République Algérienne Démocratique et Populaire

Ministre de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Saad Dahleb ,Blida

USDB

Faculté des Sciences

Département d'Informatique

Mémoire pour l'obtention

D'un diplôme de Master 2 en informatique

Option : Ingénierie de Logiciel

Sujet :

Déploiement d'une architecture de procédé logiciel dans le langage SPEM

Présenté par :

- BOUTAROUK YOUNES
- KORICHI ABDENOUR

Promoteur : Mme F.AOUSSAT

Soutenu le :

Devant le jury composé de :

Président : M.Benattiti

Examinateur : M. Toubaline

Examinateur : M. Ammeur

Promotion :2010-2011

Remerciements

Avant tout, nous remercions ALLAH le Tout Puissant qui nous a donné le courage, la volonté et la patience pour faire ce travail.

Au terme de ce projet, il nous est agréable de remercier vivement tous ceux qui, grâce à leur aide précieuse, ont permis à sa concrétisation.

Nous remercions particulièrement Mlle Aoussat, pour avoir proposé et dirigé ce travail ainsi que pour ses conseils, ses orientations, et sa gentillesse.

Nous remercions tous les membres du jury pour avoir accepté de juger notre mémoire et de nous honorer de leur présence.

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

الملخص

الغرض من عمليات البرمجيات ، إنتاج برامج مصممة وفقا للاحتياجات مع خفض التكاليف وصنع البرامج في غضون فترة زمنية معقولة ، واحدة من الأساليب المستخدمة من أجل نمذجة عمليات البرمجيات ذات ميزة وإعادة استخدامها. إعادة استخدام الخبرة و المعرفة في هذا المجال . لزيادة إعادة الاستخدام و من أجل الحصول علي عمليات برمجيات نموذجية نقترح استخدام مبادئ هندسة البرمجيات.وبالتالي فإن الهدف من عملنا هو تنفيذ جزء من هذا النهج، و بعبارة أخرى ، مهمتنا هي نشر بنى عمليات البرمجيات.

Résumé

L'objectif des procédés logiciels (PLs) et de produire des logiciels adaptés aux besoins en réduisant les coûts de maintenance et rendant les logiciels productifs dans un délai raisonnable, une des approches utilisées pour modéliser des PLs de qualité et la réutilisation des PLs.

Réutiliser le savoir faire et les connaissances de ce domaines, capitalisés pendant plusieurs décennies de recherche est l'objectif visé. Pour augmenter la réutilisabilité et pour une meilleure modélisation des procédés logiciels nous proposons d'exploiter les principes des architectures logicielles.

Par conséquence l'objectif de notre travail est l'implémentation d'une partie de cette approche, en d'autres termes, notre travail consiste à déployer des architectures de PL.

Summary

The purpose of the software processes (SP) is to produce software tailored to reducing maintenance costs and making productive software within a reasonable time, one of the approaches used to model quality of SP and reuse of SP. Reuse the expertise and knowledge in this field, funded over several decades of research is the goal. To increase the reusability and better process modeling software we propose to use the principles of software architectures.

Sommaire

Chapitre 5 : Implémentation.

Introduction générale

L'objectif du génie logiciel et de produire des logiciels adaptés aux besoins en réduisant les coûts de maintenance et en rendant les logiciels productifs dans un délai raisonnable.

L'utilisation des procédés logiciels va dans ce sens. Un procédé logiciel est la description de l'enchainement des étapes à suivre pour réaliser puis maintenir un produit logiciel. Le PL doit assurer le développement des logiciels de qualité dans les délais, en optimisant les ressources utilisées.

Une approche de modélisation de PL à base d'architecture logicielle a été proposée [1], l'objectif principal de cette approche est la réutilisation des PLs. Cette réutilisation consiste à extraire puis déployer des architectures de PLs.

L'objectif de notre travail est l'implémentation d'une partie de cette approche. Notre travail consiste à déployer des architectures de PL, en exploitant des connaissances existantes préparées pour le déploiement.

Ce déploiement se fait en exploitant des configurations de PL décrites en langage de description d'Architecture Log ACME. Ainsi que des fichiers XML décrivant les composants et connecteurs de PLs. Le résultat doit être décrit en EPF Composer.

EPF composer est une plateforme java qui vise à produire des processus logiciels personnalisables, avec un contenu processus exemplaire tel que « open UP », supportant une large variété de types de projets et styles de développement,

Le choix de EPF Composer est justifié par le fait que EPF composer respecte le méta modèle SPEM « Software Process Engineering Meta-model » qui est un standard définie par l'OMG. Aussi, EPF Composer est gratuit et accessible à tout le monde.

Pour réaliser notre travail, nous structurons notre mémoire comme suivant :

Le chapitre 1 présente les procédés logiciels et les architectures logicielles (définitions, les avantages d'architecture logicielle, les composants et les connecteurs).

Le chapitre 2 présente EPF Composer et les concepts SPEM qu'il utilise.

Le chapitre 3 présente les principes du déploiement d'architecture de procédé logiciel et tout ce qu'il y'a comme entrée et comme sortie pour la réalisation de notre programme.

Le chapitre 4 est la conception de l'application que nous avons développé avec Java Eclipse et en utilisant le langage de modélisation UML.

Le chapitre 5 est l'implémentation de notre application, nous présentons l'application développée, les outils et les langages de programmation utilisés.

Nous terminons par une conclusion générale et les perspectives futures de recherches.

Chapitre 1

Procédés Logiciels et architectures logiciels

1. Introduction :

Dans les années 80, les informaticiens ont voulu éviter la répétitivité de certaines tâches en les formalisant et en les automatisant. Ils se sont intéressés en premier à la formalisation des tâches du développement logiciel, ce qu'ils ont appelé le Procédé Logiciel.

Par la suite, l'utilisation des procédés s'est étendue à d'autres domaines permettant de les faire évoluer.[1]

Ce chapitre introduit les notions de procédé et de procédé logiciel. Nous présentons les notions de modèle et méta-modèle (modèle de modèle). Ensuite, nous présenterons les modèles de procédés logiciels et des classifications. Et nous définirons les éléments de base des procédés logiciels et les relations qui les lient, ces éléments constitueront le méta-modèle des procédés logiciels.

Pour augmenter la réutilisabilité et pour une meilleure modélisation des procédés logiciels, Nous modélisons les procédés logiciels en exploitant les principes des architectures logicielles.

Nous présentons le métamodèle générique pour l'extension future du métamodèle SPEM (Software and System Process Engineering Metamodel) avec des concepts architecturaux manquants.

2-Procédé et procédé logiciel :

2.1 Procédé :

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

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

Un procédé est une approche systématique pour la réalisation d'un objectif. Le terme systématique implique la séparation entre l'exécution réelle des tâches et l'idée abstraite sur la façon dont les tâches doivent être ou sont exécutées. [3]

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

2.2-Procédé Logiciel (Software Process) :

C'est l'ensemble des étapes, des activités, des équipes, des outils et des techniques dont le but est d'assurer la réalisation de développement et la maintenance de systèmes logiciels. [4]

Un procédé logiciel définit les tâches à réaliser, les rôles participants et les produits manipulés pour élaborer ou maintenir un système logiciel. [5]

La qualité des systèmes logiciels dépend de la qualité des procédés par lesquels ils sont créés. [3]

3-Les Classes Des Procédés logiciels :

Différentes classes des procédés logiciels ont été distinguées : [1]

3.1-procédé logiciel exécutable :

Définit la manière à ce que le développement logiciel soit organisé, géré, assisté et amélioré, en exécutant un programme spécifique au développement logiciel.

3.2-procédé logiciel semi-formel :

Non exécutable leur but est de décrire la façon dont le logiciel va être développé.

Exemple : Le "Processus Unifié" UP

Dans UP, les procédés servent à modéliser la façon dont le logiciel va être développé, mais de manière générique.

4-La Notion de Model et Meta-model procédé logiciel:

Pour pouvoir analyser, améliorer, mesurer et automatiser les procédés logiciels, il faut d'abord définir explicitement la modélisation de procédés logiciels. L'objectif principal de la modélisation de procédés est d'expliciter les pratiques de développement pour pouvoir les étudier, les améliorer et les utiliser de manière répétitive, gérable et éventuellement automatisable.

4.1-Modèles de procédé logiciels :

Plusieurs définitions ont été données du modèle de procédé logiciels nous citons les plus explicites.

Un modèle de procédé logiciel est la représentation explicite d'un procédé logiciel dans un langage de description de procédés logiciels exemple : PML (Modèles de Procédés Logiciels) [4]

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

Le modèle de procédé logiciel comprend des activités de développement du logiciel et de maintenance et des activités de gestion de projet et d'assurance qualité.

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

La modélisation d'un procédé logiciel est l'élaboration d'une description de ce procédé en utilisant un (ou plusieurs) modèle(s) de procédé. Dans cette définition, un modèle de procédé est une abstraction de procédé représenté par un Langage de Description de Procédés PML (Process Modeling Language). [2]

4.2-Méta-modèle de Procédé Logiciel :

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

Les éléments du méta-modèle de procédés logiciels se décomposent en éléments basiques et éléments secondaires. [2]

Les éléments basiques sont les suivants : (figure 1.1)

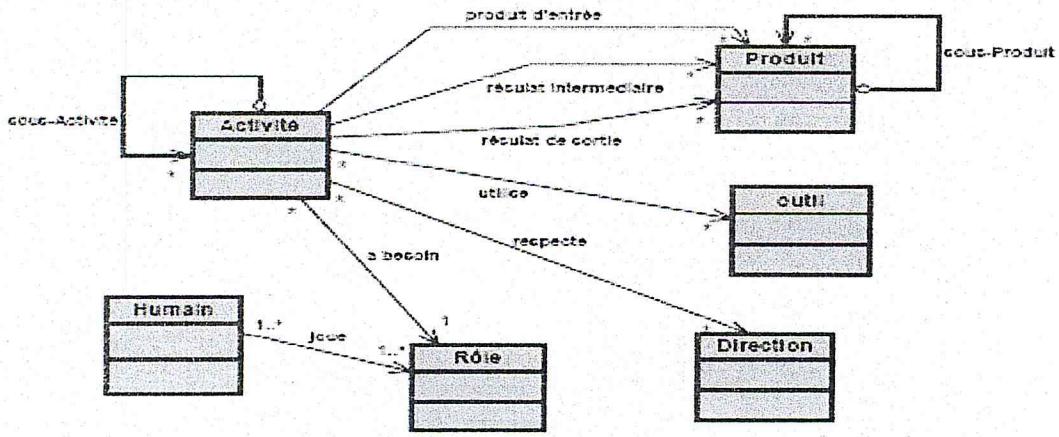


Figure 1 : Les éléments basiques de méta-modèle de procédés logiciels. [1]

