 : diagramme de composants « PBOOL Instantiator »

- **Diagramme de classes :**

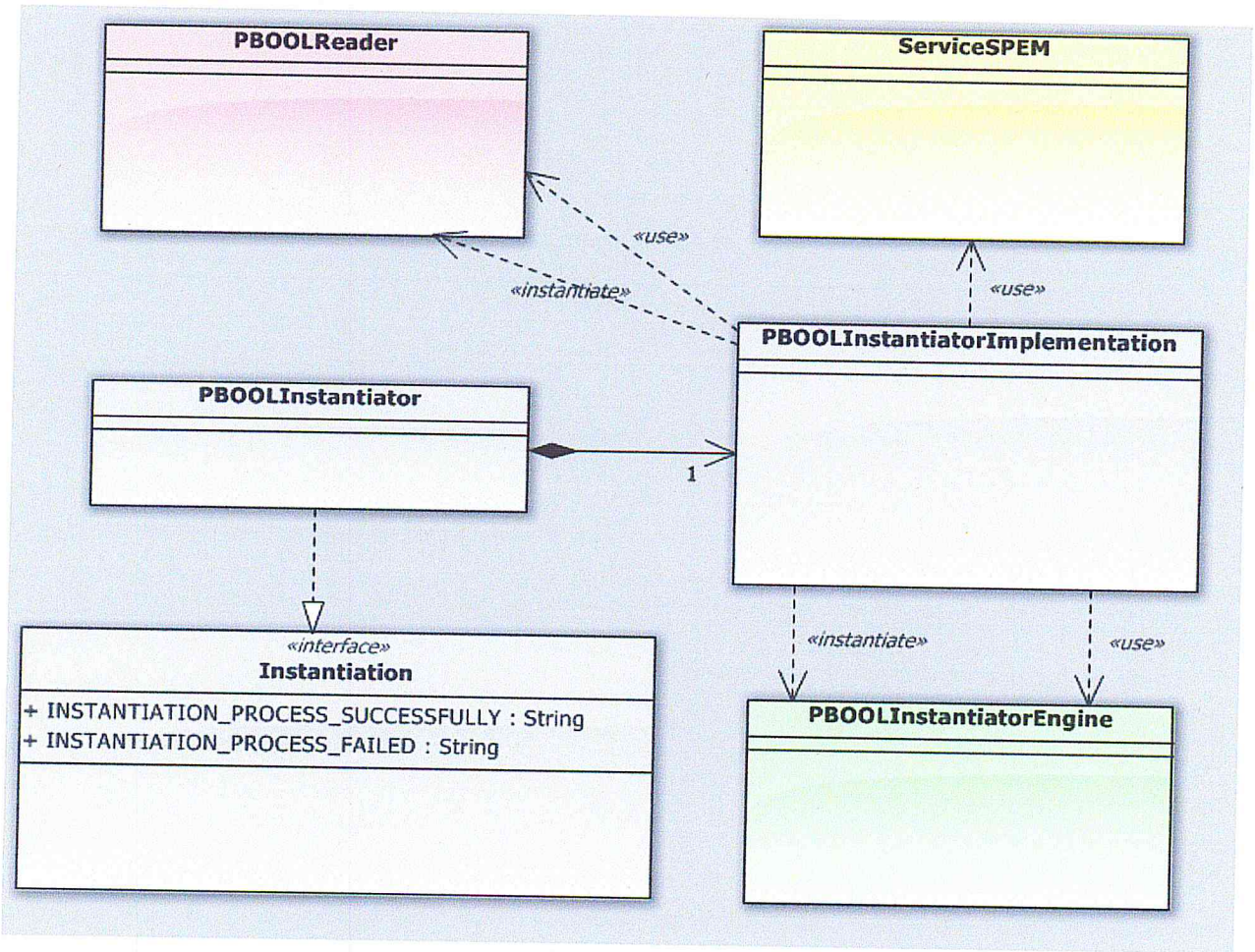

Figure 54 : Diagramme de classes « PBOOL Instantiator »

- **Description diagramme de classes :**

Instantiation	
Type de classe	Classe abstraite, interface.
Super classe	Aucune.
Attribut	**Description**
_+INSTANTIATION_PROCESS_SUCCESSFULLY : String = "instantiation process successfully"	Indique que le processus d'instanciation c'est terminé avec succès.
_+INSTANTIATION_PROCESS_FAILED : String = "instantiation process failed"	Indique un échec dans le processus d'instanciation.
Méthode	**Description**
+instantiate(filePath : String) : String	
+setServiceSPEM(service : ServiceSPEM)	

PBOOLInstantiator	
Type de classe	Classe concrète.
Super classe	Aucune.
Interface implémentées	Instantiation
Attribut	**Description**
- implementation : PBOOLInstantiatorImplementation	
_- instance : PBOOLInstantiator	
Méthode	**Description**
_+ newInstance ( ) : PBOOLInstantiator	
+ instantiate(filePath : String) : String	
+ setServiceSPEM(service : ServiceSPEM)	

PBOOLImplementation	
Type de classe	Classe concrète.
Super classe	Aucune.
Interface implémentées	Instantiation
Attribut	**Description**
- service : ServiceSPEM	
_- instance : AbstractInstantiatorImplementation	
Méthode	**Description**
_+ newInstance ( ) : PBOOLImplementation	
+instantiate(filePath : String) : String	
+setServiceSPEM(service : ServiceSPEM)	

5.3.13 Le composant PBOOL Reader :

Ce composant est fait pour lire des fichier PBOOL, extraires leurs données et les mettre à disposition de l'utilisateurs sous forme de source de données PBOOL.

Une source données c'est un ensemble d'élément d'un certain méta model qui sont regroupés et arrangés d'une manière particulière, son avantages réside dans sa souplesse.

Dans notre méta modèle PBOOL nous avons pris le stricte minimum de concepts et de relations pour l'unique raison que notre processus d'instantiation se base sur eux.

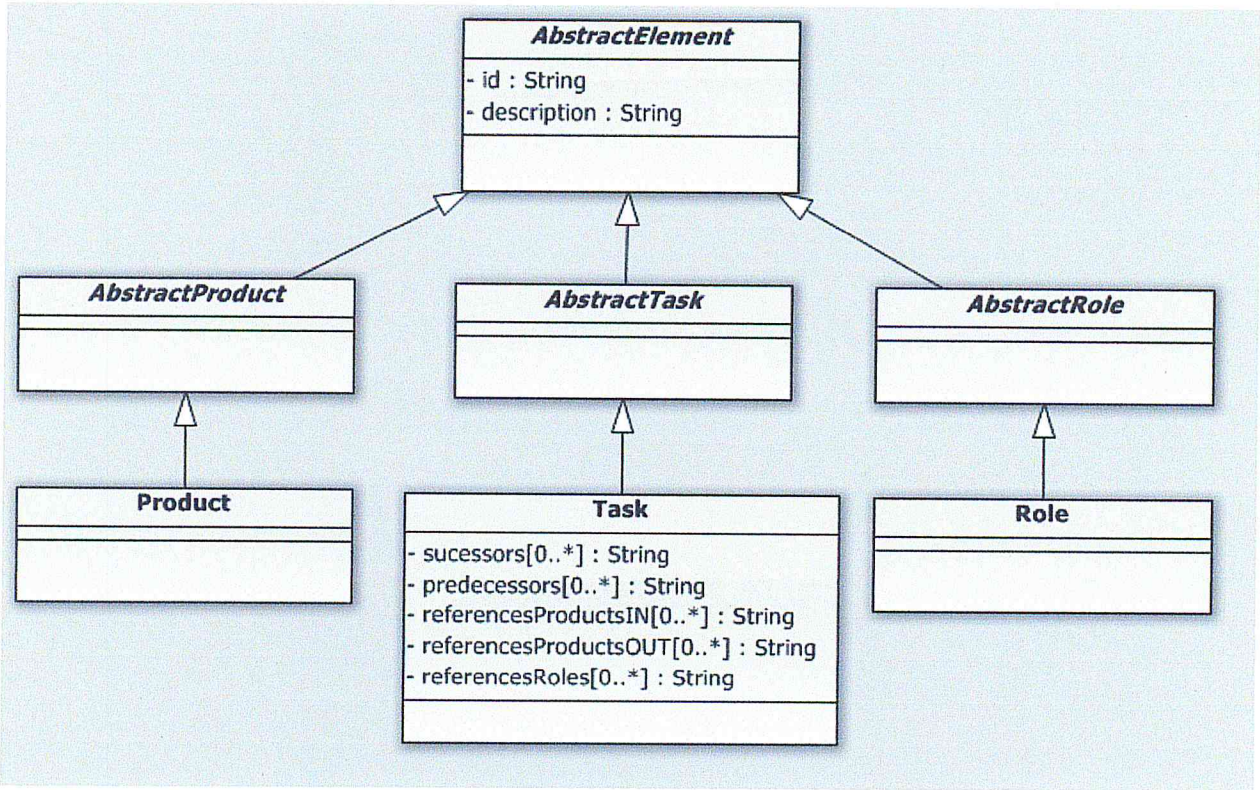

Figure 55 : méta model PBOOL partiel et simplifié.

- **Diagramme de classes :**

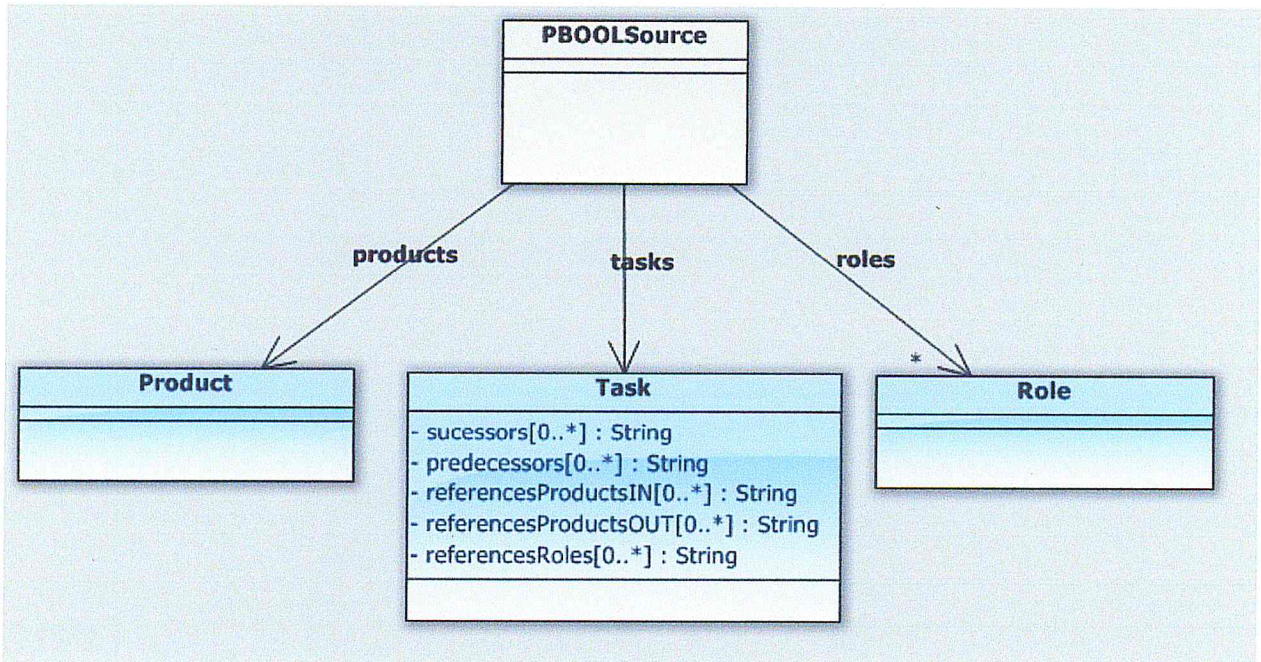

Figure 56 : diagramme de classes pour la source de données PBOOL

- **Diagramme de composants :**

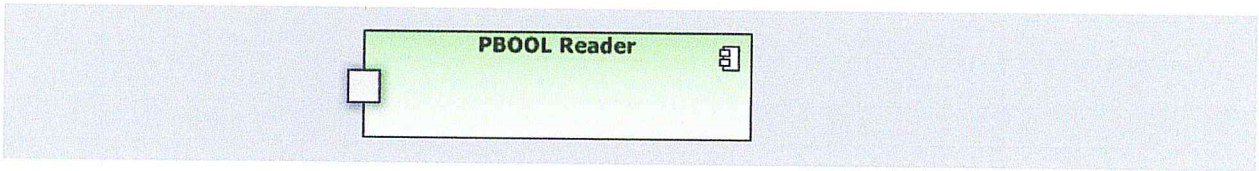

Figure 57 : Diagramme de composants« PBOOL Reader »

- **Diagramme de classes :**

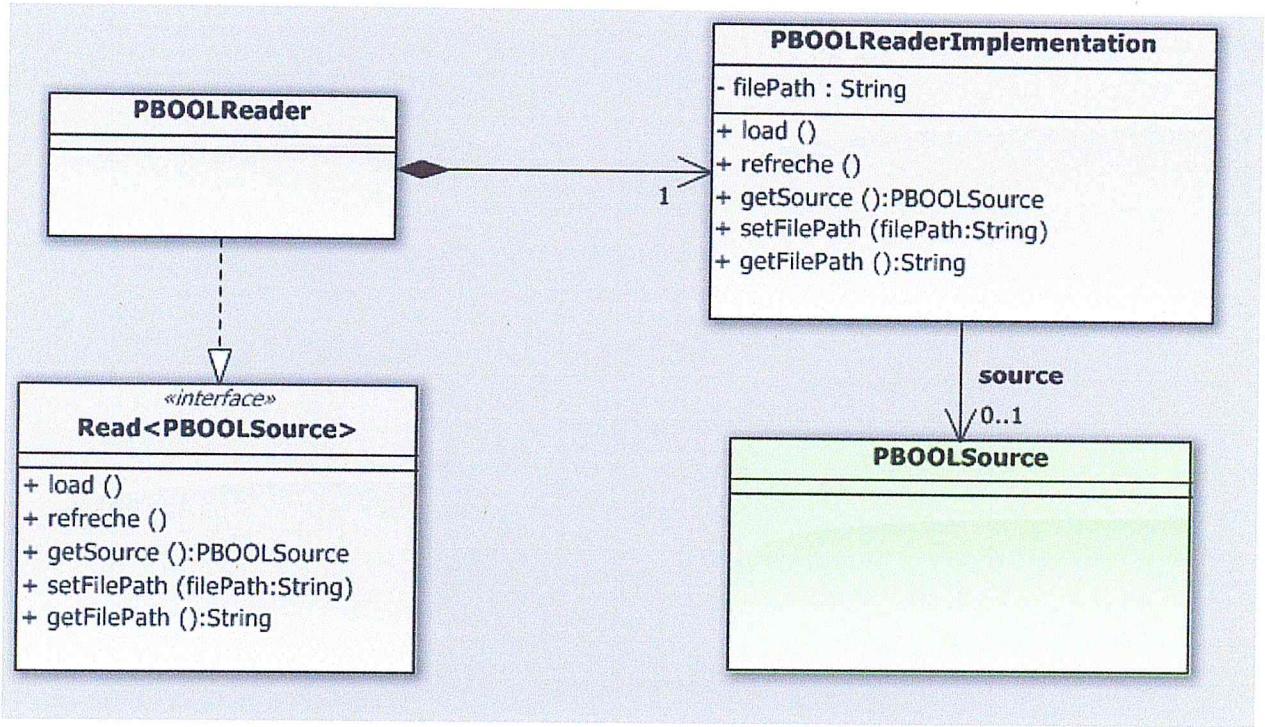

Figure 58 : diagramme de classes pour le composant « PBOOL Reader »

5.3.14 PBOOL Instantiator Engine:

Ce composant constitue le noyeau du composant « PBOOL Instantiator », il contien l'algorithme locale qui va nous permettre d'exécuter le processus d'instanciation, pour ce faire nous devons tout d'abord lui indiquer une source PBOOL valable puis nous lançon le processus d'instanciation on invokon la méthode approprié pour cela.