- **Une Activité** est une tâche pendant laquelle des opérations sur le logiciel à développer sont accomplies. Elle est souvent associée à une ou des personnes responsables de cette activité, et à des outils de production. Une activité peut être décomposée en d'autres activités, formant ainsi plusieurs niveaux d'abstraction.
- **Un Produit** est souvent un artefact (persistant et versionné), pouvant être simple ou composite, formant les données d'entrée et de sortie des activités. Les produits peuvent être des parties du logiciel à développer, et des documents associés (documents de conception, documentation de l'utilisateur, données de test).
- **Un Rôle** décrit les droits et les responsabilités d'un humain. Un humain joue un rôle dans une activité (ou plusieurs rôles dans des activités différentes).
- **Les humains** sont des agents (ou des développeurs) du modèle de procédé logiciel, que l'on peut organiser dans des groupes. Un rôle est attribué aux humains ayant les compétences et les responsabilités nécessaires pour jouer ce rôle. Un humain peut avoir plusieurs rôles, il peut également être membre de plusieurs groupes (ces groupes peuvent aussi être imbriqués).
- **Les outils** sont des systèmes qui assistent la production du logiciel. Il existe deux sortes d'outils : les outils interactifs (éditeurs textuels, outils graphiques), et les outils simplement exécutables sans interaction (compilateurs, analyseur grammatical…).

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

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

5- L'organisation de L'OMG (Object Management Group) :

Pour expliquer le contexte de SPEM, il faut partir des travaux de l'OMG "Object Management Group". Actuellement il existe un grand nombre de méta-modèles différents qui s'inscrivent dans le cadre défini par l'OMG dont l'effort le plus significatif s'appelle MDA[1]. [1]

5.1 Présentation de l'OMG :

L'OMG (Object Management Group) est une organisation internationale de Standardisation créée en avril 1989. Sa mission est de développer des spécifications standards pour l'industrie des logiciels, qui sont pérennes, fiables, offrant ainsi un cadre commun pour le développement et l'intégration des applications distribuées dans des environnements hétérogènes. [1]

L'OMG produit et distribue gratuitement des spécifications et non des logiciels.

Chacun peut réaliser des produits logiciels implémentant ces spécifications. [5].

5.2 Architecture à quatre niveaux de l'OMG :

De [5], [6], [1].

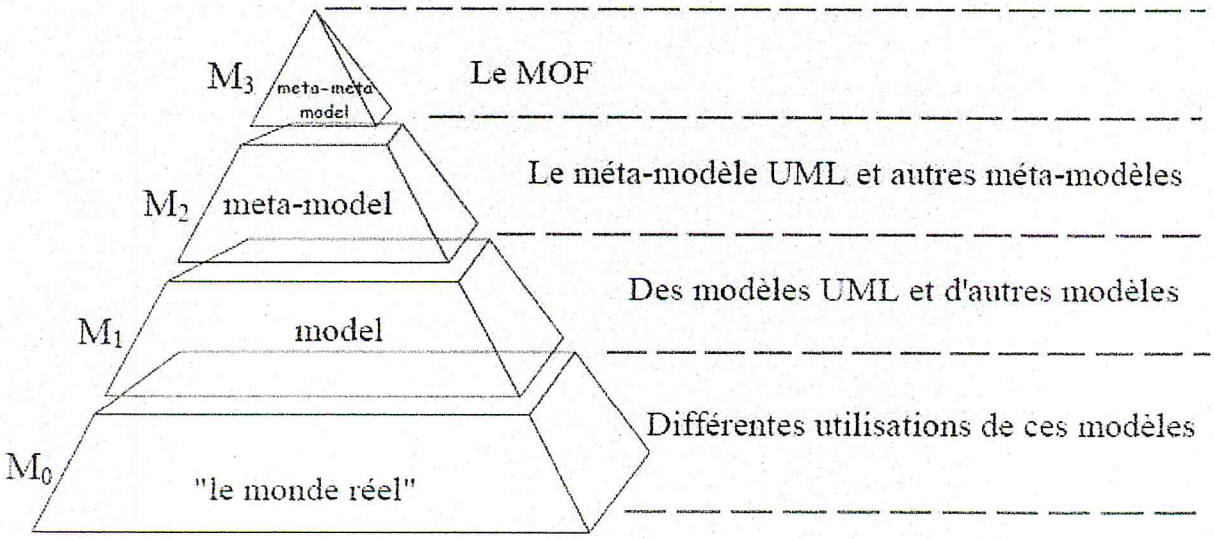


Figure 2.1 : Architecture à quatre niveaux de l'OMG. [1]

5.2.1. Le niveau M0 :

Le niveau M0 est souvent décrit comme étant le monde réel que les modèles du niveau M1 représentent. C'est le niveau des données réelles ; il est composé des informations que l'on souhaite modéliser.

[1] MDA : Model Driven Architecture

5.2.2. Le niveau M1 :

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

5.2.3. Le niveau M2 :

Le niveau M2 est composé de langages de définition de modèles, appelés aussi méta-modèle .La première définition que l'on peut faire d'un méta-modèle est celle de modèle de modèles. Il existe deux grandes techniques de méta-modélisations, le MOF (**M**eta **O**bject **F**acilitées) (voir niveau M3) et les profils UML.

Un méta-modèle définit par MOF définit un langage de modélisation spécifique à une famille de modèles.

Un profil UML définit une variante de ce langage. Une multitude de méta-modèles peut être définie, chacun décrivant les structures d'un formalisme donné. Par exemple, le méta-modèle UML qui est décrit dans le standard UML définit la structure interne des modèles UML, il représente l'ensemble des concepts UML et les relations qui existent entre eux.

Le méta-modèle SPEM: (**S**oftware **P**rocess **E**ngineering **M**anagement), standardisé également par l'OMG, décrit les concepts nécessaires à la modélisation de processus d'ingénieries logiciels.

5.2.4. Le niveau M3 :

Le niveau M3 (ou méta-méta-modèle) est composé d'une unique entité appelée le MOF : le langage unique de définition des méta-modèles. Le MOF permet de décrire la structure des méta-modèles, d'étendre ou de modifier les méta-modèles existants. Le MOF est réflexif, il se décrit lui-même, sinon il y aurait une infinité de niveaux.

6 .Architecture Logicielle :

Une définition couramment admise de l'architecture logicielle est celle de BASS, CLEMENTS et KAZMAN [27] (Définition 1). Cette définition décrit l'architecture comme une abstraction constituée d'éléments logiciels et leurs relations. Cependant, elle ne décrit pas la nature des entités logicielles représentées dans l'architecture. Elle permet ainsi à chacun de choisir les entités et peut donc être utilisée dans toutes les communautés utilisant le concept d'architecture.

Définition 1 : L'architecture logicielle d'un programme ou d'un système est la ou les structures du système, c'est-à-dire en particulier les éléments logiciels, les propriétés visibles extérieurement de ces éléments et leurs relations. [27]

Une définition plus ancienne, proposée par PERRY et WOLF [32], décrit plus clairement les entités qui composent l'architecture (Définition 2). Cette définition utilise pour les décrire le terme d'éléments architecturaux, que nous utiliserons par la suite.

Définition 2 : ... l'architecture logicielle est un ensemble d'éléments architecturaux (ou, si vous préférez, de conception) qui ont une forme particulière.

Nous distinguons trois classes différentes d'éléments architecturaux :

- les éléments de calcul.
- les éléments de données.
- les éléments de connexion.

Par la suite, la définition des éléments architecturaux a progressé et un consensus s'est formé sur leurs noms et leurs rôles dans l'architecture. On peut ainsi définir l'architecture selon la définition 3. Chaque élément architectural est ainsi nommé et leur rôle dans l'architecture est clairement défini. Il reste que la définition est encore suffisamment souple pour offrir une grande marge de manœuvre dans la description des éléments architecturaux.

Définition 3 : L'architecture est une vue abstraite d'un système en terme d'éléments architecturaux [29]. Ces éléments sont :

- les composants qui décrivent les fonctionnalités métier de l'application ;
- les connecteurs qui décrivent les communications et connexions entre les composants.
- la configuration qui décrit la topologie des connexions entre composants et connecteurs.

Dans cette section, nous présentons les différents éléments architecturaux. Pour chacun, nous proposons une synthèse des points communs entre les définitions couramment utilisées. Ensuite, nous présentons la notion de style architectural qui constitue un autre élément essentiel d'une architecture. En effet, il définit un ensemble de contraintes ainsi qu'une sémantique et une terminologie sur les éléments architecturaux.

7. Avantages des architectures logicielles :

L'architecture est un point clé affectant la plupart des attributs d'un système. Ces impacts ont été décrits par GARLAN et PERRY[33] , du point de vue de la conception. En se plaçant du point de vue de la maintenance, nous pouvons énumérer les impacts de l'architecture selon cinq angles :

- **compréhension du système :** l'architecture fournie une représentation d'un système à un haut niveau d'abstraction. Cette vue synthétique du système met en valeur la plupart des décisions de conception ainsi que les conséquences de ces mauvaises décisions. En effet, l'architecture met en valeur les contraintes du système qui justement intéressent les personnes réalisant la maintenance.

Elle permet ainsi à ces personnes de concentrer les efforts d'exploration du système sur les informations architecturales et ainsi de comprendre plus rapidement la conception et le fonctionnement du système.

- **réutilisation** : l'architecture permet facilement d'identifier les composants réutilisables d'un système.

Elle permet également, à travers les connecteurs, d'identifier les dépendances existantes entre ces parties réutilisables d'un système. Au final, ces identifications rendent la réutilisation des fonctionnalités du système plus facile mais aussi plus sûre.