Dans la phase d'instanciation le composant PBOOL Instantiator transcrit les différents connaissances PBOOL en connaissance SPEM, en d'autres termes ce derniers crée dans notre ontologie un procédée logiciel qui respectent les standards décrit par le méta model SPEM et qui et équivalent à celui qui nous à servie de source.

- **Diagramme de composants :**

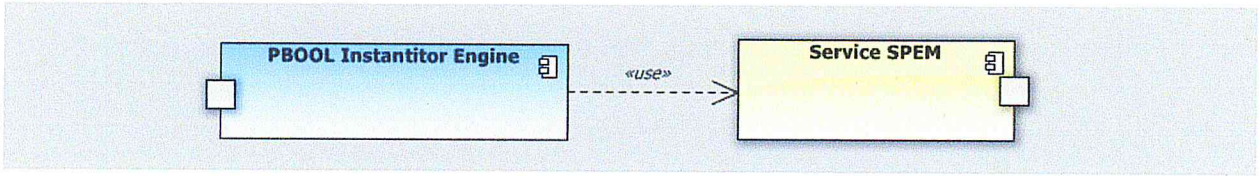

Figure 59 : diagramme de composants «PBOOL Instantiator Engine»

- **Diagramme de classes :**

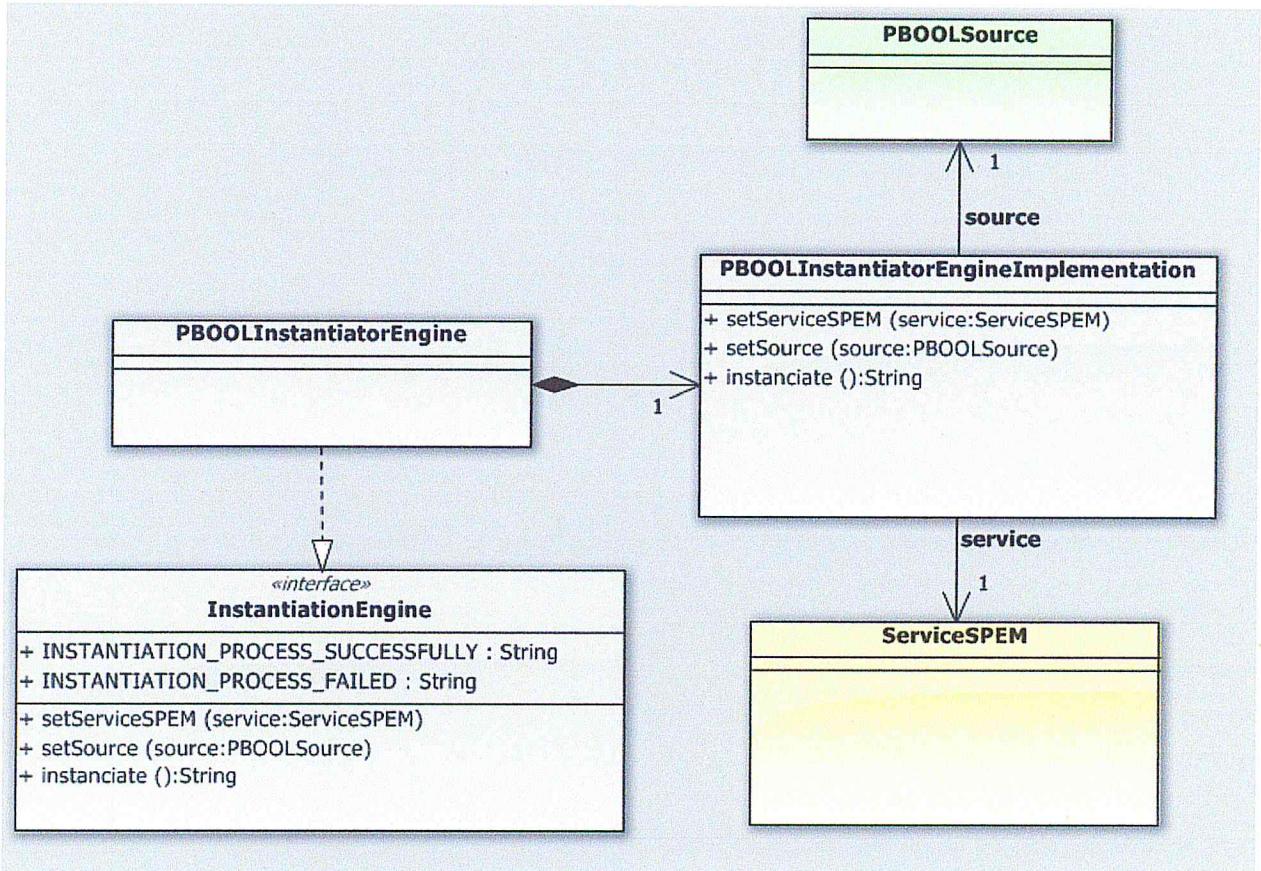

Figure 60 : diagramme de classes « PBOOL Instantiator Engine »

5.3.15 Le composant EPF Instantiator :

Ce composant est dédié spécialement pour instancier des procédés logiciels qui respectent le méta mode SPEM.

Pour y remédier nous avons décomposé ce travail en deux parties :

- La lecture et le chargement des connaissances dans le format EPF.
- L'écriture des connaissances au sein de notre ontologie domaine.

- **Diagramme de composants :**

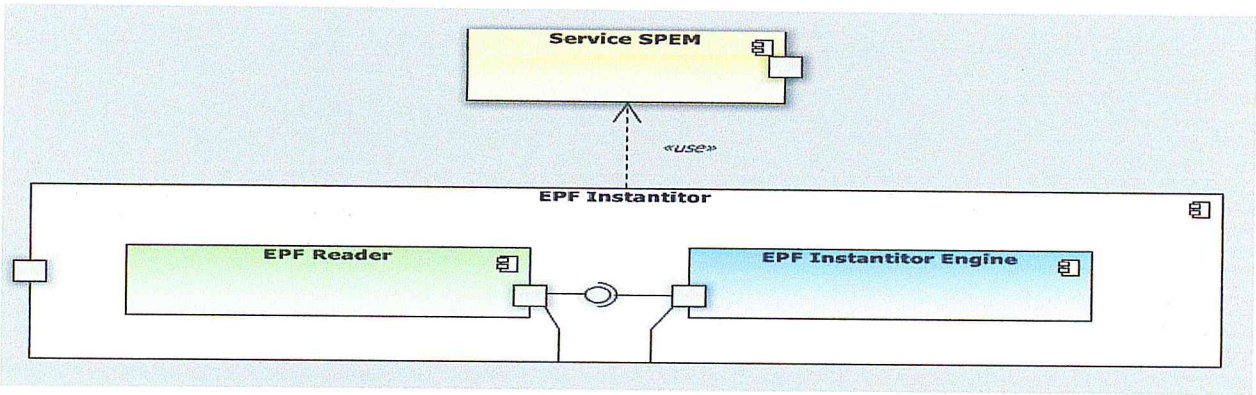

Figure 61 : diagramme de composants « PBOOL Instantiator »

- **Diagramme de classes :**

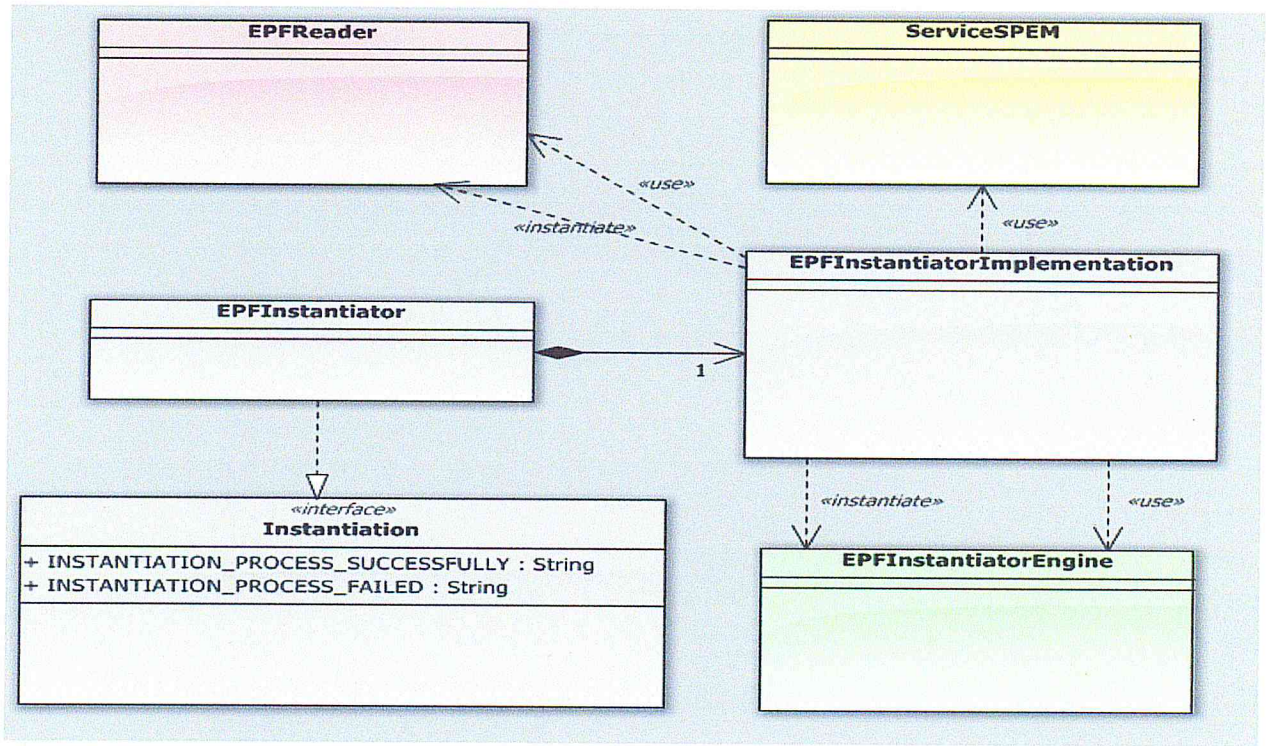

Figure 62 : Diagramme de classes « PBOOL Instantiator »

- **Description diagramme de classes :**

Instantiation	
Type de classe	Classe abstraite, interface.
Super classe	Aucune.
Attribut	**Description**
_+INSTANTIATION_PROCESS_SUCCESSFULLY : String = "instantiation process successfully"	Indique que le processus d'instanciation c'est terminé avec succès.
_+INSTANTIATION_PROCESS_FAILED : String = "instantiation process failed"	Indique un échec dans le processus d'instanciation.
Méthode	**Description**
+instantiate(filePath : String) : String	
+setServiceSPEM(service : ServiceSPEM)	

EPFInstantiator	
Type de classe	Classe concrète.
Super classe	Aucune.
Interface implémentées	Instantiation
Attribut	**Description**
- implementation : EPFInstantiatorImplementation	
_- instance : EPFInstantiator	
Méthode	**Description**
_+ newInstance ( ) : EPFInstantiator	
+instantiate(filePath : String) : String	
+setServiceSPEM(service : ServiceSPEM)	

EPFInstantiatorImplementation	
Type de classe	Classe concrète.
Super classe	Aucune.
Interface implémentées	Instantiation
Attribut	**Description**
- service : ServiceSPEM	
_- instance : EPFInstantiatorImplementation	
Méthode	**Description**
_+ newInstance ( ) : EPFInstantiatorImplementation	
+instantiate(filePath : String) : String	
+setServiceSPEM(service : ServiceSPEM)	

5.3.16 EPF Reader :

Ce composant est fait pour lire des fichier EPF, les fichier EPF sont décrits dans le méta model des procédés logiciel SPEM, se qui les rend compatible avec notre ontologie de domaine.

Comme pour PBOOL Reader nous avons aussi utilisé une source de données pour celui la, la source de donnée EPF est composé d'un ensemble de collections de nos différents concepts procédés logiciels.

Dans notre cas, nous avons utilisé est le même méta modèle SPEM que celui définis par l'organisation l'OMG.

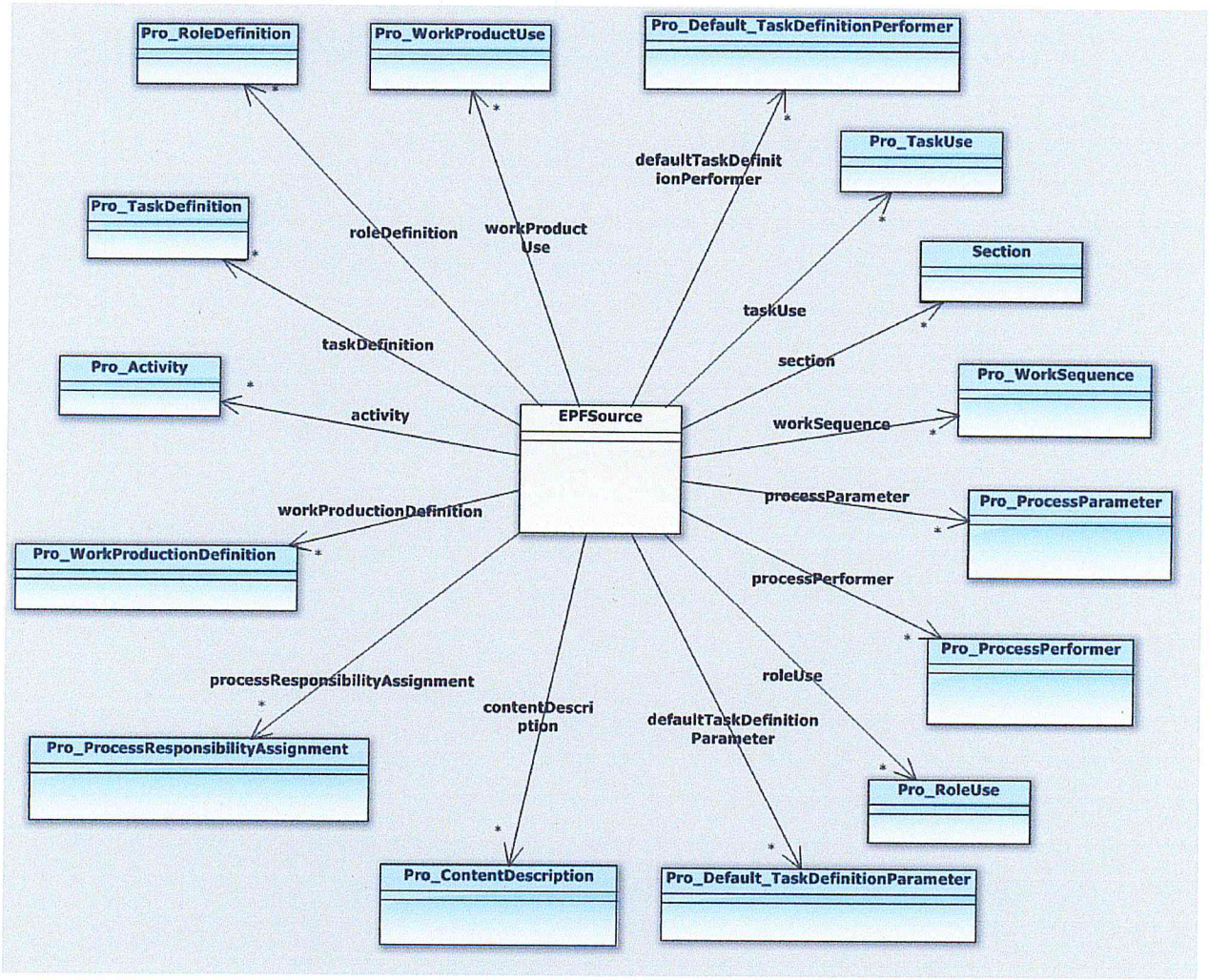

Figure 63 : diagramme de classes pour la source de données EPF

- **Diagramme de composants :**

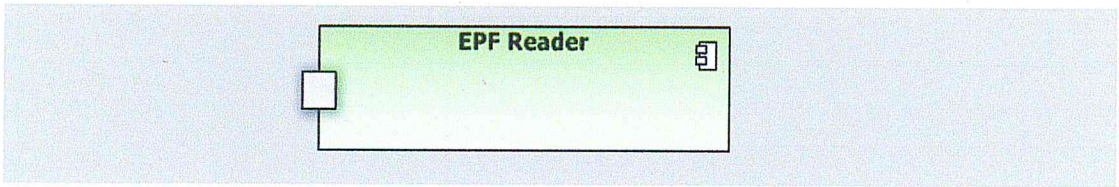

Figure 64 : Diagramme de composants« PBOOL Reader »