- **évolution** : l'architecture fournie un squelette du système. Ce squelette permet d'identifier les parties fortement utilisées ainsi que les parties potentiellement fragiles. L'architecture permet ainsi de mettre en valeur les parties nécessitant une attention particulière lors de l'évolution du système. Mais l'architecture permet également de révéler une image précise des dépendances entre les composants. Cette image est nécessaire pour connaitre les impacts de la modification d'un composant sur les autres composants du système et donc les conséquences des différentes évolutions.

Elle permet aussi de modifier ces dépendances pour améliorer certains attributs du système tels que la performance ou l'interopérabilité. Enfin, l'architecture permet de corriger les erreurs à la source plutôt que là où elles apparaissent. Ceci peut être réalisé en localisant le composant fautif ou encore certaines dépendances ou contraintes non documentées.

- **analyse** : la vue abstraite fournie par l'architecture permet de mesurer différents attributs tels que la consistance du système, son respect du style architectural ou encore d'autres attributs de qualité.

Elle permet également de vérifier que les changements prévus dans le système sont conformes au style et aux objectifs de qualité fixés à la conception.

- **gestion de projet** : la gestion des projets de maintenance du système peuvent reposer sur les composants du système. De plus, l'architecture permet une gestion plus précise des coûts et des risques de modifications, en particulier en soulignant les dépendances entre les composants.

Elle permet également une évaluation des qualités du système dans son ensemble, mais aussi des qualités de chaque composant. Ceci permet d'identifier les parties les plus faibles du système et ainsi d'examiner et de cibler précisément leurs faiblesses. Cette identification des composants les plus faibles permet de mettre en valeur les composants les plus problématiques et de décider de leur réingénierie ou de leur redéveloppement. Enfin, la connaissance de la valeur de chaque composant et de leurs dépendances permet de planifier la réingénierie d'un système complexe en ordonnant les modifications selon leurs impacts sur la qualité du logiciel et le risque qu'elles soulèvent.

8. Les composants [29]

Les composants sont les éléments architecturaux qui encapsulent la partie métier de l'architecture.

Un composant logiciel est une unité de composition possédant des interfaces spécifiées par contrat et des dépendances contextuelles explicites.

Un composant logiciel peut être déployé indépendamment et est sujet à la composition par un tiers.

Le consensus relatif autour de cette définition provient sans doute du fait qu'elle englobe les définitions des composants issues de différents domaines. Ainsi, elle permet de décrire

aussi bien un composant abstrait tels que ceux présents dans les premières phases de conception qu'un composant exécutable selon un modèle tel que CCM (Component Corba Model) ou COM (Component Object Model), [29]
Par exemple. Cette généricité est due à la fois au manque de précision sur les structures externes du composant et sur l'absence de description des structures internes du composant.

Au delà de cette définition, les composants ont bénéficié des rapprochements entre les différentes communautés utilisant l'architecture logicielle. La définition a, ainsi, progressivement évolué pour préciser les éléments communs aux différentes communautés.

Grâce à cette évolution, nous pouvons compléter la définition des composants en décrivant plus précisément leurs structures externes et internes ainsi que leurs relations de composition.

8.1 Structure externe d'un composant

La structure externe d'un composant est généralement caractérisée par deux types d'éléments.

- Les premiers sont les interfaces du composant. Elles sont la spécification des services fournis et requis par le composant.
- Les seconds sont les propriétés du composant. Elles servent à documenter l'architecture en décrivant les aspects relevant de la conception ou de l'analyse du composant.

1. **Interfaces** Les interfaces sont la partie visible d'un composant (Figure 1). Elles servent à déclarer les services fournis et requis par le composant et constituent, d'après la définition proposée par Djalel CHEFROUR[35]:

« Un composant est une entité logicielle qui fournit un service particulier via une interface séparée de l'implantation mettant en œuvre ce service. »

Une interface est composée de deux dimensions :

- **une dimension sémantique :** les interfaces décrivent la sémantique des fonctionnalités fournies et requises proposées par le composant. Ces fonctionnalités, appelées aussi services, déterminent le type de l'interface. Ainsi, si le service associé à une interface décrit le comportement fonctionnel du composant. c'est une interface fournie. Au contraire, si le service décrit les fonctionnalités dont le composant a besoin pour fonctionner, l'interface est une interface requise ;

- **une dimension structurelle :** en plus de décrire les fonctionnalités du composant, l'interface est le point d'interaction entre le composant et son environnement. Cet aspect structurel est parfois décrit comme une entité séparée, appelée port. C'est cette dimension qui est représentée lorsque l'on schématise un composant (Figure 1).

2. **Propriétés**. Les propriétés des composants sont de trois types :
   - **propriétés non fonctionnelles :** ces propriétés peuvent concerner la structure, le comportement ou les fonctionnalités du composant. Elles sont, par exemple, liées à la sécurité, la performance ou la portabilité du composant. Elles peuvent être configurables en fonction du contexte d'exécution particulier. Elles peuvent, entre autre, permettre de simuler le comportement d'un composant avant même son implémentation ;

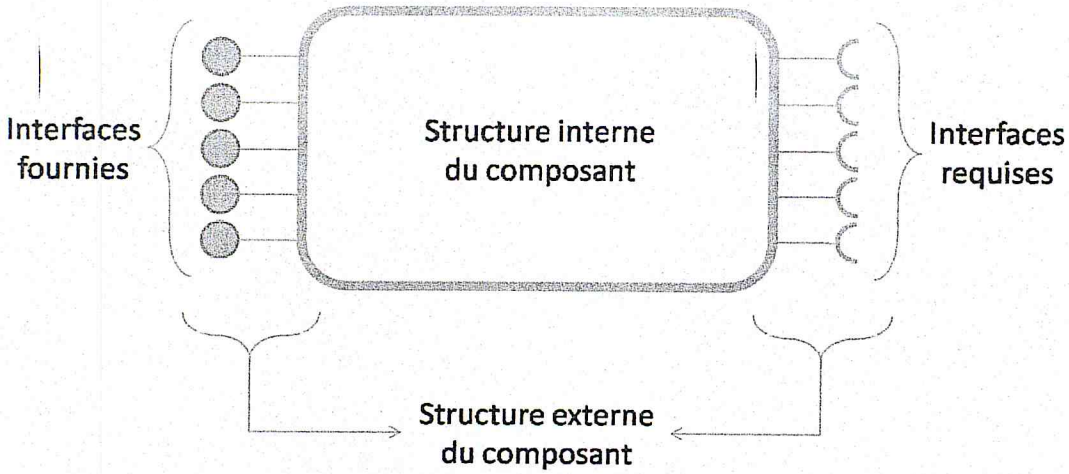


Figure 1 : Représentation d'un composant [35]

- **contraintes :** ces propriétés définissent des contraintes sur les aspects fonctionnels ou non du composant. Ce sont des propriétés qui doivent être vérifiées pour que le système soit considéré comme cohérent.

- **contrats :** ces propriétés portent sur les interfaces du composant. Les contrats portant sur les interfaces fournies définissent des contraintes et garantissent que les services de l'interface respectent ces contraintes. Les contrats portant sur les interfaces requises imposent aux composants fournisseurs un ensemble de contraintes qui doivent être vérifiées par les services fournis au travers de leurs contrats.

8.2 Structure interne d'un composant

Il existe deux types de structures internes des composants [27] :
D'abord, elle peut être constituée par une description ou une implémentation, dans un langage de programmation, des fonctionnalités du composant. Les composants possédant une telle structure interne sont des composants atomiques. Ils sont les blocs de base de l'architecture.

Le second type de structures internes est composé d'autres composants. Ces composants sont des composites constitués de composants internes. Ces composants internes peuvent eux aussi être atomiques ou composites. Comme le composant atomique, le composant composite dispose de ses propres interfaces. Les fonctionnalités proposées ou utilisées par ces interfaces peuvent alors être déléguées aux composants internes ou directement pris en charge par l'implémentation du composant composite. Pour réaliser cette délégation et pour interagir avec ses composants internes, le composant composite possède également des interfaces internes qui comme pour les externes peuvent être fournies ou requises.
Néanmoins, si la structure interne de tous les composants composites est constituée de composants,

les composants composites peuvent être classés selon deux axes :[35]

- **les interactions entre les composants internes :** soit les composants internes peuvent interagir directement entre eux, soit les composants internes n'interagissent qu'avec le composant composite. Dans ce dernier cas, le composite peut avoir à jouer le rôle de médiateur entre les différents composants internes.

- **la délégation des interfaces :** soit le composant composite peut prendre en charge les interfaces, soit il délègue ces interfaces aux composants internes à travers des ports spécifiques. Dans ce dernier cas, le composant composite constitue une enveloppe pour les services fournis par ses composants internes. Au contraire, dans le premier cas le composite propose des services supplémentaires par rapport à ses composants internes.

La figure 2 illustre les quatre cas possibles en fonction de ces deux axes.

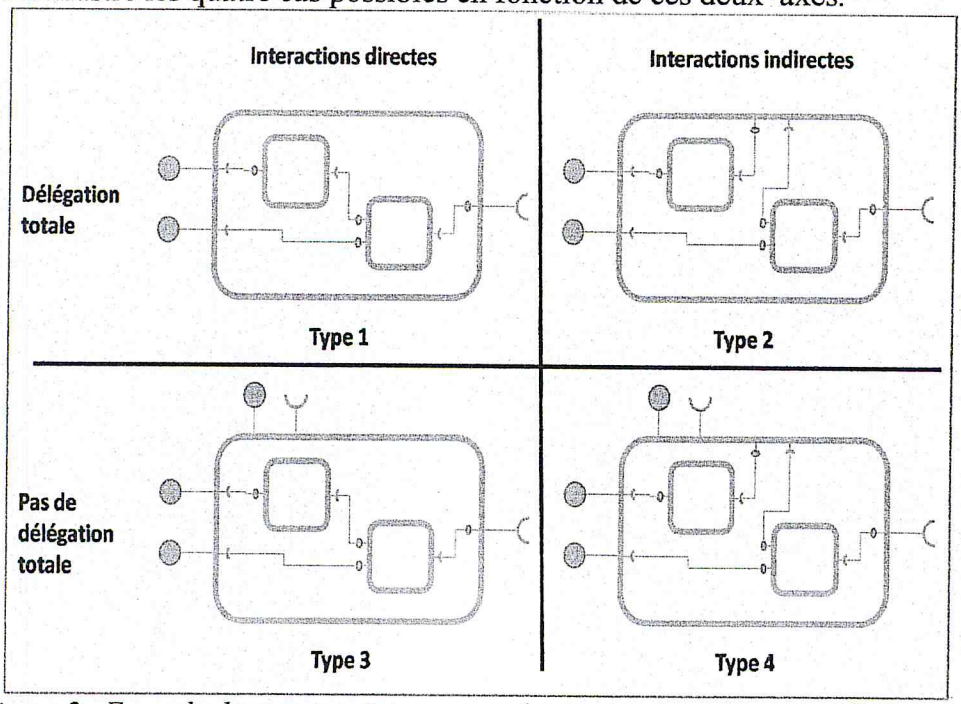


Figure 2 : Exemple de structure interne pour les composants composites[35]

8.3 Relations de compositions des composants

Les relations de compositions entre composants sont de deux types [27]. Il existe d'abord des relations de compositions horizontales. Ce sont les relations déterminées par les interfaces requises et les connecteurs. Ces relations permettent d'assembler les composants et les connecteurs. Elles n'imposent pas de contraintes particulières sur les éléments impliqués. A ce titre, ces relations sont les plus simples à utiliser pour assembler des composants provenant de sources diverses.

Les autres relations sont dites verticales. Ces relations représentent les relations entre un composant composite et ses sous-composants. Elles possèdent une sémantique différente de celle de la composition horizontale et constituent la différence majeure entre les composants composites et les composants atomiques. En effet, contrairement aux relations de compositions horizontales, la composition verticale impose des contraintes strictes sur les différentes parties impliquées. Ces contraintes varient en fonction du type de la relation qui peut être plus ou moins forte. Par exemple, une relation de composition verticale forte implique la simultanéité dans la création et la destruction du composite et des sous-composants. Ces contraintes sont associées à la sémantique de la relation de composition

verticale, mais elles ne sont pas représentées directement dans l'architecture. Ainsi, ces relations apparaissent, comme les relations horizontales, à travers des interfaces requises ou fournies qui relient le composite et les sous-composants.

9. Les connecteurs

Les connecteurs constituent un élément architectural au même titre que les composants. Ainsi, ce sont des entités de premier plan dans une architecture. Cette considération identique pour les composants et les connecteurs est l'atout principal des architectures. En effet, la distinction entre les aspects métiers et ceux de communication se fait plus facilement en considérant de la même manière les deux entités.

A la différence des composants, les définitions des connecteurs sont peu nombreuses et relativement convergent.

Par exemple, SHAW et GARLAN [36] présentent les connecteurs selon la définition suivante.

« Les connecteurs gèrent les interactions entre les composants ; c'est-à-dire, ils établissent les règles qui gouvernent les interactions entre composants et spécifient tous les mécanismes auxiliaires nécessaires. »

D'après cette définition, les connecteurs décrivent les communications et les connexions entre les composants en modélisant de manière explicite les interactions entre les composants. Un connecteur peut décrire des interactions simples comme un appel de méthode ou des interactions plus élaborées telles que des accès à des bases de données.

Cette définition propose une description des rôles des connecteurs. De la même manière, dans la plupart des travaux, l'aspect structurel est laissé de côté au profit de l'aspect fonctionnel. Ceci permet de conserver suffisamment de généricité et de pouvoir utiliser toutes entités remplissant le rôle adéquat.

Cependant, l'augmentation de l'intérêt pour cet élément architectural a conduit à une formalisation de sa structure. Nous pouvons ainsi décrire plus précisément les structures internes et externes des connecteurs.

9.1 Aspect Fonctionnel du connecteur

L'aspect fonctionnel des connecteurs est décrit en détail dans la taxonomie de MEDVIDOVIC [36] .

Elle classe les connecteurs selon deux niveaux décrivant les services et les techniques utilisés par les connecteurs.

Le premier niveau de la taxonomie de Medvidovic contient quatre types de connecteurs qui se distinguent suivant les services qu'ils proposent :

- **la communication** : les connecteurs de communication transfèrent les données entre les composants. Ce service est l'élément de base de l'interaction entre composants. Les connecteurs de communication permettent aux composants de transmettre les messages, d'échanger les données ou de communiquer les résultats des calculs.

- **la coordination :** les connecteurs de coordination gèrent le transfert de contrôle entre les composants, permettant aux composants d'interagir par transfert du flot d'exécution. Les appels de fonctions ou les invocations de méthodes sont des exemples de connecteurs de coordination simple.

- **la conversion :** les connecteurs de conversion servent d'interprètes entre composants hétérogènes, c'est-à-dire provenant de sources différentes et surtout non prévus pour interagir à l'origine. Les connecteurs convertissent le service fourni par un composant pour permettre à un autre de recevoir son service requis sous le bon format. Les conversions peuvent porter sur le nombre ou le type des données échangées, sur la fréquence ou l'ordre des interactions ou encore sur le nom des services.

- **la facilitation :** même lorsque les composants sont prévus pour être assemblés entre eux, il peut être nécessaire d'introduire des mécanismes pour faciliter et optimiser leurs interactions. Les connecteurs de facilitation offrent donc des services tels que l'équilibrage des charges, la planification ou encore le contrôle de la concurrence.

Les services permettent de faire un premier classement des connecteurs mais ils laissent de coté de nombreux détails essentiels sur leur mise en pratique par les connecteurs. Pour répondre à cela, le second niveau classe les composants en fonction de la manière dont ils réalisent le service. Les huit types proposés peuvent être utilisés pour proposer plusieurs services. Leurs mises en pratique dépendent de plusieurs paramètres qui sont représentés, dans la taxonomie, par leurs dimensions et dont les valeurs possibles sont représentées par les valeurs de la taxonomie.

Les huit types sont [37]**:** les appels de procédures, les évènements, les accès aux données, les liens, les flots, les arbitres, les adaptateurs, les distributeurs.

Dans notre travail nous utiliserons trois types (les liens, les arbitres, les adaptateurs)

- **les liens (ex transmission):** les connecteurs de type « liens » permettent la mise en place de canaux de communication entre les composants. Ces canaux sont ensuite utilisés par les autres connecteurs pour mettre en place leur service.Les connecteurs liens proposent donc un service de facilitation, en particulier de la phase de mise en place du système. Ils peuvent d'ailleurs disparaitre après cette phase.

- **les arbitres (ex précédence):** les connecteurs arbitres organisent le fonctionnement du système. Ainsi, ils peuvent remplir deux services. Le premier est un service de facilitation. En effet, les connecteurs arbitres fournissent les outils pour négocier le niveau des services ou encore les interactions nécessitant un certain niveau d'isolation ou de fiabilité. L'autre service rendu est la coordination. En effet, les connecteurs arbitres peuvent rediriger le flot d'exécution entre les composants. Cette redirection peut dépendre, par exemple, d'une planification ou d'un objectif d'équilibrage des charges entre les composants.

- **les adaptateurs (ex fusion)** : les connecteurs adaptateurs proposent un service de conversion. L'adaptateur peut ainsi permettre l'interaction entre composants qui ne sont pas prévus pour communiquer. Pour cela, il doit adapter la politique de communication et le protocole d'interaction des composants.
  Par exemple, les adaptateurs permettent de résoudre les problèmes de polymorphisme en faisant le lien entre des interfaces fournies et requises qui se complètent mais n'ont pas le même nom.

9.2 Structure externe du connecteur [30],[38]

A l'image des composants, un connecteur est caractérisé par deux éléments : ses interfaces qui spécifient les types et le rôle des composants communicant à travers le connecteur ; et ses propriétés qui décrivent les aspects relevant de la conception ou de l'analyse des connecteurs et documentent l'architecture d'une manière similaire aux propriétés des composants.

1. **Interfaces :** Les interfaces, appelées rôles dans certains langages de description d'architecture tel que Wright[30], sont la partie visible du connecteur (Figure 3). Elles servent à déclarer les participants à l'interaction décrite par le connecteur. Comme celles des composants, elles constituent les points de connexion entre les connecteurs et les composants. Néanmoins, à la différence des composants, les interfaces ne décrivent pas de services fonctionnels mais des mécanismes de connexion. Elles décrivent également le rôle de chacun des composants impliqués.

Contrairement aux composants, on ne distingue généralement pas de directions pour les interfaces de connecteurs. Cependant, dans le reste de ce manuscrit, nous utilisons, comme pour les composants, les appellations d'interfaces entrantes et sortantes. Les premières relient le connecteur à des interfaces fournies de composants alors que les secondes relient le connecteur à des interfaces requises de composants.

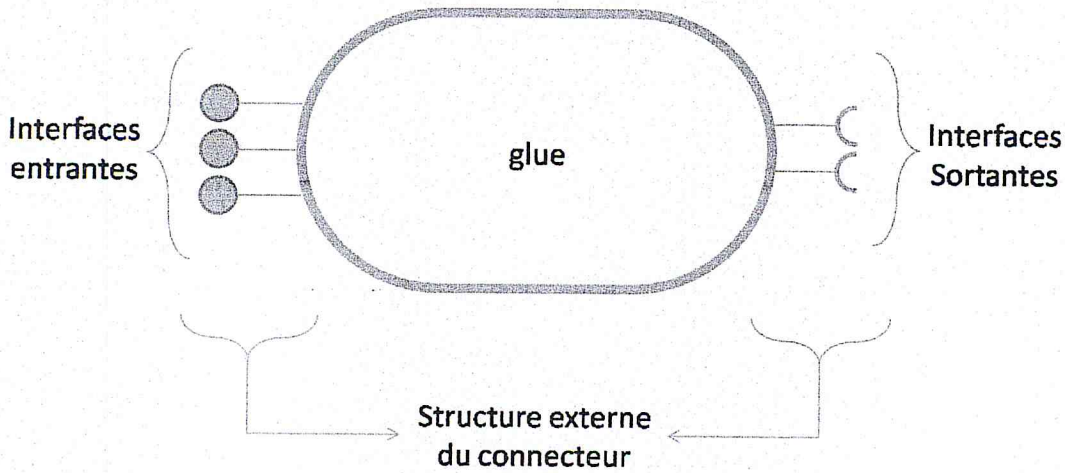


Figure 3 : Représentation d'un connecteur[35]

3. **Propriétés** : Les propriétés des connecteurs sont de deux types :

- **propriétés non fonctionnelles :** elles spécifient les besoins du connecteur pour une implémentation correcte. Par exemple, elles peuvent concerner la performance ou la sécurité. Comme pour les composants, ces propriétés marquent une séparation claire entre les aspects fonctionnels et non fonctionnels. Elles permettent ainsi de simuler le comportement des connecteurs à des fins d'analyse, de définition des contraintes ou encore de sélection des connecteurs.