- **Diagramme de classes :**

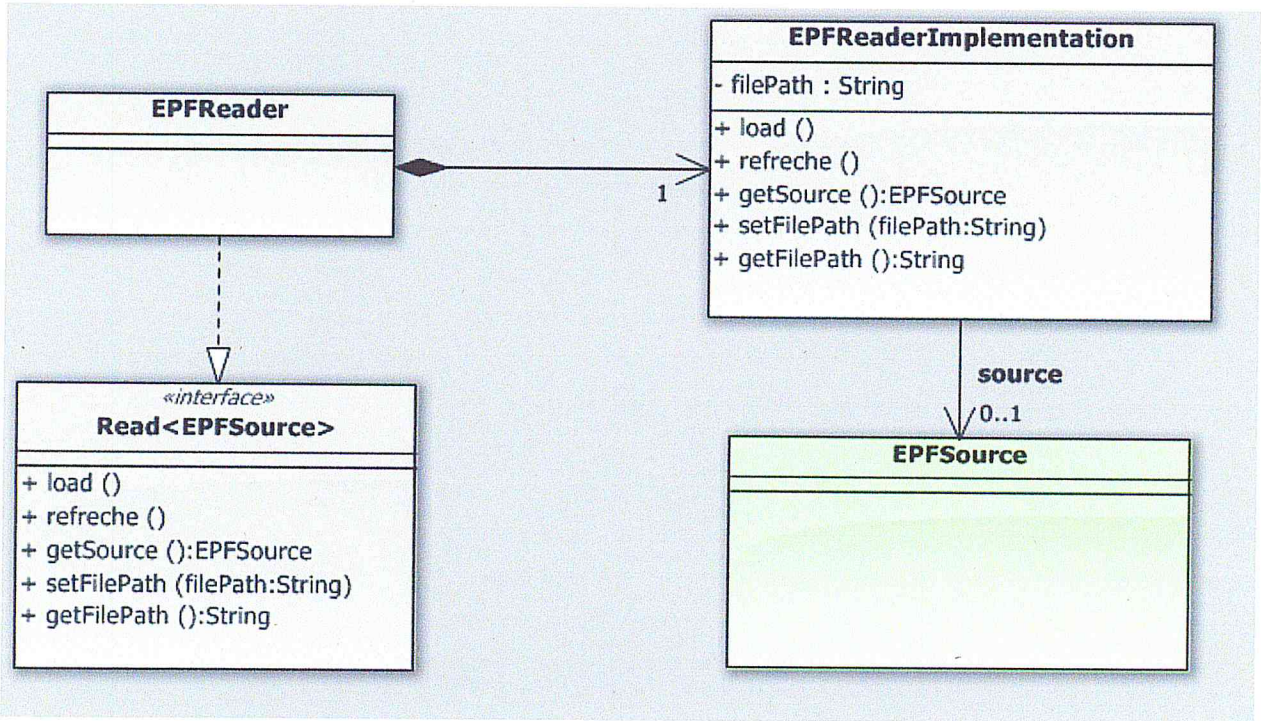

Figure 65 : diagramme de classes pour le composant « PBOOL Reader »

5.3.17 Le composant EPF Instantiator Engine:

Ce composant constitue le noyeau du composant « EPF Instantiator », il contien l'algorithme locale qui va nous permettre d'executer le processus d'instanciation d'un procédé logiciel en EPF, pour ce faire il suit les mêmes étapes que son conjoint « PBOOL Instantiator Engine » la seul différence réside dans le faite que la source utilisé est une source EPF, et dans la manière d'écrire ces données « algorithme locale d'instanciation EPF ».

- **Diagramme de composants :**

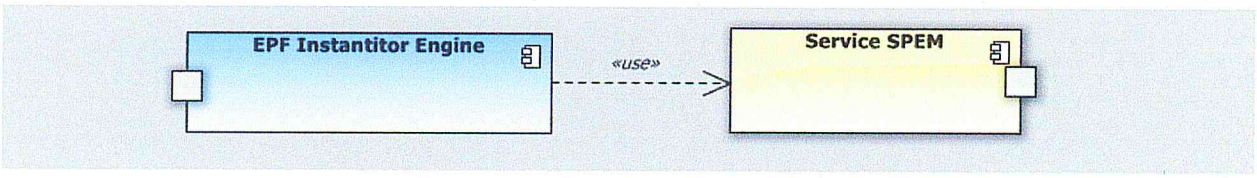

Figure 66 : diagramme de composants «PBOOL Instantiator Engine»

- **Diagramme de classes :**

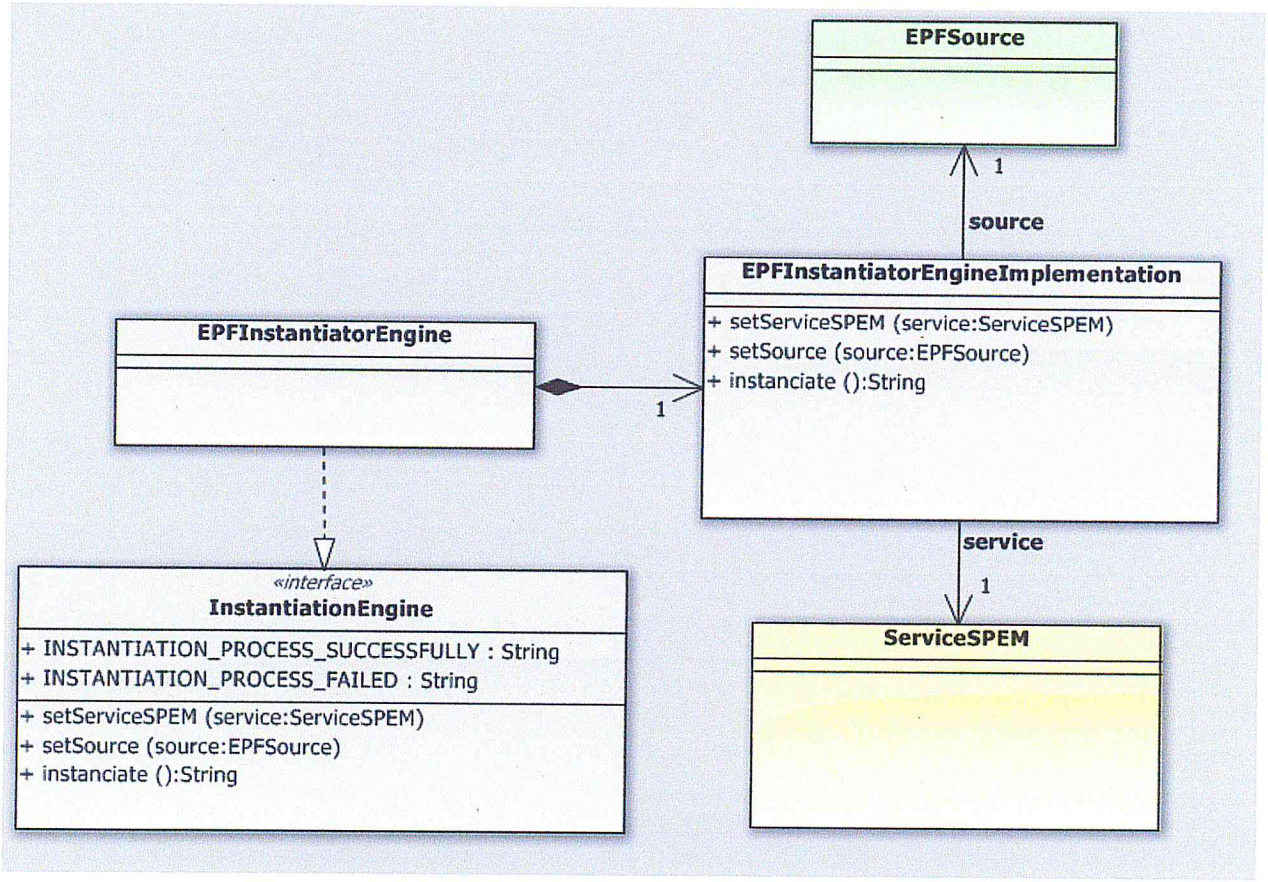

Figure 67 : diagramme de classes « PBOOL Instantiator Engine »

5.4 Le composant Deployment Application :

Ce composent a été crée pour satisfaire notre besoin de déploiement décrit dans le diagramme de cas d'utilisation globale.

Le processus de déploiement consiste à projeter une configuration PL dans un langage de procédés logiciels bien déterminé.

Dans notre cadre d'étude on c'est limités uniquement aux configurations décrites en ACME, ces derniers vont être déployées en procédés logiciels qui doivent respecter le méta model SPEM.

Les figures Figure 68, Figure 69 et Figure 70 montrent le méta model ACME que nous avons créé et qui regroupe l'ensemble des concepts architecturaux nécessaire à la définition d'une architecture logicielle.

Pour faire le déploiement nombreuse sont les techniques « modes » à utiliser, pour cela nous avons créés et implémenter spécialement deux modes de déploiement qui sont le déploiement total et le déploiement distribué.

Le déploiement distribué consiste à récupérer les différents composants, et les faire déployer un à un d'une manière individuel, pour y faire on doit suivre les étapes suivantes:

- Chercher et trouver des produits correspondants aux ports dans notre ontologie de domaine.
- Chercher un enchainement d'activité dans notre ontologie de domaine qui peuvent satisfaire nos produits en entrées et en sortis, ceci est fait via le composant du raisonnement.
- Crée un document EPF vide.
- A l'intérieur du document, crée une activité EPF qui porte le même nom que le composant, cette dernière utilise les mêmes ports en entrées et en sortis que ceux utilisées lors du raisonnement et pour terminer ajouter les activités trouvé dans l'étape du raisonnement comme étant des tâches qui appartiennent à l'activité.

Le déploiement total est un autre mode déploiement, et comme son nom l'indique il permet de déployer l'intégralité de la configuration.

Le déploiement se fait de la manière suivante :

- Chercher une correspondance port/produit dans notre ontologie de domaine, et cela pour tout les composants.
- Chercher un enchainement d'activité dans notre ontologie de domaine qui peuvent satisfaire nos produits en entrées et en sortis, ceci est fait via le composant du raisonnement.
- Déduire l'enchainement des activités pour les connecteurs.
- Crée un document EPF vide.
- A l'intérieur du document, crée des activités EPF qui porte le même nom que leur composant, de la même manière qu'on a vue dans le déploiement distribué.
- A l'intérieur du document, crée les activités EPF pour les connecteurs si elles existent.
- Lier les activités des connecteurs avec celles des composants par des liens de transmissions.