- **contraintes :** ces propriétés définissent les conditions d'utilisation du connecteur. Comme pour les contraintes portant sur les composants, ces contraintes doivent être vérifiées pour que le système soit considéré comme cohérent.

9.3 Structure interne du connecteur [35],[38] :

De la même manière que pour les composants, on distingue deux types de structures internes pour les connecteurs : structure atomique ou composite.

1. **Connecteur atomique** [38]: La structure interne des connecteurs atomiques est appelée glu (Figure 3). Elle forme la passerelle entre les interfaces du connecteur. Pour cela, elle décrit le protocole de communication entre les interfaces, points d'accès des composants vers le connecteur. Ce type de connecteur est le plus couramment utilisé. En effet, même si les connecteurs sont, en théorie, des entités du même niveau que les composants, dans la pratique, les outils utilisant ou décrivant les architectures considèrent les composants comme les éléments prépondérant de l'architecture. Ainsi, alors que les composants composites sont utilisés et répandus, les connecteurs sont souvent considérés comme des interactions simples, et sont donc décrits par des connecteurs atomiques.

Par exemple, dans les langages de description d'architecture, les connecteurs sont proposés de trois façons : il existe un seul type de connecteur simple, parfois même sans représentation explicite. les connecteurs doivent être choisis dans un ensemble prédéfini (UNICON) ; enfin, il est parfois possible de définir un connecteur atomique.

4. **Connecteur composite** [38]: Les connecteurs composites ont une structure interne plus complexe que celles des connecteurs atomiques. A l'image des composants composites, les connecteurs composites possèdent une structure interne composée de composants, de connecteurs et d'une configuration. Ainsi la structure interne d'un connecteur composite est une architecture interne à ce connecteur (Figure 4).

Ce type de connecteur permet, par exemple, de décrire des protocoles complexes de communication qui demandent un pré et un post-traitement des données.

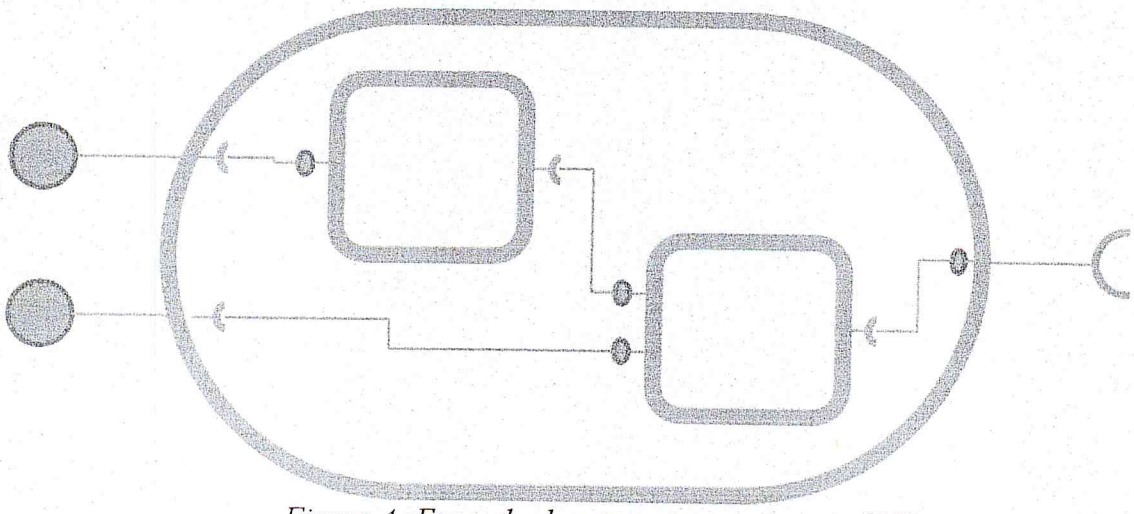


Figure 4 :Exemple de connecteur composite[35]

10. La configuration [37] :

La configuration définit la structure de l'architecture. Pour cela, la configuration d'une architecture présente la topologie des connexions entre les composants et les connecteurs ainsi que les propriétés de cette topologie. Par conséquent, la configuration est caractérisée par deux éléments : la structure de la configuration, représentant la topologie des connexions entre composants et connecteurs, et les propriétés de la configuration.

10.1 Structure de la configuration

La structure de la configuration représente la topologie des connexions entre les composants et les connecteurs. Elle vérifie la correspondance entre les interfaces des composants et des connecteurs. Pour représenter cette topologie, la structure de la configuration peut prendre deux formes [37]:

- **multigraphe :** cette représentation de la structure de la configuration utilise un multigraphe, c'est à-dire un graphe dont les arêtes peuvent connecter plus de deux sommets. Les sommets du multigraphe représentent les composants de l'architecture, alors que les arêtes représentent les connecteurs. Le défaut principal de cette représentation est qu'elle ne place pas les composants et les connecteurs au même niveau.

- **graphe biparti :** un graphe biparti contient deux types de sommets. L'origine et l'extrémité de chaque arête du graphe appartiennent obligatoirement à un type de sommets différents. Ce graphe biparti permet de représenter les deux éléments architecturaux, *i.e.* composants et connecteurs, par des sommets. Cette représentation permet ainsi de pallier les défauts du multigraphe en considérant les composants et les connecteurs comme des entités du même niveau. Pour représenter une structure de configuration, le graphe biparti doit également vérifier que chaque sommet de type connecteur est de degré minimum deux. Ceci permet de vérifier que chaque connecteur relie au moins deux composants.

10.2 Les propriétés de la configuration

Les propriétés de la configuration sont similaires à celles des composants et des connecteurs. Comme pour les autres éléments architecturaux, ces propriétés sont de deux types[37] :

- **propriétés non fonctionnelles :** certaines propriétés non fonctionnelles ne peuvent pas être exprimées au niveau des composants ou des connecteurs. Il faut donc exprimer ces propriétés au niveau de la configuration.

Ces propriétés concernent par exemple l'environnement de déploiement de l'architecture.

- **contraintes :** les contraintes portant sur la configuration s'ajoutent à celles portant sur les autres éléments architecturaux. Elles permettent d'exprimer des contraintes portant sur plusieurs éléments architecturaux. Par exemple, une contrainte peut exprimer une relation entre deux composants.

Les contraintes de la configuration peuvent également être globales et portent sur l'ensemble des éléments architecturaux.

11-Introduction à l'architecture de procédé logiciel [25],[39] :

Modéliser de nouveaux procédés logiciels (PLs) en phase avec les nouvelles pratiques, méthodes, outils de développements est une nécessité pour la réussite de projets de développement logiciels. Les Modèles de PLs décrivant l'enchainement d'activités, les responsables, les ressources et les outils utilisés pour la réalisation du produit logiciel doivent refléter et s'adapter à la réalité du développement. Utiliser des modèles de PLs de qualité est alors une garantie pour la réussite des projets de développements logiciels.

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 pratiques et le savoir faire acquis par les précédentes expériences de modélisation et d'exécution de PLs éprouvés est la solution que nous préconisons.

Métamodélisation architecturale des procédés logiciels Afin d'augmenter la réutilisabilité de ces connaissances, nous optons pour la modélisation de PLs à base d'architectures logicielles.

11-1. Sémantique adoptée aux concepts architecturaux pour les procédés logiciels

Pour augmenter la flexibilité des PLs, il est certain que le connecteur a un rôle déterminant dans l'exécution des modèles de PLs. La possibilité offerte de spécifier, personnaliser, contrôler, adapter, faciliter les interactions entre activités PL est un atout majeur. Les transitions entre activités peuvent être maitrisées et les déviations d'exécution par rapport à la modélisation limitées.[25]

Dans notre approche le connecteur procédé est considéré comme une entité de première classe. Il est possible de lui affecter des fonctionnalités capitales qui permettent de spécifier différents types de transfert lors de la modélisation ou de l'exécution des PLs. Ainsi, nous remarquons que des connecteurs procédés peuvent être identifiés pour faciliter et adapter le transfert de produits [13][15] .Aussi, l'aspect contrôle de flux peut être traité par des connecteurs explicites, ainsi, il est possible de définir des connecteurs qui évaluent l'exécution puis décident des changements à opérer pour une meilleur exécution.

Nous définissons notre connecteur procédé comme une activité qui permet de "faciliter et contrôler" les transitions entre les activités procédés. Contrairement, au "Composant Procédé" le "Connecteur procédé" ne crée pas de nouveaux produits, mais "adapte et contrôle" des produits existants.

La distinction entre Activés de "création" de produit (qui constitueront les composants procédés) et Activités "d'adaptation et de contrôle" de données (qui constitueront les connecteurs procédés) modifie la sémantique des concepts identifiés. Ainsi, notre interprétation des concepts architecturaux des PLs conduit à l'identification de :

- **Composant composite** : Décrit comme un assemblage de composants procédés et de connecteurs explicites.
- **Composant procédé élémentaire** : Décrit un traitement réalisé sur des produits en entrée pour "la création" de nouveaux produits en sortie.
- **Port procédé (Interface du composant)** : L'interface d'un composant est un ensemble de points d'interactions du composant procédé ; elle spécifie les services fournis et requis nécessaires de l'exécution du composant procédé. L'interface du composant procédé est un ensemble de "Ports Procédés", les ports requis correspondent aux "données en entrée" nécessaires à l'exécution du composant

procédé, Les ports fournis correspondent aux "données en sortie". Deux types de ports sont définis :

- Ports flux de données (Data Flow Ports) : Spécifiques aux produits des PLs, Ils permettent le transfert des produits logiciels du PL.
- Ports flux de contrôle (Control flow Ports) : Spécifiques aux flux d'exécution des PLs, Ils permettent d'identifier l'ordre et l'état d'exécution du PL.

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

Concept procédé logiciel.	**Notre sémantique ajoutée**
Fragment de procédé.	Composant procédé composite : Composé de composants et de connecteurs explicites.
Activité de **"création"** de nouveaux produits logiciels.	Composant procédé élémentaire.
Donnée (en entré /en sortie) d'une activité de création.	Port procédé : peut être un Port flux de donnée ou un Port flux de contrôle.
Structure procédé logiciel.	Configuration procédé : Ensemble de composants procédés et connecteurs procédés respectant des contraintes d'assemblage.
Cycle de vie du logiciel.	Stye procédé : Introduit formellement avec les concepts "Types", invariants et contraintes.
Donnée fournie ou requise (données en entré ou en sortie) d'une activité d'adaptation.	Rôle connecteur : peut être un rôle connecteur "Data Flow" ou un rôle connecteur "Control Flow".
L'activité **"d'adaptation ou de contrôle"** des données.	Connecteurs explicites : Taxonomie de Connecteurs prédéfinies.
Lien de précédence entre Activité de création et Activité d'adaptation.	Attachement : Un lien entre un Port et un Connecteur Rôle de même type.
Lien de délégation entre activités.	Binding : Un lien entre les Ports ou entre Connecteurs Rôles de même type.

Tableau1 : Notre correspondance entre concepts de procédés logiciels et concepts d'architectures logicielles. [25]

- **Rôle connecteur (Interface connecteur) :** L'interface de connecteur procédé est représenté par de "Rôle Connecteur". De la même manière que les ports procédés, ils représentent les données (le produit ou flux d'exécution) requis ou fournis par les connecteurs procédés deux types de "Connecteur Rôle" sont définis : les "Data Flow" connecteur Rôle et les "Control flow" connecteur rôle.
- **Binding** : Est un "lien" entre les Ports procédés internes et les Ports procédés externes qui permet de décrire la structure interne de la configuration procédé ou d'un composant composite. De la même manière, le binding des rôles Connecteur permet de définir des connecteurs complexes en combinant plusieurs connecteurs procédés. Le binding se fait entre ports procédés de même type.
- **Attachement** : Est un "lien" entre un "Port Procédé" et un "Rôle Connecteur" qui permet de formaliser l'enchainement des composants et des connecteurs procédés. L'attachement se fait entre port et connecteur rôle de même type.
- **Configuration procédé :** Elle décrit l'ensemble logique des composants procédés et des connecteurs procédés en déterminant explicitement les contraintes d'assemblage de la structure procédé. Une configuration peut respecter un style prédéfini tel que le cycle

de vie de logiciel ou pas.

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

Nous regroupons Les concepts identifiés en un métamodèle générique basé sur UML. Ce métamodèle est indépendant du métamodèle SPEM, il servira de base de réflexion pour l'extension du métamodèle SPEM. Nous introduisons les concepts "Type Composant", "Type connecteur", "Type Port", "Type Role" pour décrire formellement les styles architecturaux de PLs.

Ces concepts sont définis indépendamment des concepts PLs, ils sont introduits pour décrire formellement les styles de PLs.

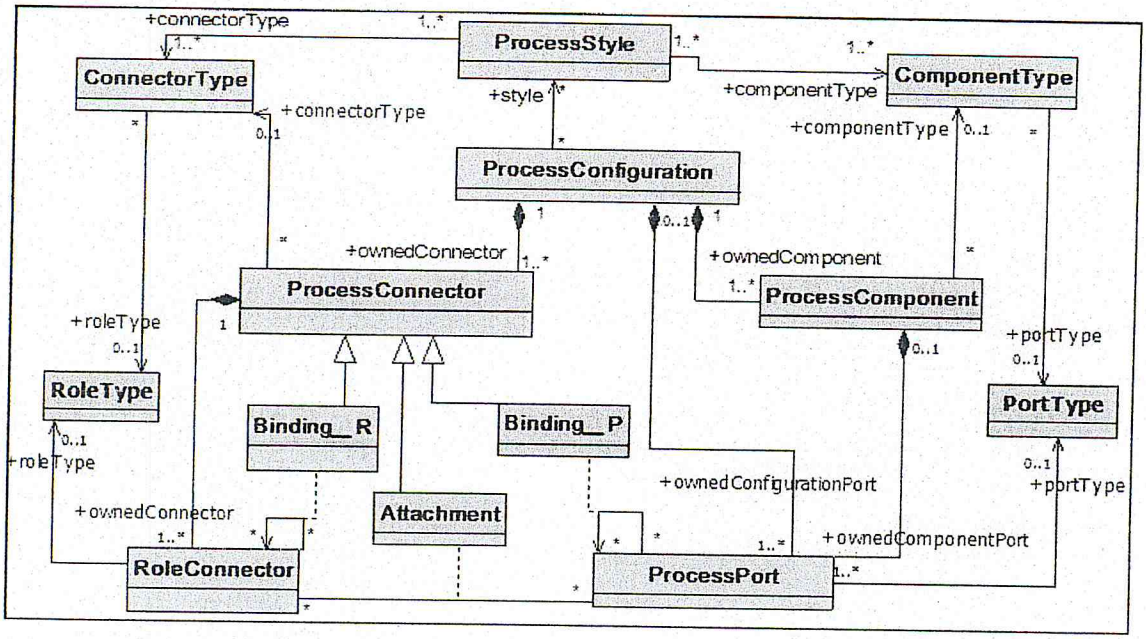


Figure 5 – Notre métamodèle générique regroupant les concepts architecturaux des procédés logiciels. [25]

Ainsi, comme pour les architectures logicielles, une "configuration Procédé" est constituée de "composants procédés" connectés à travers des "connecteurs procédés". La description de la structure interne des "connecteurs procédés" et des "composants procédés" se fait en utilisant des "binding". Le "style procédé" est décrit à travers des associations aux classes "Type" des différents concepts.

11-2 Sémantique des Connecteurs de procédés logiciels [15], [25]

En analysant le comportement des modèles de PLs nous repérons les activités récurrentes d'adaptation et de contrôle de flux. Ainsi, nous identifions une taxonomie de connecteurs explicites pour la modélisation des architectures PLs. Ces connecteurs procédés nous offrent la possibilité de gérer les interactions indépendamment du type du PL. Ces connecteurs sont très intéressants pour les méthodes agiles respectant des processus où la flexibilité et la dynamicité sont très recherchées. Deux types de connecteurs sont définis :

- **Connecteur DataFlow** : Correspond à une activité d'adaptation du produit. Ces activités sont indépendantes du type du PL et permet la gestion des transferts de

données entre activités. Nous nous inspirons des activités d'adaptation de l'environnement APEL [15] pour définir nos connecteurs, ces activités sont des activités de "Fusion", "Diffusion" de Fragmentation de produits (Figure -6-).

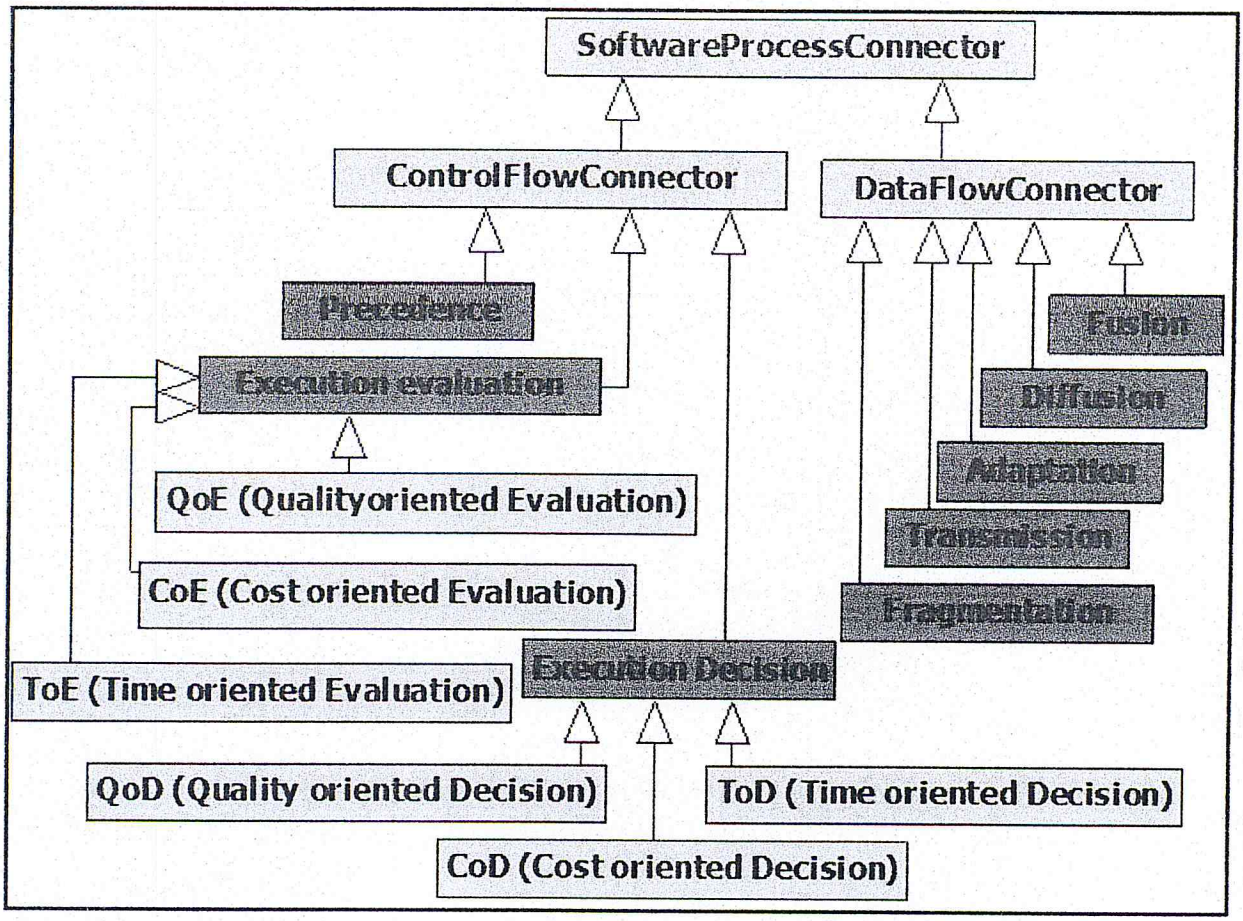


Figure 6 : Taxonomie de connecteurs procédés prédéfinis. [25]

- **Connecteur ControleFlow** : Correspond à des activités de transfert et de contrôle de flux. Ces connecteurs permettent de définir le style d'exécution du PL en évaluant la qualité de son exécution. Le connecteur Control Flow peut exploiter des critères de qualité d'exécution définies pour le PL, ainsi que, les résultats effectifs de l'exécution du PL afin d'orienter l'exécution en court du PL. Ainsi, passer d'une "phase" à la phase suivante, ajouter un "Incrément", une "Itération" ou un "Sprint", peut être pris en charge totalement ou partiellement par un connecteur ControleFlow. Ces connecteurs peuvent être très efficaces pour l'exécution des méthodes agiles qui exigent une grande flexibilité et une gestion efficace.