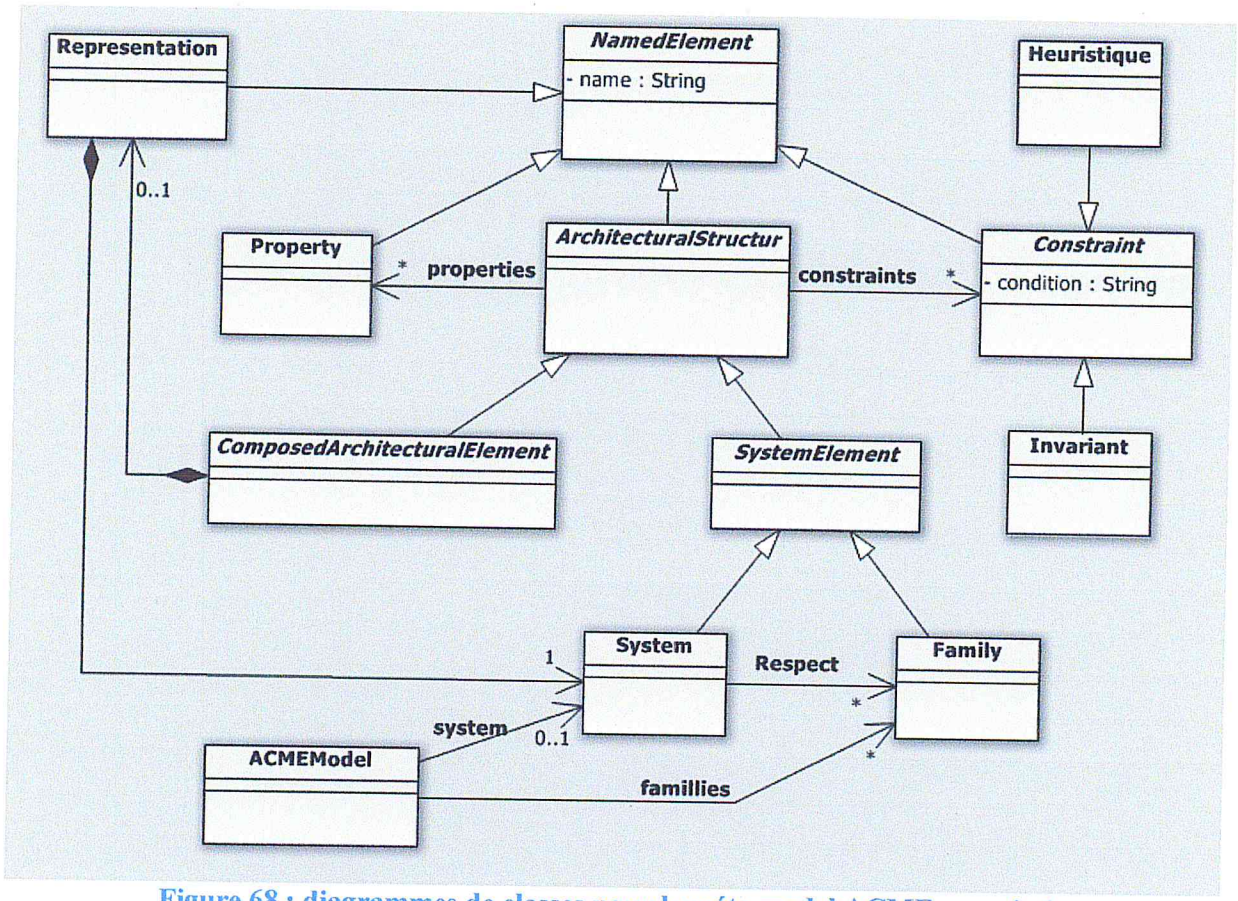

Figure 68 : diagrammes de classes pour le méta model ACME - partie 1

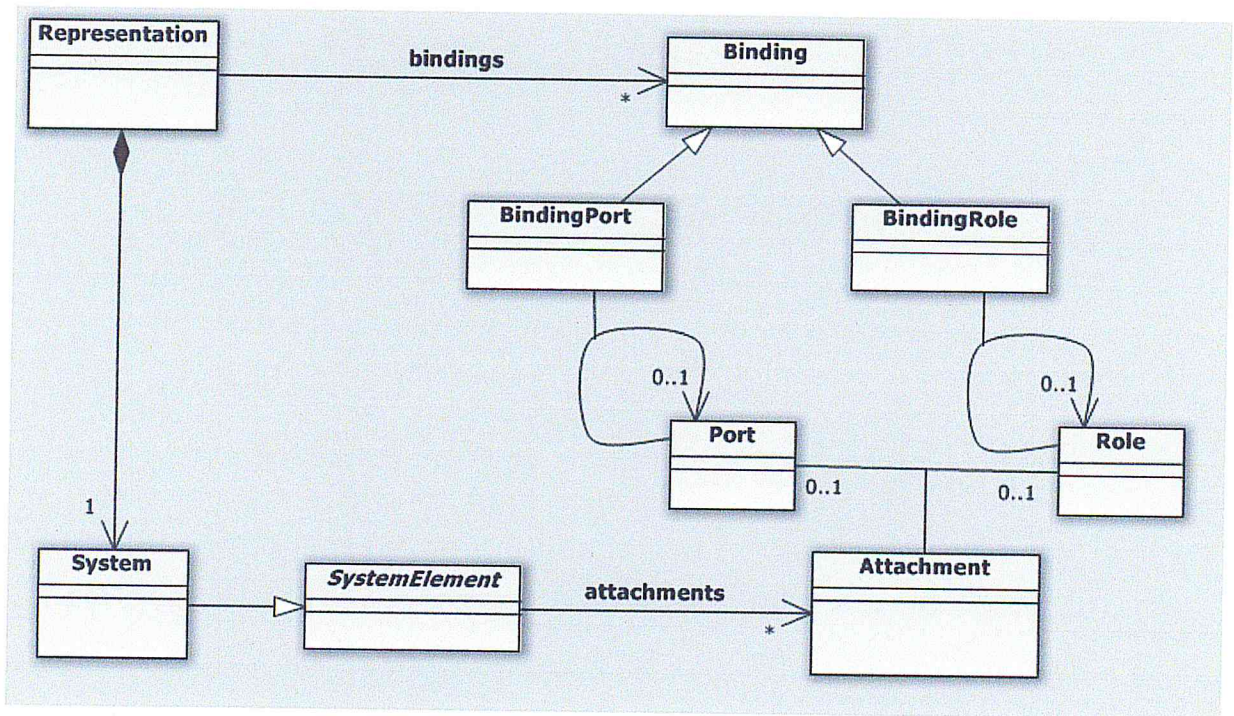

Figure 69 : diagrammes de classes pour le méta model ACME - partie 2

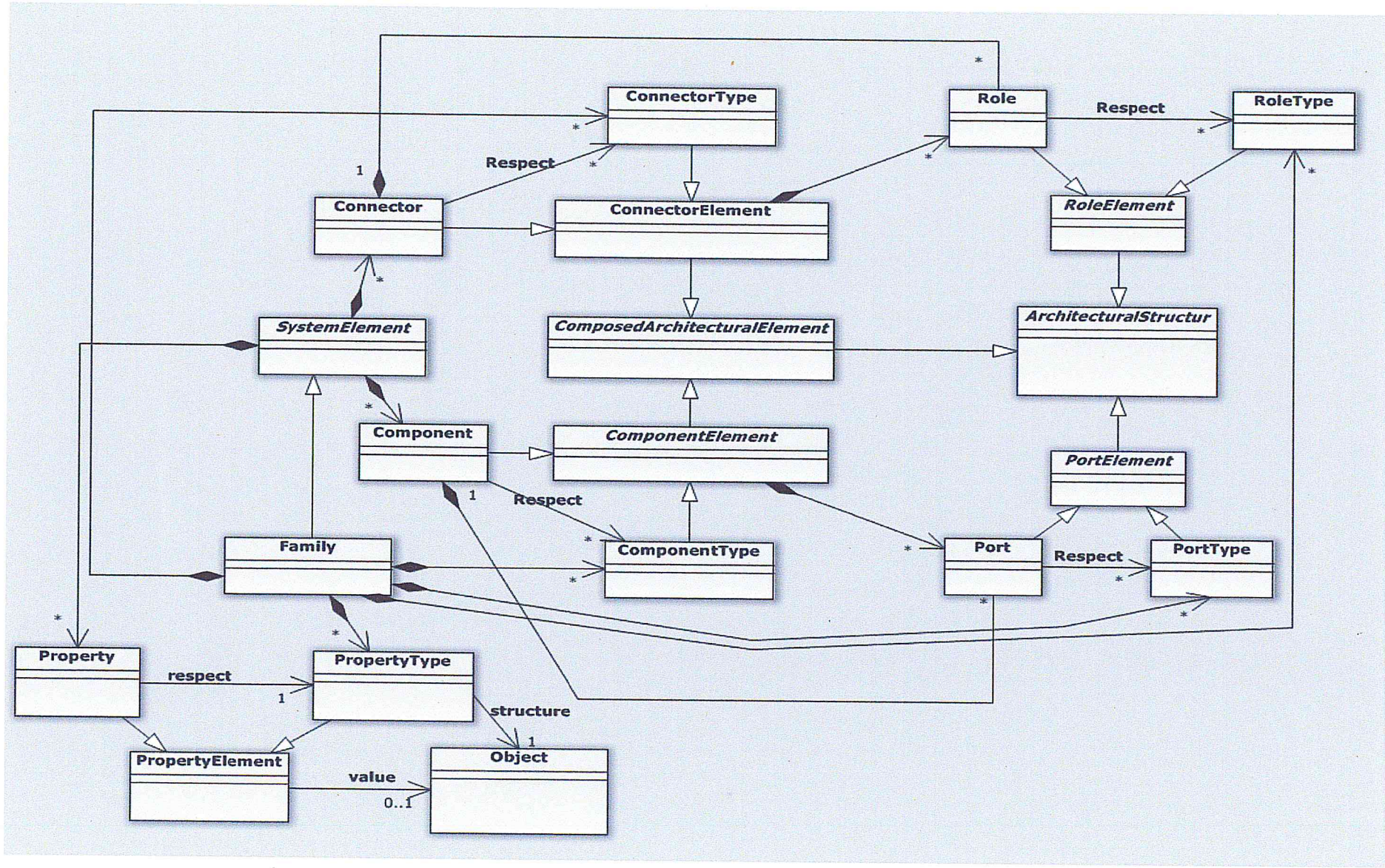

Figure 70 : diagrammes de classes pour le méta model ACME - partie 3

- **Diagramme de composants:**

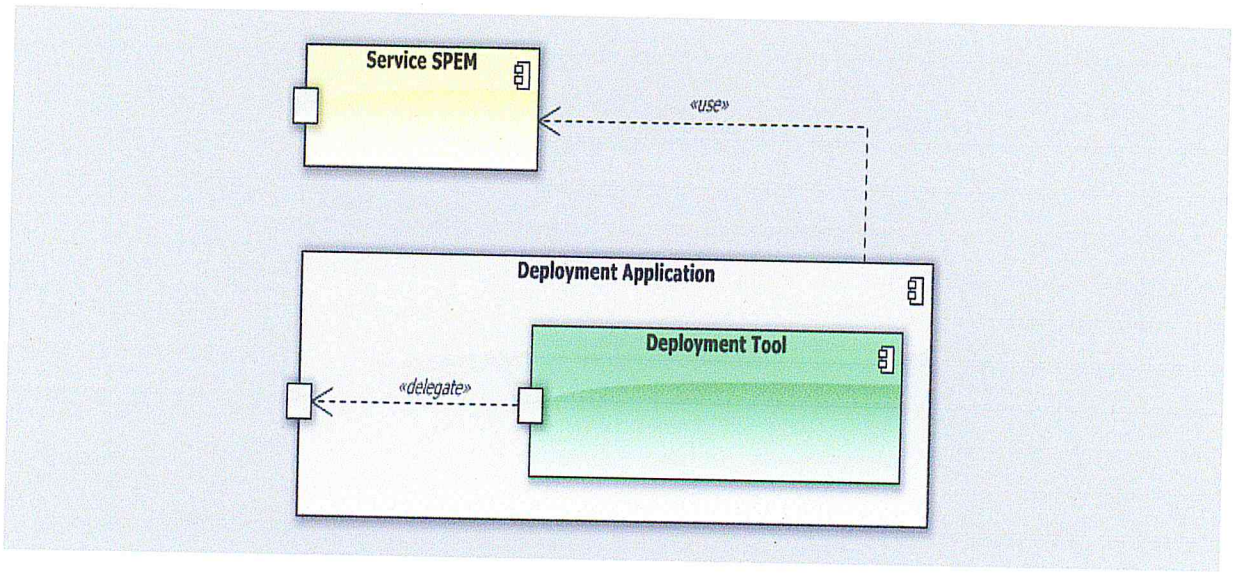

Figure 71 : diagramme de composants « Deployment Application »

- **Diagramme de classes :**

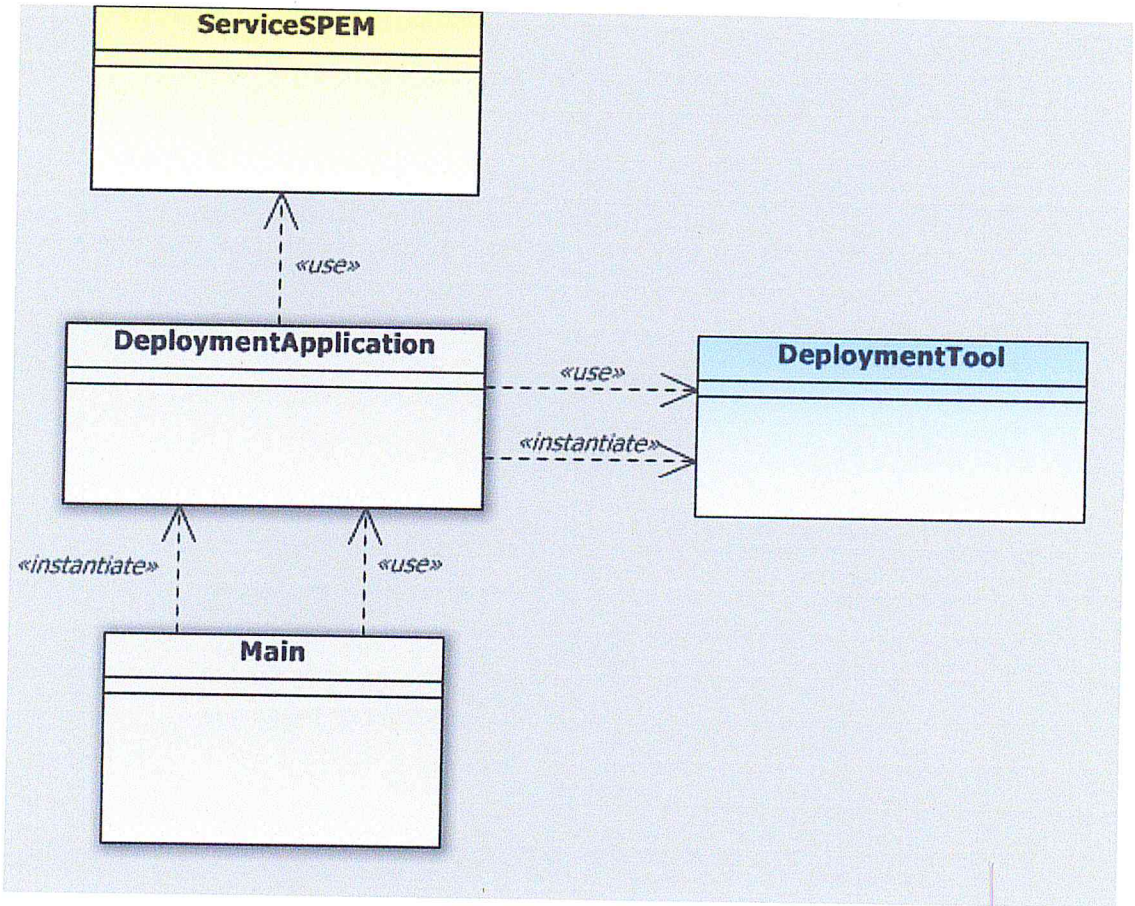

Figure 72 : diagramme de classes « Deployment Application »

- **Description diagramme de classes :**

DeploymentApplication	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
_-service : ServiceSPEM	
_- instance : DeploymentApplication	
- directoryPath : String	
Méthode	**Description**
_+ newInstance( ) : DeploymentApplication	
_+ setServiceSPEM(service : ServiceSPEM)	
_+ getServiceSPEM( ) : ServiceSPEM	
+ full_mode (configurationPath : String, limite : Integer, service : ServiceSPEM) : String	
+ distributed _mode (configurationPath : String, limite : Integer, service : ServiceSPEM) : String	

Main	
Type de classe	Classe concrète.
Super classe	Aucune.
Méthode	**Description**
_+ main(arg : String[])	Cette méthode configure l'application du déploiement d'une manière générale puis elle la lance.

5.5 Le composant « Extraction Application » :

Ce composent a été crée pour extraire des architectures logiciels, ces derniers doivent respecter les standards décrits par l'ADL ACME.

L'extraction a pour but de trouver un ensemble de configurations qui puissent satisfaire nos port data en entrées et en sortis, pour y faire ont dois suivre les étapes décrites dans la Figure 75.