11-3.Sémantique ajoutée aux styles de procédés logiciels [25] :

En plus de l'introduction des connecteurs procédés explicites, aussi, une contribution aussi importante est l'introduction formelle du style de PL. Définir des styles architecturaux pour les PLs facilite non seulement leur modélisation en exploitant les caractéristiques des structures récurrentes, mais aussi, permet de créer de nouveaux Modèles de PLs en combinant des styles différents. L'identification d'invariants, contraintes de la structure récurrente d'un processus donné permet d'élargir l'utilisation de cette solution à d'autres types de processus, qui ne sont pas forcément précédés logiciels.

La figure -7- illustre bien la vue architecturale du PL selon la sémantique adoptée. La configuration procédé, est un assemblage de composants procédés et de connecteurs procédés. Le contrôle de flux est assuré par des connecteurs "Controle Flow", par contre, le transfert des produits est assuré par des connecteurs "DataFlow". Ces deux types de connecteurs ont leur propre type de ports.

L'interprétation du cycle de vie du logiciel comme style architecturale est incontestable. La sémantique rajoutée aux concepts architecturaux PL, nous permet non seulement de définir des styles "topologiques" de PL (les cycles de vie de logiciels par exemple), mais aussi de définir des styles d'exécution en paramétrant les connecteurs "Control Flow" définis dans notre taxonomie. Ainsi, la configuration présentée dans la figure -4- respectant le style topologique "cycle de vie en V" peut être combinée non seulement à d'autres styles topologiques, mais aussi, à d'autres styles d'exécution en introduisant d'autres types de connecteurs "Control Flow". Par exemple, en introduisant des connecteurs Contol Flow "Qualité d'exécution (QoE)", la configuration illustrée dans la figure -7- peut être modélisée pour s'exécuter en donnant la priorité au temps, au coût ou à la qualité de réalisation.

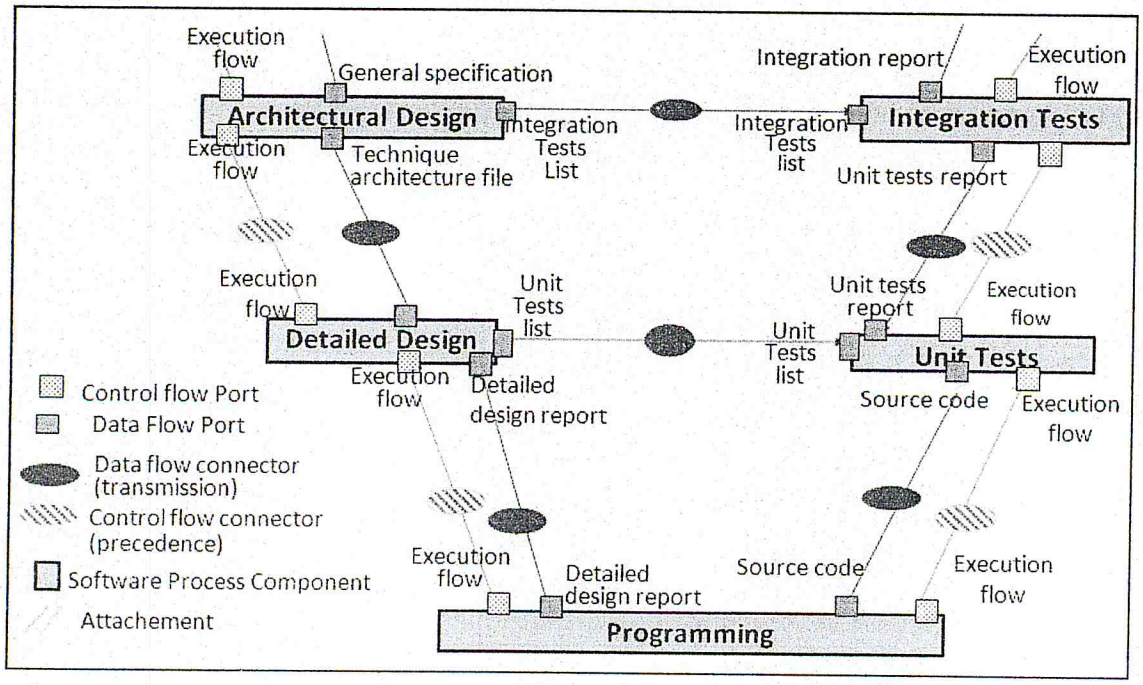


Figure 7 : Configuration procédé logiciel respectant le style topologique "cycle de vie en V".[15]

12. Conclusion :

Nous avons présenté dans ce chapitre les concepts de base concernant les procédés logiciels, Ainsi nous avons présenté les notions de modèle et de méta modèle de PLs.

Concernant les architectures logicielles nous avons présenté en détail les notions de composants, connecteurs, configuration.

Pour manipuler les architectures de PL, nous avons présenté la sémantique adoptée à cet effet, cette sémantique est regroupée dans un méta modèle d'architecture de PL.

L'objectif de notre travail est le déploiement d'architecture de PL.
Nous déployons des Architectures de PL en modèle de PL manipulable sous EPF **(Eclipse Process Framework)**, Dans le chapitre suivant nous présentons EPF, Ainsi que le fichier XML du modèle de PL résultant.

Chapitre 2

EPF

1.Introduction :

Dans notre travail nous déployons des architectures de PL en modèles de PL utilisables sous « EPF Composer»

Dans ce chapitre nous détaillons le Framework de « EPF Composer » qui vise à produire des processus logiciels personnalisables, avec un contenu processus exemplaire et d'outils, supportant une large variété de types de projets et styles de développement, nous présentons ses éléments de base et la logique de son fonctionnement

EPF Composer respecte le méta modèle SPEM, par conséquent nous détaillons le méta modèle SPEM , et les concepts de SPEM utilisés dans EPF Composer .

Nous présentons aussi le fichier résultant d'EPF Composer qui est le modèle de PL décrit en XML.

2.Le Projet EPF Composer:

L'objectif de notre travail est le déploiement de l'architecture de PL. Le résultat du déploiement doit être décrit sous EPF Composer (Eclipse Process Framework). Dans ce chapitre nous présentons les détails de cet environnement.

Eclipse est une communauté open source, dont les projets sont axés sur la construction d'une plate-forme de développement ouverte composée de cadres extensibles, des outils pour développer, déployer et gérer des logiciels à travers le cycle de vie.

Le projet Eclipse a été initialement créé par IBM en Novembre 2001 et soutenue par un consortium (partenariat) de fournisseurs de logiciels. La Fondation Eclipse a été créé en Janvier 2004, un organisme indépendant, sans but lucratif pour agir en tant que responsable de la communauté Eclipse. [1]

Les projets de la fondation Eclipse de haut niveau sont :

- Business Intelligence and Reporting Tools Project
- Data Tools Platform Project (4 sous-projets)
- Device Software Development Project (5 sous-projets)
- Eclipse Project (5 sous-projets)
- Eclipse Modeling Project (8 sous-projets)
- SOA Tools Project
- Eclipse Technology Project (28 sous-projets)
- Tools Project (10 sous-projets)
- Eclipse Test and Performance Tools Project (4 sous-projets)
- Eclipse Web Tools Platform Project (5 sous-projets)
- Total des projets 71 projets.

L'EPF Composer est un sous-projet du projet «Technologie»

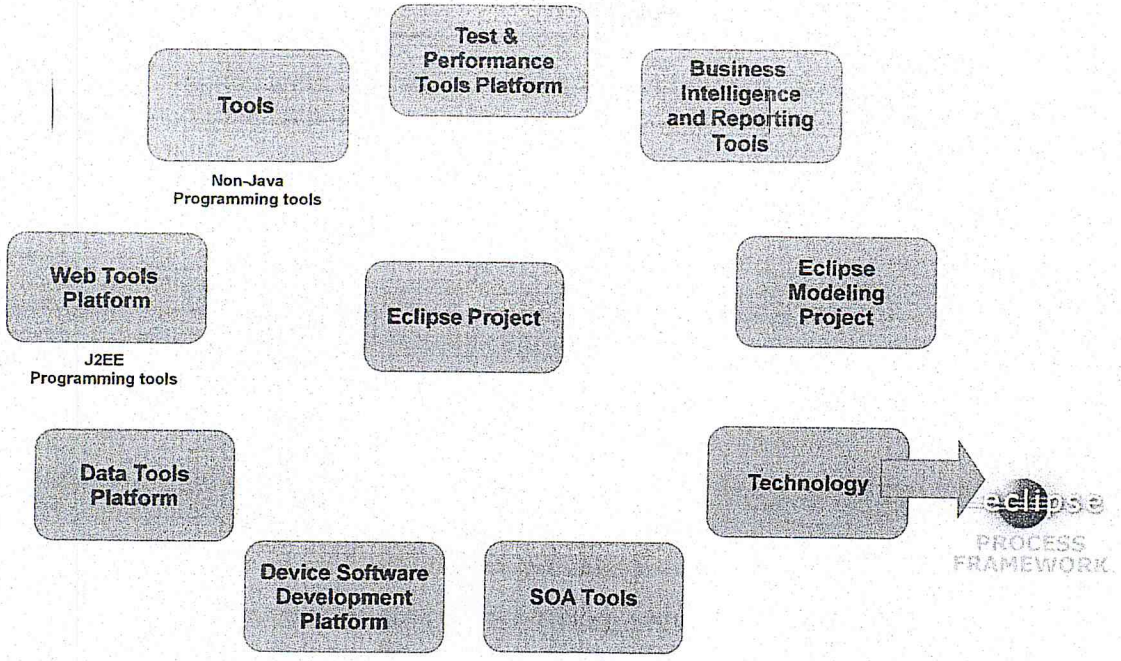


Figure 1 : les projets d'eclipes [1]