- **Diagramme de composants :**

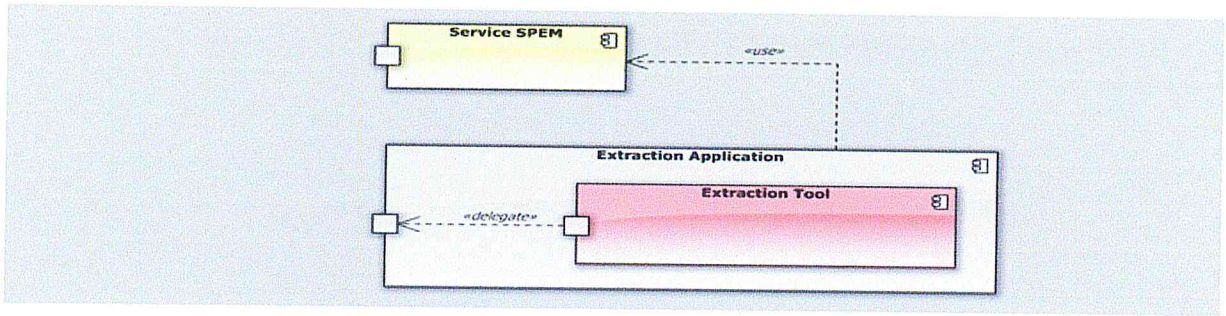

Figure 73 : diagramme de composants « Extraction Application »

- **Diagramme de classes :**

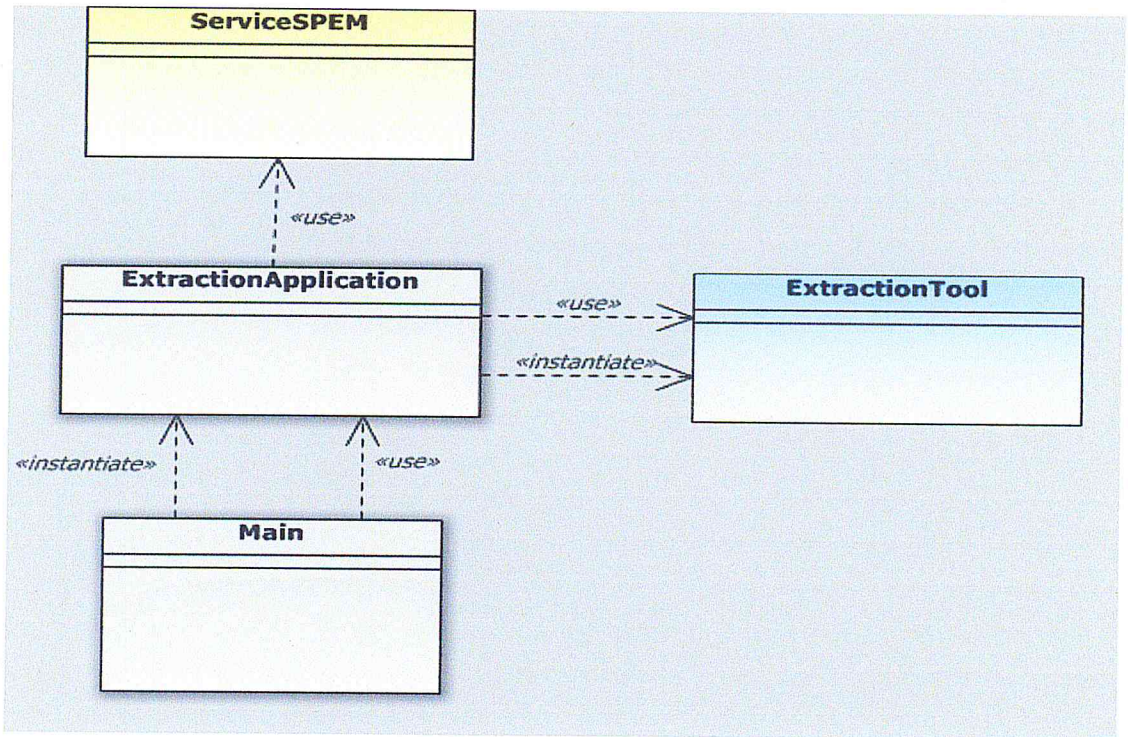

Figure 74 : diagramme de classes « Extraction Application »

- **Description diagramme de classes :**

ExtractionApplication	
Type de classe	Classe concrète.
Super classe	Aucune.
Attribut	**Description**
_-service : ServiceSPEM	
_- instance : ExtractionApplication	
- directoryPath : String	
Méthode	**Description**
_+ newInstance( ) : DeploymentApplication	
_+setServiceSPEM(service : ServiceSPEM)	
_+getServiceSPEM( ) : ServiceSPEM	
+ setDirectoryPath(directoryPath : String)	
+ unstyled_extraction_mode(dataInPorts : String [ ], dataOutPorts : String [ ]) : String	Extraction des configurations selon le mode non stylé.

Main	
Type de classe	Classe concrète.
Super classe	Aucune.
Méthode	**Description**
_+ main(args : String[])	Cette méthode configure l'application de l'extraction d'une manière générale puis elle la lance.

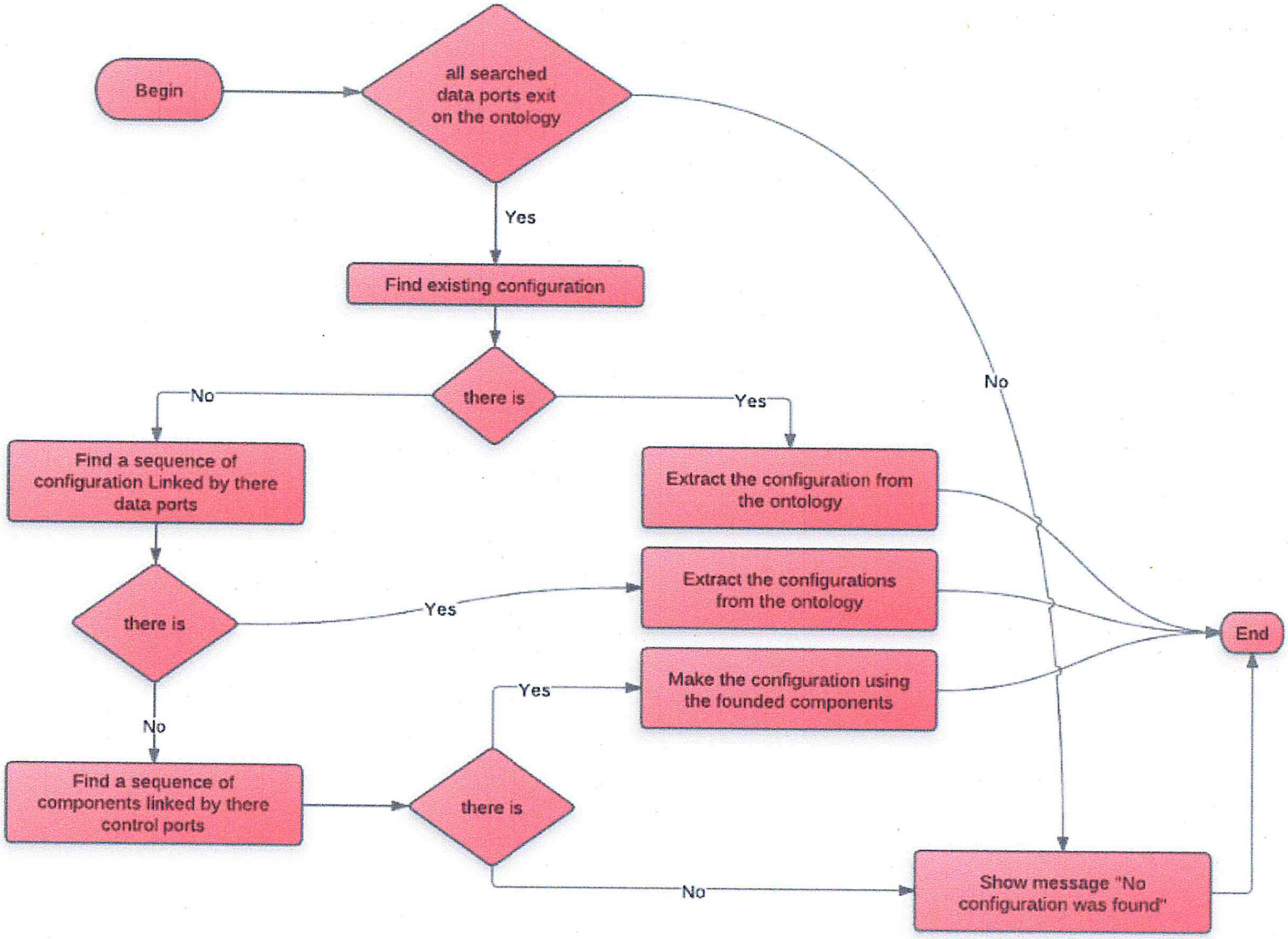

Figure 75 : algorithme principale d'extraction des configurations

6 Implémentation :

6.1 -Introduction :

Après avoir effectué la conception de notre système, nous allons à présent entamer la partie réalisation de ce dernier. Nous présenterons alors, dans un premier lieu, l'environnement de développement (langages et outils) ensuite, les différents patrons de conception utilisés lors de notre conception et nous terminerons par une petite conclusion.

6.2 Environnement de développement :

6.2.1 Langages Java :

C'est un langage de programmation orienté objet, développé par Sun Microsystems. Il permet de créer des logiciels compatibles avec de nombreux systèmes d'exploitation (Windows, Linux, Macintosh, Solaris). Java donne aussi la possibilité de développer des programmes pour téléphones portables et assistants personnels. Enfin, ce langage peut-être utilisé sur internet pour des petites applications intégrées à la page web (applet) ou encore comme langage serveur (JSP)[33].

6.2.2 Outils :

6.2.2.1 Eclipse :

Eclipse IDE est un environnement de développement intégré libre (le terme Eclipse désigne également le projet correspondant, lancé par IBM) extensible, universel et polyvalent, permettant potentiellement 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 plug-in (en conformité avec la norme OSGi) : toutes les fonctionnalités de cet atelier logiciel sont développées en tant que plugin [34].

6.2.2.2 API :

- **Jena :** est un Framework Java pour développer des applications Web sémantique. Jena fournit une collection d'outils et de bibliothèques Java pour vous aider à développer le Web sémantique et applications de données lié-, les outils et les serveurs.

Le cadre de Jena comprend:

- Un API pour la lecture, l'écriture et le traitement de données RDF en XML, N-triples et formats de tortue.
- Un API pour la manipulation de l'ontologie OWL et les ontologies RDFS.
- Un moteur d'inférence à base de règles pour raisonner avec RDF et OWL sources de données.
- Un magasin pour permettre un grand nombre de triplets RDF être efficacement stockée sur le disque.
- Un moteur de recherche compatible avec la dernière spécification SPARQL
- Des serveurs pour permettre aux données RDF à publier d'autres applications en utilisant une variété de protocoles, y compris SPARQL. [35]

- **Zip** : Il permet de manipuler des fichiers au format Zip et GZIP. L'algorithme utilisé est la méthode DEFLATE. Cette API fourni également des méthodes utilitaires pour le contrôle de l'intégrité des fichiers via les méthodes CRC-32et Adler-32. [36]

- **JDOM**: JDOM est une API pour les documents XML. Contrairement aux DOM, qui ont été conçues pour être utilisé dans une variété de langages de programmation, JDOM a été créé spécifiquement pour le langage de programmation Java. Ainsi, JDOM a été optimisé pour Java, et est plus facile à utiliser que les DOM. JDOM n'est pas basée sur DOM, et il peut être utilisé indépendamment du processeur XML sous-jacent. [37]

6.3 Les patrons de conception :

Dans cette section nous allons présenter quelques patrons de conception que nous avons soigneusement utilisé au sein de notre projet.

6.3.1 Singleton :

Motivation :

Parfois, il est important d'avoir une seule instance d'une classe. Par exemple, dans un système, il devrait y avoir qu'un seul gestionnaire de fenêtres (ou seulement un système de fichiers ou seulement un spouleur d'impression).

Habituellement les singletons sont utilisés pour la gestion centralisée des ressources internes ou externes, et ils fournissent un point d'accès global à eux-mêmes.

Le pattern singleton est l'un des modèles simples de conception: il s'agit d'une seule classe qui peut s'instancier d'elle même, afin de s'assurer qu'il ne crée pas plus d'une instance, dans le même temps, il fournit un point d'accès global à cette instance. Dans ce cas, la même instance peut être utilisée partout, étant impossible d'invoquer directement le constructeur à chaque fois.

Intention :

- Assurez-vous qu'une seule instance d'une classe est créée.
- Fournir un point d'accès global à l'objet.

Implémentation :

La mise en œuvre implique un membre statique dans le "Singleton" classe, un constructeur privé et une méthode statique publique qui renvoie une référence à l'élément statique. [38]

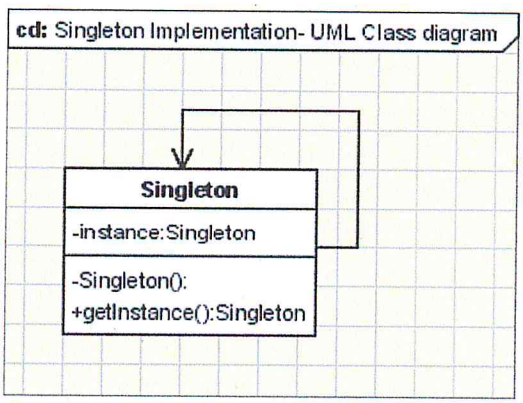

Figure 76 : exemple « patron Singleton »

6.3.2 Stratégie :

Motivation :

Il ya des situations courantes lorsque les classes ne diffèrent que par leur comportement. Dans ce cas, ça serai une bonne idée d'isoler les algorithmes dans des classes séparées afin d'avoir la possibilité de sélectionner différents algorithmes lors de l'exécution.

Intention :

Définir une famille d'algorithmes, encapsuler chacun d'eux, et de les rendre interchangeables.

Le patron stratégie permet à l'algorithme de se varier indépendamment des clients qui l'utilisent.

Implémentation :

- Le patron stratégie définit une interface commune à tous les algorithmes pris en charge. Context utilise cette interface pour appeler l'algorithme défini par une ConcreteStrategy.
- Chaque classe ConcreteStrategy implémente un algorithme.

Context :

- Contient une référence vers un objet de Stratégie.
- Peut définir une interface qui permet de stratégie accède à ses données.

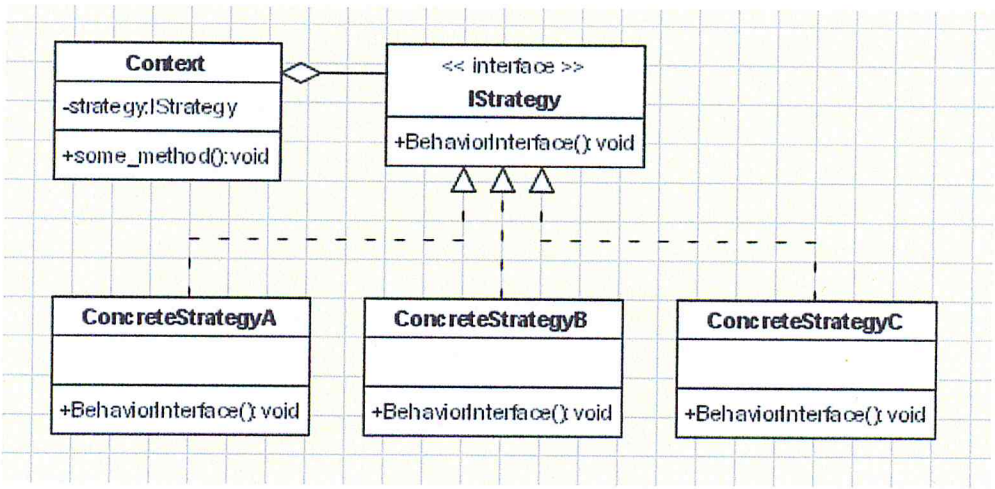

Figure 77 : exemple « patron Stratégie »

L'objet du Context contient une référence vers la ConcreteStrategy qui doit être utilisé. Quand une opération est nécessaire, alors l'algorithme est exécuté à partir de l'objet de stratégie. Le Context n'est pas au courant de l'implémentation de la stratégie. Si nécessaire, des objets d'addition supplémentaires peuvent être définis pour transmettre des données d'objet du contexte vers la stratégie.

L'objet Context reçoit les demandes des clients et les délégués à l'objet de stratégie. Habituellement, la ConcreteStrategy est créé par le client et transmise au Context. A partir de ce moment, le client interagit uniquement avec le Context.

Ce patron de conception nous a été fort utile pour la mise en œuvre de la solution concernant le problème de l'instanciation. [39]

7 Conclusion :

Dans ce chapitre nous avons présenté la conception et l'implémentation de notre système. Cette étape nous a par ailleurs, permis de nous familiariser avec les différentes fonctionnalités qu'offre notre système.

1. Introduction :

Après avoir détaillé notre conception, et après avoir fait l'implémentation il est grand temps de passer à l'étape des tests, celle-ci a pour objectif de valider l'ensemble des objectifs cités dans la phase de définition des besoins.

Tout d'abord nous allons entamer ce chapitre par la définition de quelques outils de travail, ensuite nous passerons aux jeux de test.

2. Outils utilisés :

2.1 Protégé :

Protégé est un système graphique pour la création d'ontologies. Il a été créé par le SMI (Stanford Medical Informatics) à l'université Stanford. Il est très populaire dans le domaine du Web Sémantique et au niveau de la recherche en informatique.

Protégé est un éditeur d'Ontologies distribué en open source développé en Java. C'est un éditeur hautement extensible, capable de manipuler des formats très divers, tels que: RDF, RDFS, OWL,... etc. grâce aux plugins dédiés.

Dans le modèle des connaissances de PROTEGE, les ontologies consistent en une hiérarchie de classes ou de concepts représentés sous forme générique qui ont des attributs, qui peuvent eux-mêmes avoir certaines propriétés. L'édition des listes de ces trois types d'objets se fait par l'intermédiaire de l'interface graphique, sans avoir besoin d'exprimer ce que l'on a à spécifier dans un langage formel : il suffit juste de remplir les différents formulaires correspondant à ce que l'on veut spécifier.

Aujourd'hui, il regroupe une large communauté d'utilisateurs et bénéficie des toutes dernières avancées en matière de recherche ontologique : compatibilité OWL de référence, services inférentielles, gestion de bases de connaissances, visualisation d'ontologies, etc. [40]

2.2 ACMEStudio :

AcmeStudio est un environnement de conception architecturale qui a été développé à Carnegie Mellon University. Il fournit une interface graphique qui vous permet de dessiner des architectures dans différents styles, et de manipuler et d'analyser ces conceptions.

La terminologie utilisée dans AcmeStudio est la même que celle utilisé pour la description des l'architectures Acme, et en fait AcmeStudio peut lire et écrire des descriptions Acme. [41]

1. Jeux de tests :

1.1 Instantiation :

Pour valider l'instantiation nous avons choisie un petit teste simple, qui consiste à remplir notre ontologie de domaine dans son état vide et cela par le biais de deux fichier de procédé logiciel, l'un décrit en PBOOL et l'autre en SPEM, ensuite nous allons observer le contenue de l'ontologie et nous vérifions par la même occasion la cohérence des données écrites.

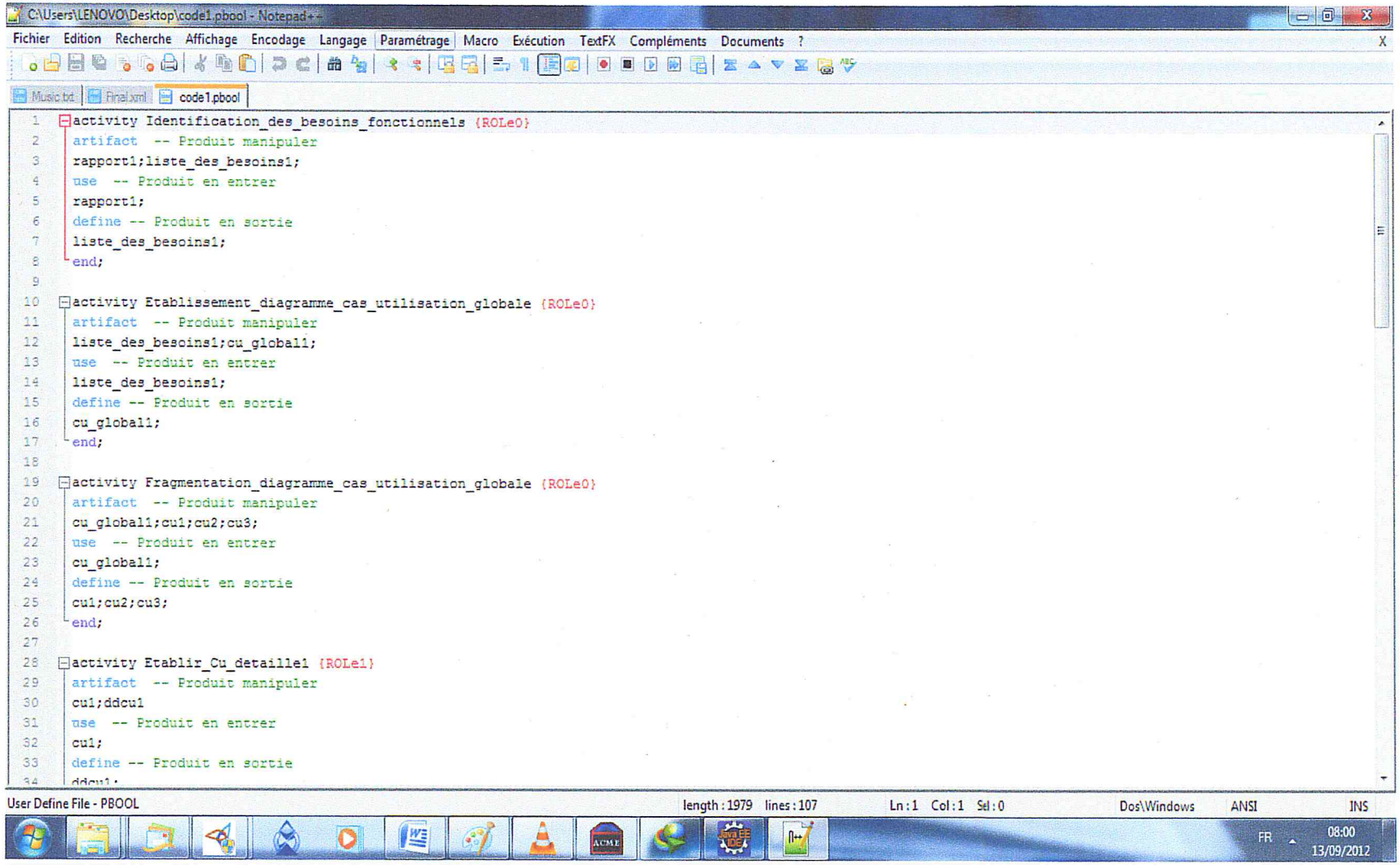

Figure 78 : exemple fichier procédé logiciel écrit en PBOOL

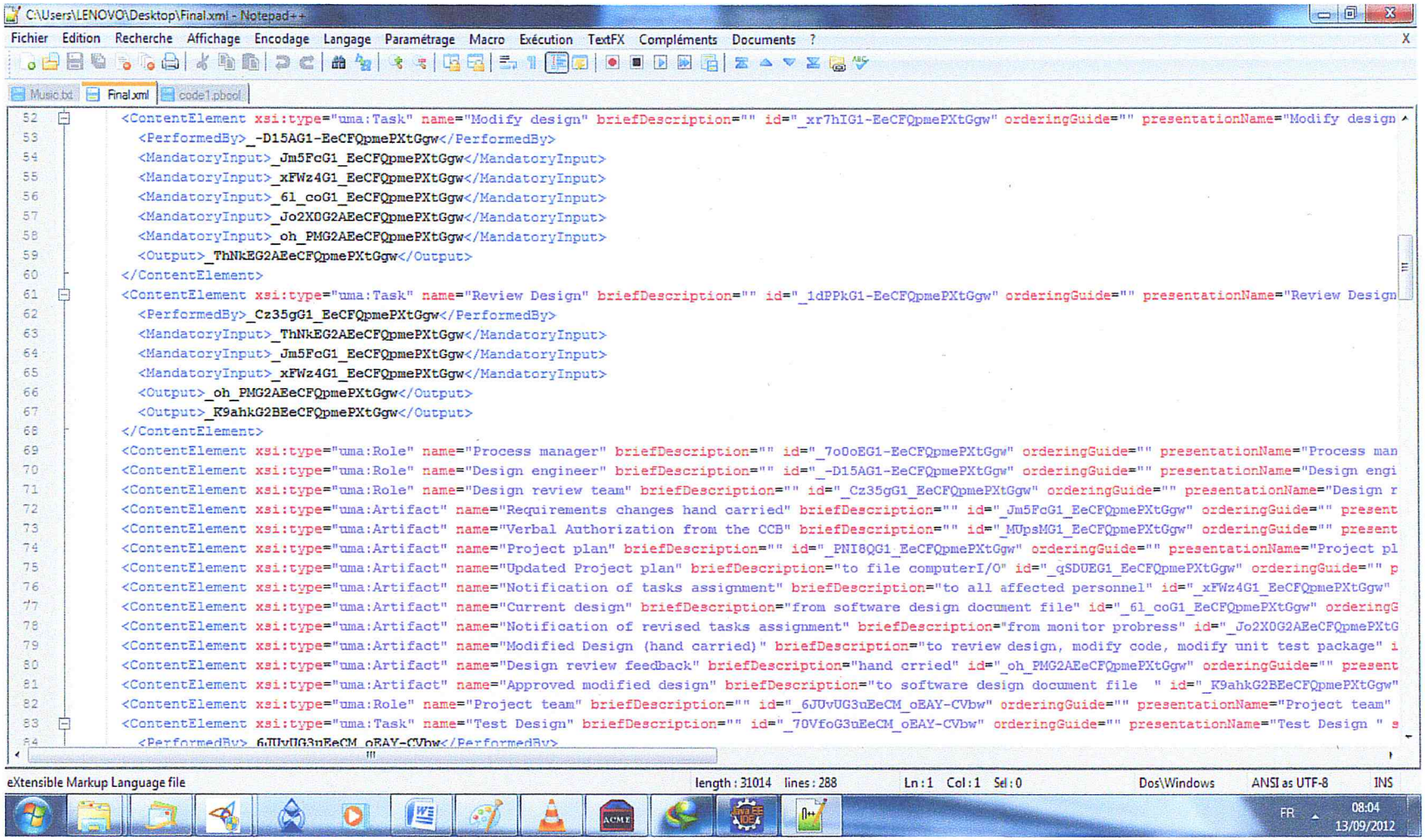

Figure 79 : exemple fichier procédé logiciel EPF

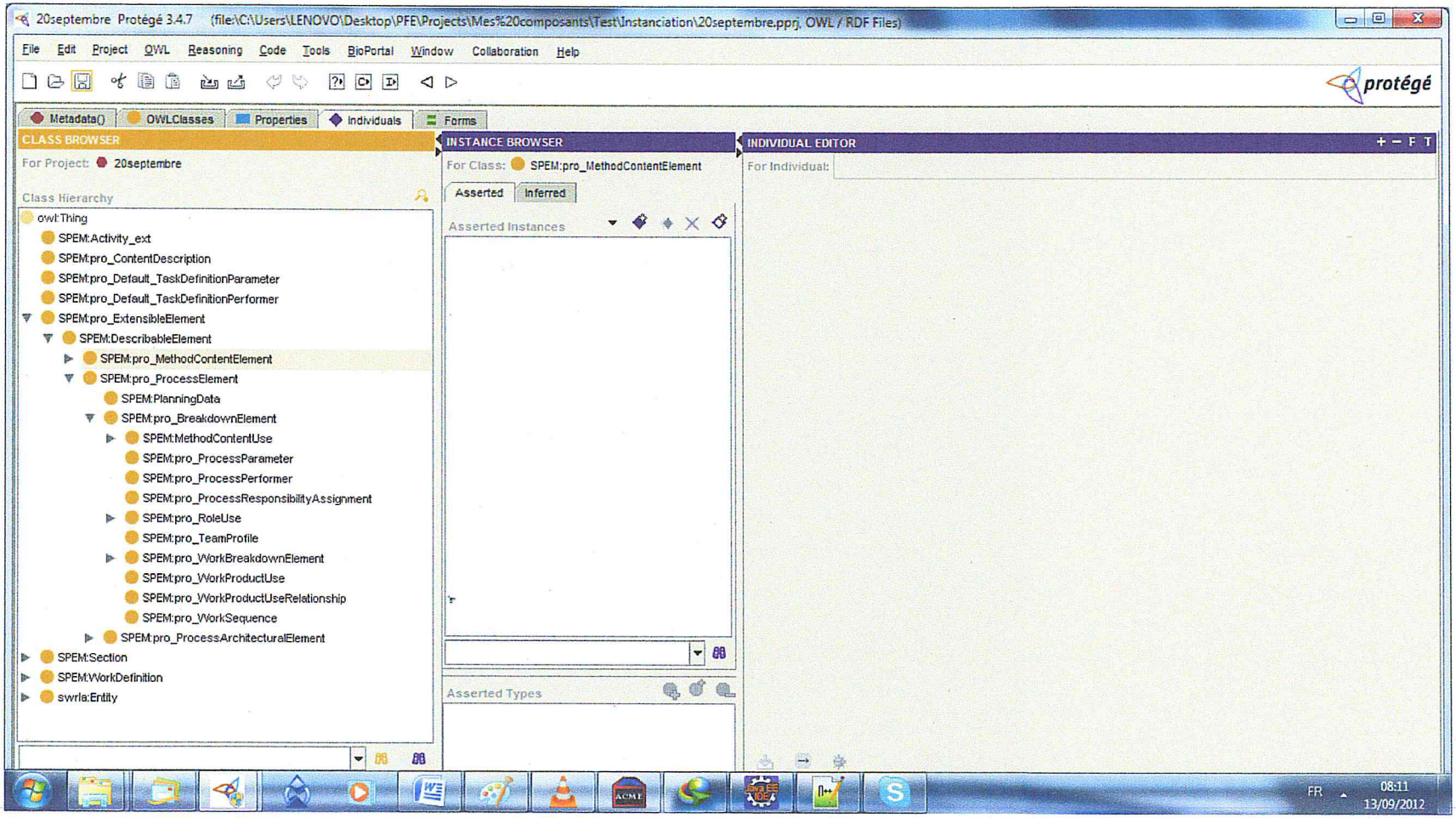

Figure 80 : SPEMOntology à l'état vide

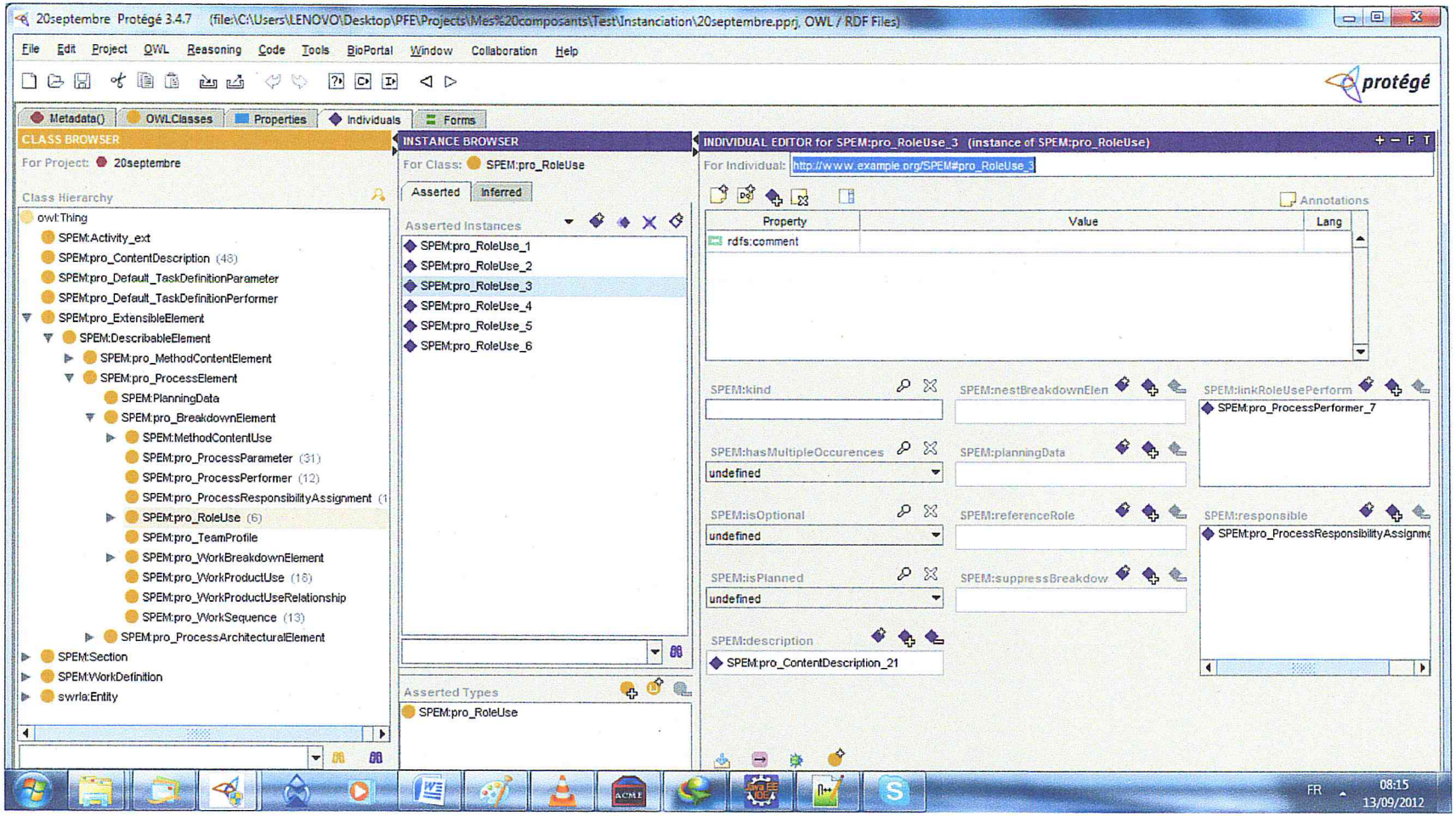

Figure 81 : SPEMOntology après le processus d'instanciation

1.2 Déploiement :

Pour valider il suffit d'utiliser une configuration et essayer de la déployer en utilisant le mode totale et le mode distribué, le résultat de cette opération dans le meilleur des cas est un fichier EPF « voir plusieurs» qui est ouvrable avec l'éditeur EPF Composer.

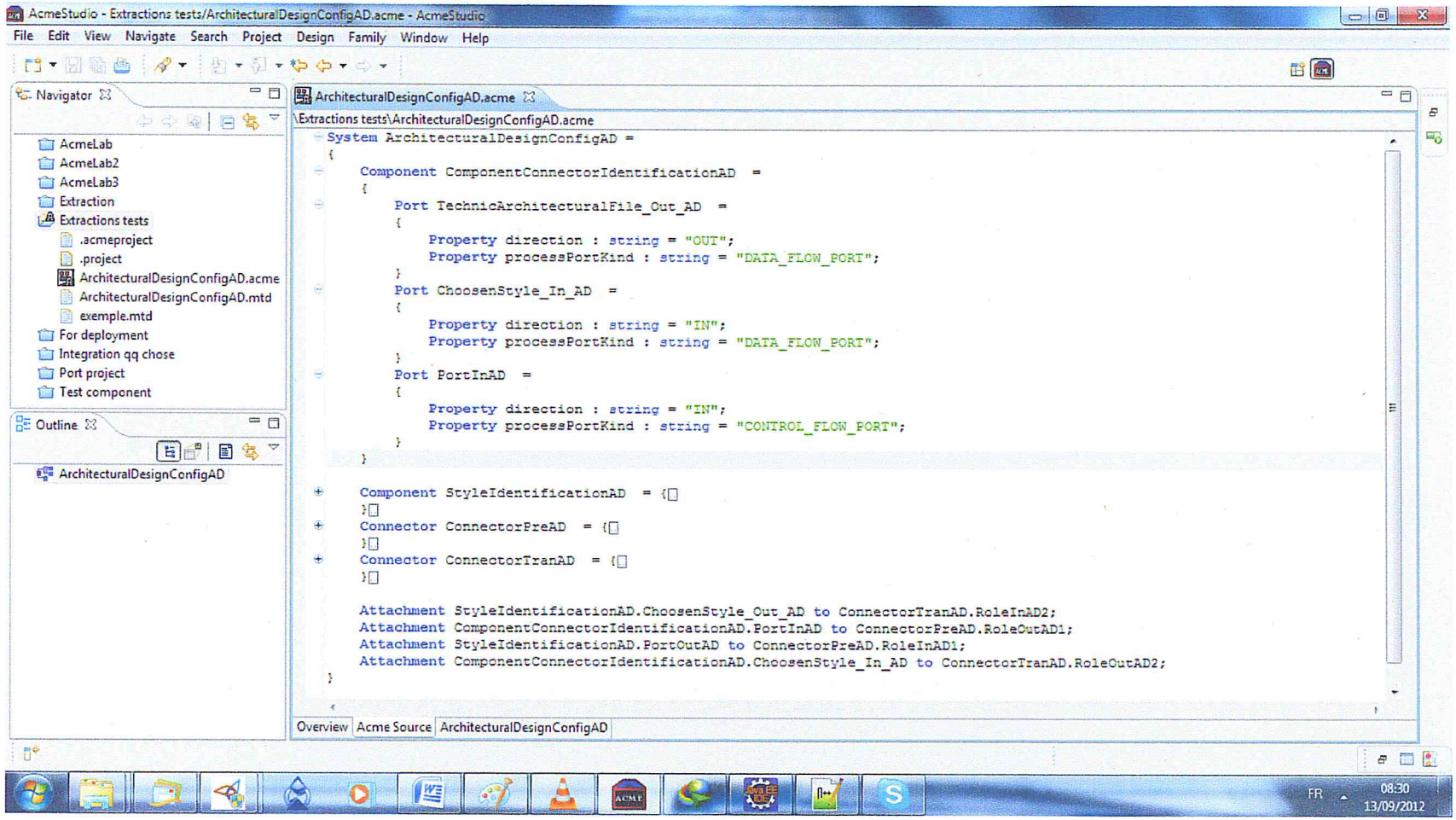

Figure 82 : configuration ACME pour le test de deploiement

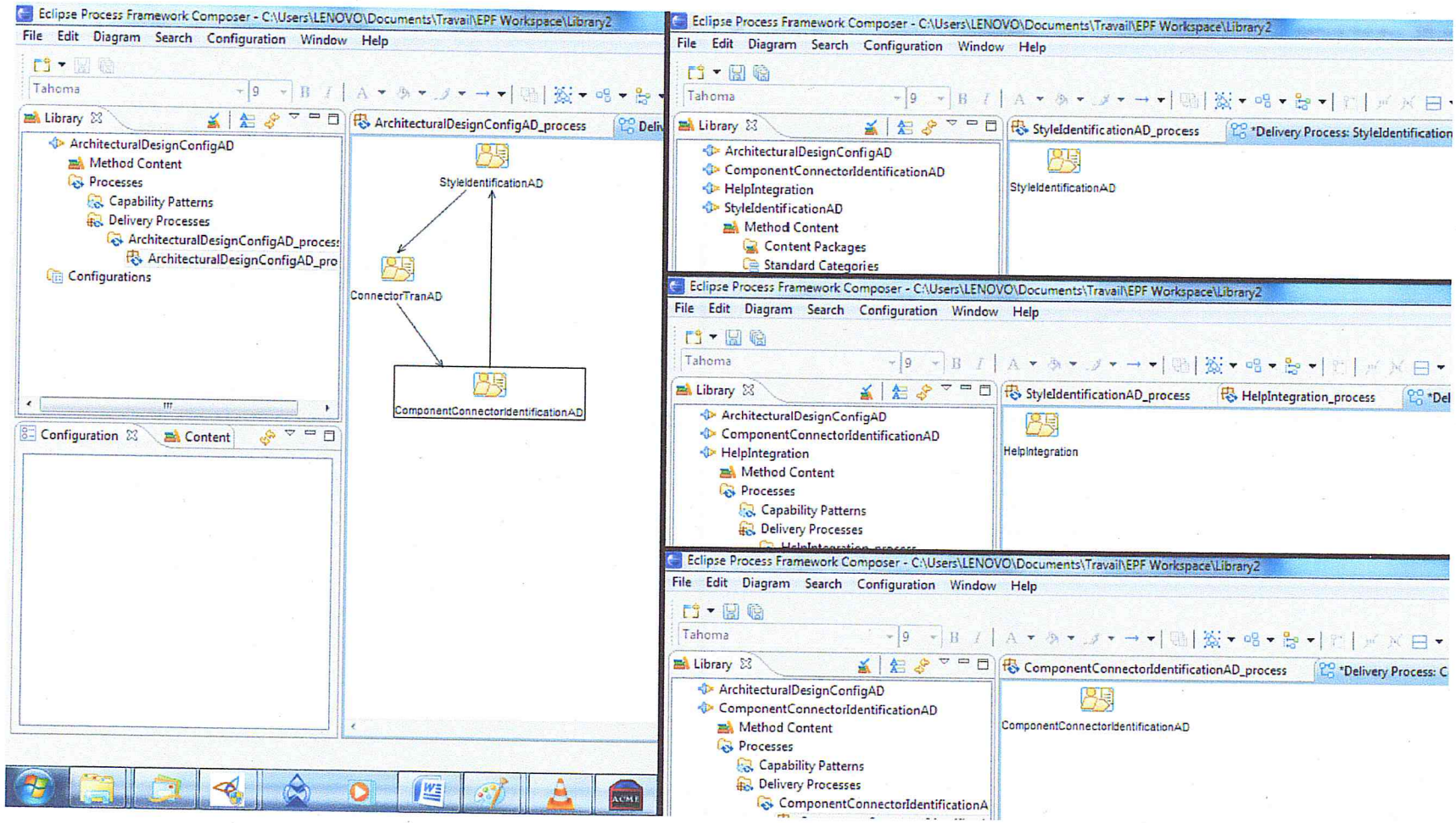

Figure 83 : déploiement total versus déploiement distribué

1.3 Extraction :

Pour valider l'extraction il faut établir un jeu de test sur le quel il faut transmettre des ports data in et des ports data out, ensuite il faut vérifier la cohérence de la configuration en résultat si elle existe bien sure.

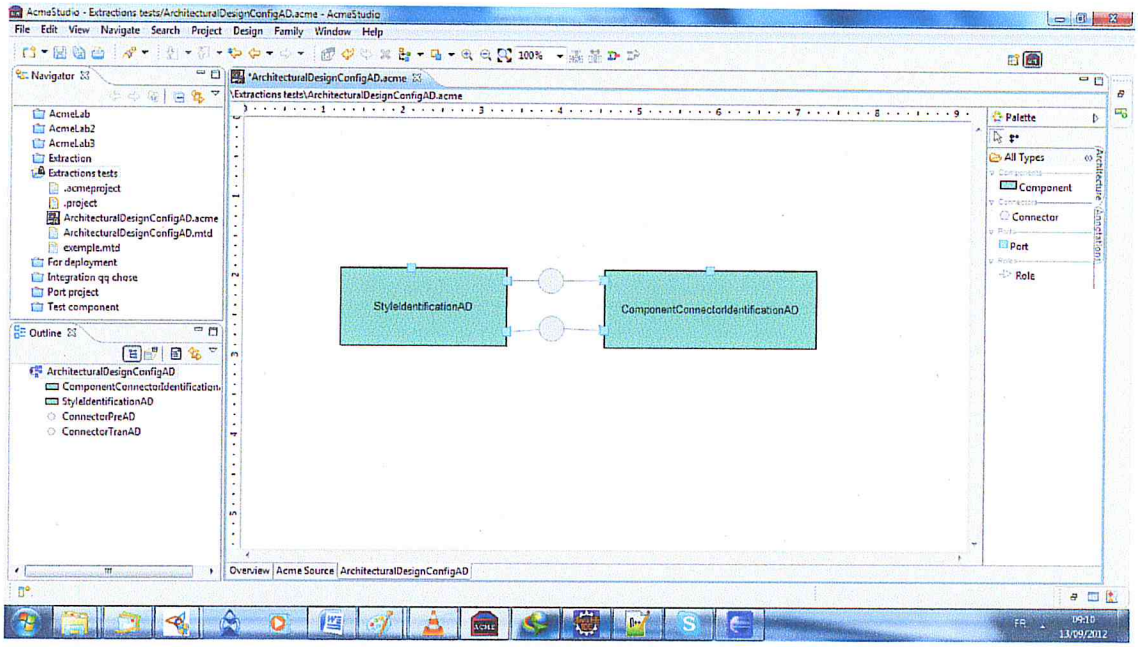

Figure 85 : exemple configuration ACME

2. Conclusion :

Dans ce chapitre dans nous avons fait un bref détour sur la façon dont ont a testé les différents fonctionnalités de notre système, bien que les tests qu'ont à fait étais tous positives cela reste à confirmer car la phase des tests est généralement attribué à une autre personne que le développeur du système lui-même.

1. Conclusion :

La modélisation et l'amélioration des PLs est un sujet de recherche d'actualité, la concurrence technologique et commerciale entre développeurs de logiciels, les pressions du marché du logiciel (produire vite, bien, et pas cher), en plus des avancées technologiques qui doivent continuellement être pris en compte, sont autant de motivations qui incitent à l'amélioration des modèles de PLs.

L'exploitation des ALs pour la prise en charge de l'aspect structure du modèle de PLs, combinée à l'exploitation d'une ontologie de domaine pour la prise en charge l'aspect expérience et savoir faire pour la modélisation et l'exécution des modèles de PLs est une des solutions adoptées. Dans ce contexte, nous avons étudié l'approche de réutilisation de PLs AoSP (Architecture oriented software Process). L'approche étudiée [20] traite un nombre significatif de domaines : Les procédés logiciels, les architectures logicielles et les ontologies de domaines. Ces trois points ont fait l'objet du chapitre I.

Plusieurs modules ont été réalisés, en effet, cette approche a fait l'objet de plusieurs PFE soit pour l'obtention de diplôme d'ingénieur ou de master en informatique à l'université SAAD DAHLEB Blida. L'objectif de notre travail est d'intégrer ces réalisations en un prototype qui fonctionne de manière cohérente. C'est dans cette perspective que nous avons étudié les réalisations effectuées par les étudiants de l'université de Blida afin de cerner leur fonctionnement et de proposer une solution d'intégration. Ce point a fait l'objet du chapitre II.

Malheureusement, les réalisations ne pouvaient pas être intégrées et cela pour le raisons suivantes :

- Les applications réalisées n'ont pas prévue la phase d'intégration.
- Une mauvaise conception, redondance et non séparation des classes fonctionnelles des classes techniques (non utilisation de patron de conception).
- Les applications ne sont pas modulaire (développé en un seul bloc ce qui rend leur réutilisabilité difficile voir impossible).
- L'ontologie de domaine était modifiée dans chaque projet et ne pouvais être exploité par toutes les applications en même temps.
- Des algorithmes de recherche et de parcours sont trop simples, et pas très efficaces, car ils ne traitent pas tous les cas possibles.

2. Perspectives :

Pour réaliser notre travail, nous avons développé notre application en plusieurs modules indépendants et qui fonctionne ensemble de manière cohérente, ces applications sont :

- Une application qui permet la capitalisation des connaissances par l'instanciation de SPEMOntology. Deux instantiateurs spécifiques à deux langages de modélisation de PLs ont été développés, l'instantiateur PBOOL et l'instantiateur EPF. Cette application fait intervenir au minimum 20 classes.

- Une application fait la recherche et l'affichage d'une configuration PL à partir des connaissances capitalisées. La configuration résultat est décrite à l'aide de l'ADL ACME, Elle fait intervenir environ 10 classes.

- Une application qui fait le déploiement de la configuration PL. Le déploiement passe par la phase de recherche de connaissances composant et la phase d'adaptation de connaissances connecteur, l'application fait intervenir environ 20 classes.

Le travail réalisé est très consistant, cependant, certaines améliorations peuvent être effectuées et qui sont comme suit :

- La mise en place des composants GUI pour chaque application, car nous nous somme focalisés plus sur le fonctionnement que sur l'interface.
- Mise ne place de composant qui fait les mises à jours des applications.
- Amélioration du composant service SPEM pour une manipulation pertinente de SPEMOntology (restauration, reprise après pannes).

Annexe A : Le langage Java

1. Présentation de JAVA :

Java est un langage de programmation compilé et interprété, orienté objet. Il a été créé en 1991 par Sun pour pallier aux contraintes que posait le C++, et cela dans le but de développer des logiciels pour l'électronique grand public (appareils ménagers). La syntaxe de ce langage est assez proche du C et est plus claire que le C++. L'objectif pour lequel JAVA a été conçu nécessite certaines caractéristiques comme : La robustesse, la compatibilité, la facilité de programmation et la petite taille du runtime ou des codes générés. JAVA présente beaucoup d'avantages grâce auxquels il a mérité son succès:

- Orienté objet : la brique de base du programme est donc l'objet, instance d'une classe.
- Indépendant des architectures.
- **Portable :** une fois le programme est compilé, il pourra fonctionner aussi bien sous des stations Unix, que sous Windows ou autre.
- **Interprété :** par la machine virtuelle de JAVA (JVM).
- Gère la mémoire automatiquement.
- Possède une API très riche : différents packages permettent d'accéder au réseau, aux entrées/sorties et aux différents composants graphiques.
- Distribué, simple, robuste, sécurisé, multithread et dynamique.

Sa portabilité et son indépendance des architectures ou plates-formes constituent son principal atout et grâce auquel il peut fonctionner sur différentes plates-formes et sous différents systèmes d'exploitation. Ces deux caractéristiques sont dues à la machine virtuelle Java (JVM). A coté de ses nombreux avantages, JAVA présente un seul inconvénient : la lenteur lors de la conversion des instructions de la JVM en instructions compréhensibles par la machine. Cependant deux solutions sont mises en place pour éviter cet inconvénient :

- Compiler le code pour une machine spécifique, à ce moment là le programme n'est plus portable.
- Doter la JVM d'un JIT (Just In Time compiler) qui traduit le code JVM d'une classe dès sa première utilisation en code spécifique à la machine.

2. JAVA Virtual Machine (JVM) :

La machine virtuelle appelée aussi interpréteur, joue le rôle d'un traducteur. Elle traduit en Bytecode le code compilé (ensemble de fichiers de classes) qui est sous forme de pseudo-code et non en code machine, et cela afin d'être exécuté sur n'importe quelle plate-forme matérielle et logicielle : que l'on soit sur un Pentium, un PowerPC, un Sparc

ou sur un Alpha, sous Windows, MacOs, Solaris ou Linux, etc. Cependant, la présence de la JVM au niveau de la plate forme est nécessaire à l'exécution des programmes JAVA sur une plate -forme quelconque.

Les JVMs sont fournies soit par le JDK (Java Developpement Kit), soit par les navigateurs ou bien par les environnements de développement spécifiques tel que Borland JBuilder. Des JVM capables d'exécuter du code Java, peuvent être fournit également par certaines solutions telle Java Plug-in de Sun. La JVM peut être obtenue à partir du site de Sun Microsystems « Oracle actuellement ».

3. Environnement de développement

Il existe plusieurs environnements de développement pour JAVA tel le JDK (Java Development Kit) qui est gratuit, mais la plus part sont payants. Ces environnements mettent à la disposition des développeurs : un éditeur intégré, un compilateur, un débuggeur sophistiqué et de nombreux générateurs de code (surtout concernant les interfaces graphiques).

3.1 Le Java Development Kit (JDK)

Le JDK est un environnement gratuit comportant un compilateur et une machine virtuelle (minimal et suffisant). Il est téléchargeable à partir du site de Sun. Il comporte l'ensemble des éléments qui ont pour but le développement, la mise au point et l'exécution des programmes Java. Il peut être considéré comme un ensemble d'outils plus un jeu de classes et de service plus un ensemble de spécifications. Il existe plusieurs versions de JDK, il est important de connaître la version employée car les classes disponibles peuvent être différentes d'une version à une autre.

4. Application Programming Interface (API)

Une API fournit aux programmeurs les outils de base nécessaires afin de leur faciliter le travail. Elle se compose d'un ensemble de fonctions, routines et méthodes. Par l'API de JAVA le programmeur va pouvoir accéder à toutes les ressources de la machine, et celles du réseau Internet. Elle est divisée en packages.

Plusieurs packages sont fournis en standard avec JAVA tels :

- java.awt et javax.swing pour les interfaces graphiques,
- java.applet pour la réalisation des applications qui peuvent s'exécuter dans un navigateur Internet.
- java.sql pour accéder aux bases de données.
- java.beans pour la programmation orientée composants (les Java

Beans sont des composants logiciels réutilisable tel NetBeans ou JBuilder).

- java.rmi pour la mise en place des applications réparties telles les applications client/serveur.
- D'autres packages se trouvent dans d'autres applications et il faudra les importer tels : Les API PROTEGE et JENA pour la manipulation des ontologies.

5. Eclipse

Eclipse est un environnement de développement intégré (Integrated Development Environment), développer par I.B.M, dont le but est de fournir une plateforme modulaire pour permettre de réaliser des développements informatiques. Eclipse utilise énormément le concept de modules nommés "plugins" dans son architecture.

Hormis le noyau de la plateforme nommé "Runtime", tout le reste de la plateforme est développé sous la forme de plugins. Ce concept permet de fournir un mécanisme pour l'extension de la plate-forme et ainsi fournir la possibilité à des tiers de développer des fonctionnalités qui ne sont pas fournies en standard par Eclipse. Les principaux modules fournis en standard avec Eclipse concernent Java mais des modules sont en cours de développement pour d'autres langages notamment C++,

Cobol, mais aussi pour d'autres aspects du développement (base de données, conception avec UML, ...). Ils sont tous développés en Java soit par le projet Eclipse soit par des tiers commerciaux ou en open source. Les modules agissent sur des fichiers qui sont inclus dans l'espace de travail (Workspace). L'espace de travail regroupe les projets qui contiennent une arborescence de fichiers. Bien que développé en Java, les performances à l'exécution d'Eclipse sont très bonnes car il n'utilise pas Swing pour l'interface homme-machine mais un toolkit particulier nommé SWT associé à la bibliothèque JFace. SWT (Standard Widget Toolkit) est développé en Java par IBM en utilisant au maximum les composa nts natifs fournis par le système d'exploitation sous jacent. JFace utilise SWT et propose une API pour faciliter le développement d'interfaces graphiques.

Les points forts d'Eclipse :

Eclipse possède de nombreux points forts qui sont à l'origine de son énorme succès dont les principaux sont :

- Support de plusieurs plateformes d'exécution : Windows, Linux, MacOs X, ..
- Une plate-forme ouverte pour le développement d'applications et extensible grâce à un mécanisme de plug-ins.

- Plusieurs versions d'un même plugin peuvent cohabiter sur une même plateforme.
- Un support multi langage grâce à des plug-ins dédiés : Cobol, C, PHP, C#, ...
- Malgré son écriture en Java, Eclipse est très rapide à l'exécution grâce à l'utilisation de la bibliothèque SWT.
- Un historique local des dernières modifications.
- Une exécution des applications dans une JVM dédiée sélectionnable avec possibilité d'utiliser un débogueur complet (points d'arrêts conditionnels, visualiser et modifier des variables, évaluer les expressions dans le contexte d'exécution, changement du code à chaud avec l'utilisation d'une JVM 1.4, ...)
- Propose le nécessaire pour développer de nouveaux plug-ins. La plate-forme est entièrement internationalisée dans une dizaine de langue sous la forme d'un plug-in téléchargeable séparément.

Liste des figures

Liste des figures

Liste des figures

Liste des tableaux

Références bibliographiques

[1] Walt Scacchi, Process Models in Software Engineering, J.J. Marciniak (ed.), *Encyclopedia of Software Engineering, 2nd Edition*, John Wiley and Sons, Inc, New York, December 2001.

[2] Alfonso Fuggetta. 2000. Software process: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (ICSE '00). ACM, New York, NY, USA, 25-34. DOI=10.1145/336512.336521 http://doi.acm.org/10.1145/336512.336521

[3] Benoît Combemale, Ingénierie Dirigée par les Modèles (IDM) État de l'art, 2008 http://hal.inria.fr/docs/00/37/15/65/PDF/mde-stateoftheart.pdf

[4] Alan W. Brown: Model driven architecture: Principles and practice. Software and System Modeling 3(4): 314-327 (2004)

[5] Lonchamp J. "A structured conceptual and terminological framework for software process engineering", in Proc. Of the 2nd Inter. Conf. on Software Process, Berlin, 1993, IEEE Computer Society Press, Pages: 41-53.

[6] *Kamal Zuhairi Zamli*, PROCESS MODELING LANGUAGES: A LITERATURE REVIEW, *Malaysian Journal of Computer Science, Vol. 14 No. 2, December 2001, pp. 26-37.*

[7] Silvia, T. Acuña, Xavier Ferré. Software Process Modeling. World Multi conference on Systemics, Cybernetics and Informatics (SCI), Orlando USA. 2001. DOI:10.1.1.23.5438

[8] Lizbeth GALLARDO, Une approche à base de composants pour la modélisation des procédés logiciels, DEA en informatique, université de Grenoble, 2000.
www-adele.imag.fr/Les.Publications/reports/DEA2000Gal.pdf

Références bibliographiques

[9] BENOÎT COMBEMALE, Approche de méta modélisation pour la simulation et la vérification de modèle : Application à l'ingénierie des procédés, thèse de doctorat, l'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE, 2008.

[10] Zamli, K. Z., Mat Isa. N. A., A survey and analysis of process modeling languages. Malaysian Journal of Computer Science, ISSN 0127-9084 Vol.17 No. 2, 2004. pp. 68-89. http://ejum.fsktm.um.edu.my/ArticleInformation.aspx?ArticleID=310

[11] Reda Bendraou, Marie-Pierre Gervais, and Xavier Blanc: UML4SPM: A UML2.0-Based Meta model for Software Process Modeling. MoDELS 2005: 17-38

[12] Cregut, X., Coulette, B.: PBOOL: an Object-Oriented Language for Definition and Reuse of Enactable Processes. J. Software - Concepts and Tools, vol. 18, 1997, No. 2, pp. 47-62.

[13] Kellner, M. I, Feiler, P. H, Finkelstein, A, Katayama, T., Osterweil, L. J, Penedo, M.H, Dieter Rombach, H., (1996), 'ISPW- 6 Software Process Example', 6th International Software Process Workshop.

[14] Charlotte Hug, Méthode, modèles et outil pour la méta-modélisation des processus d'ingénierie de systèmes d'information, thèse de doctorat, UNIVERSITÉ JOSEPH FOURIER - GRENOBLE I, 2009.
http://tel.archives-ouvertes.fr/docs/00/43/76/92/PDF/These_Charlotte_Hug.pdf

[15] W. Curtis, M. I. Kellner and J. Over. "Process Modeling". Communication of ACM, 35 (9), 1992, pp. 75-90.

[16] Object Management Group (OMG), (2008), 'Software and Systems Process Engineering Meta Model (SPEM)', v2.0 http://www.omg.org/cgi-bin/docFormal/2008-04-01.

Références bibliographiques

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

[18] Lortal Gaëlle, Etat de l'art Ontologies et Intégration/Fusion d'ontologies, 2002.

[19] 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.

[20] Fadila Aoussat, Mohamed Ahmed-Nacer, Mourad Oussalah: Reusing Approach for Software Processes based on Software Architectures. ICEIS (1) 2010: 366-369

[21] Fadila Aoussat, Mohamed Ahmed Nacer, Mourad Oussalah, Approach for software processes models reusing based software architecture.
WMSCI'10, Orlendo, USA.

[22] Kheddouci Anissa, Maouchi Hassna, Inférence et Extraction d'architectures de PL respectant des styles de PLs. Ingéniorat d'état en informatique, Université SAAD DAHLEB Blida, 2010-2011.

[23] Mehiout Kamel, Mazouni Boualam, Instanciation automatique de SPEMOntology à partir de modèles EPF, Master en informatique, Université SAAD DAHLEB Blida, 2010-2011.

[24] Boutarouk Younes, Korrichi Abdenour, Déploiement d'architectures de procédés logiciels en language SPEM, Master en informatique, Université SAAD DAHLEB Blida, 2010-2011.

[25] Belhhemri Djaouida, Sidoumou Aicha, Raisonnement et Inférence de fragments de précédés logiciels. Ingéniorat d'état en informatique, Université SAAD DAHLEB Blida, 2009-2010.

[26] Mahiout Mounir, Guenawi Fateh, Instanciation automatique de SPEMOntology à partir de modèle PBOOL+, Ingéniorat d'état en informatique, Université SAAD DAHLEB Blida, 2008-2009.

[27] Selt Rachid, Ontology OWL pour l'inférence d'architectures de procédés logiciels, Ingéniorat d'état en informatique, Université SAAD DAHLEB Blida, 2009-2010.

Références bibliographiques

[28] Amara Safia, Modélisation d'architectures de procédés logiciels avec l'ADL ACME, Ingéniorat d'état en informatique, Université SAAD DAHLEB Blida, 2009-2010.

[29] Chikhi Imane, Djouadi Imane, Génération d'une ontologie de domaine à partir du méta modèle SPEM. Ingéniorat d'état en informatique, Université SAAD DAHLEB Blida, 2008-2009.

[30] EPF, Eclipse Process Framework, http://www.eclipse.org/epf/

[31] ATL, Atlantis Transformation Language, liste des modules d'ATL. http://wiki.eclipse.org/ATL/User_Guide_-_Overview_of_the_Atlas_Transformation_Language

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

[33] http://www.futura-sciences.com/fr/definition/t/internet-2/d/java_485/

[34] http://www.techno-science.net/?onglet=glossaire&definition=517

[35] http://jena.apache.org/

[36] http://cyberzoide.ftp-developpez.com/java/zip.pdf

[37] http://dret.net/glossary/jdom

[38] http://www.oodesign.com/singleton-pattern.html

[39] http://www.oodesign.com/strategy-pattern.html

[40] Mlles NADJI Khadidja et MEZZI Melyara, Conception et réalisation d'un système de construction d'Ontologies par fusion des Ontologies existantes, master en informatique, université de Blida, promotion 2011.

[41] http://www.cs.cmu.edu/~acme/zip/ASTutorial1.zip