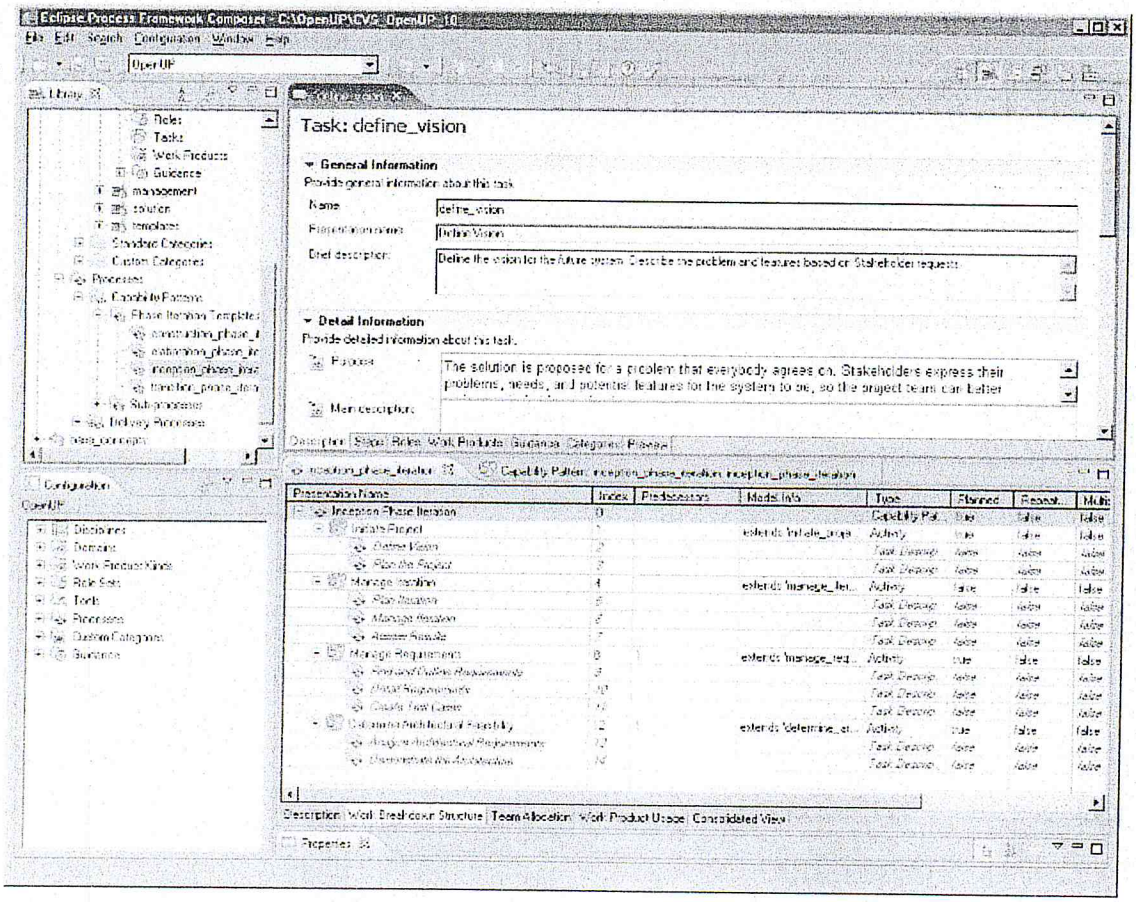


Figure 2 :outil exemplaire: EPF Composer

2.1.Objectifs du projet EPF Compser :

Le processus de projet de cadre a deux objectifs :

- Pour fournir un cadre extensible et des outils exemplaires pour l'ingénierie des processus logiciels - méthode et la création de processus, de gestion de bibliothèque, la configuration et la publication d'un processus.
- Pour fournir un contenu processus exemplaire et extensible pour une gamme de développement de logiciels et, de soutenir les processus de gestion du développement itératif, agile, et progressive, et applicable à un large éventail de plates-formes de développement et les applications.

3. EPF Composer et SPEM :

Avant de nous pencher à l'EPF Composer en détail, nous avons besoin de définir les concepts procédé logiciel de base de SPEM manipulé par EPF composer.

Remarque : EPF utilise SPEM .

Bien que le titre implique des processus logiciels, tout processus peut être représenté à l'aide SPEM.

SPEM « **S**oftware **P**rocess **E**ngineering **M**eta-model » que l'on peut traduire par « Méta-modèle d'ingénierie des systèmes et procédés logiciels » est un méta Modèle (ou modèle décrivant les concepts) visant à décrire le processus de production de logiciels pour répondre à ces problématiques.[5]

Le **S**oftware **P**rocess **E**ngineering **M**eta-model permet de décrire le processus de production des logiciels et de faciliter le partage des informations indépendamment de leur type (formel, semi formel, gestion de projet ...). [5]

Le processus de génie logiciel méta-modèle définit un langage formel pour décrire les processus de développement. Le méta-modèle SPEM est très général et peut être utilisé pour décrire tout processus de développement dans tous les domaines.

SPEM définit les éléments utilisés dans la description d'un processus, ainsi que les éléments utilisés pour structurer et gérer cette information. Tels sont les éléments de base servant à structurer et gérer l'information :
Method Library, Method Configuration, Method Plug-in, Delivery Process.[3]

- **Une bibliothèque de méthode (Method Library)** est une collection de la méthode plug-ins et des méthodes de configurations. Une méthode plug-in est un conteneur pour le contenu qui peut être indépendamment importé et exporté à partir d'une bibliothèque (c'est à dire qu'ils peuvent "plug-in" à la bibliothèque). Les méthode plug-ins peuvent réutiliser les informations dans d'autres plug-ins.,et sont encore sous-divisé en paquets pour simplifier la gestion de l'information.

- **Une méthode de configuration (Method Configuration)** définit une logique sous-ensemble d'une bibliothèque de méthodes (ie les plug-ins et des paquets) qui sera publiée ou exportée Vous pouvez penser à une configuration de la méthode comme un «filtre» appliqué à la bibliothèque qui vous donne seulement ce dont vous avez besoin pour votre usage particulier.

- **Method Plug-in** Une Method Plug-in représente un conteneur physique pour les Method Packages et Process Packages.. Il définit un niveau de granularité plus grand pour la modularisation et l'organisation du contenu et des processus de méthode.

- **Delivery Process (Processus de livraison)** une approche complète et intégrée pour effectuer un type de projet spécifique

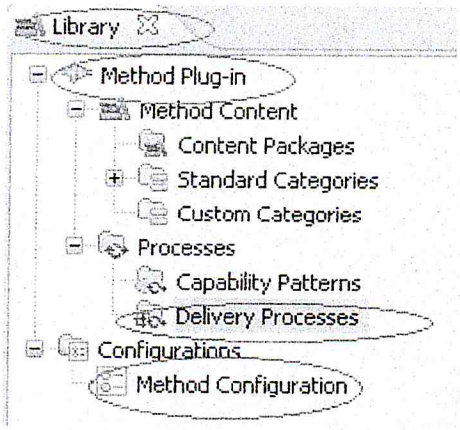


Figure : les éléments de base de SPEM utilisés dans EPF Composer.

4. EPF et ses Concepts de base : Method Content, Process de epf composer :

Pour travailler efficacement avec Eclipse Process Framework Composer, nous devons comprendre les concepts de base qui sont utilisés pour organiser le contenu et qui sont Method content et Process.

Remarque : Method content et Process contiennent les éléments de SPEM qu'on a défini précédemment.

Le principe le plus fondamental dans le cadre du processus Eclipse est la séparation du contenu de base réutilisable « Method » de son application dans les « process ».

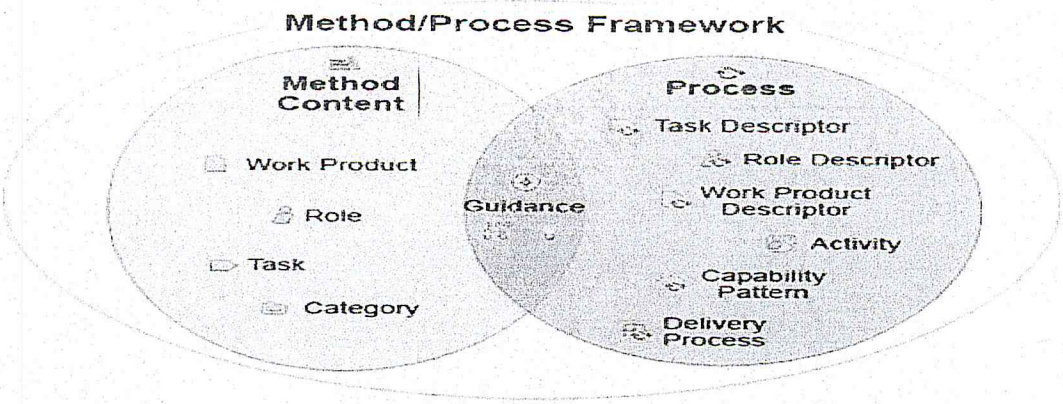


Figure 2 : Methode/Process Framework[4]

- SPEM fait une nette séparation des préoccupations entre Method Content et Process.
- Method Content définit le contenu hautement réutilisable.
- Process réutilise ce contenu pour créer des processus de bout en bout.[4]

Nous allons examiner chacune d'elles séparément, mais la chose importante à comprendre est que le Method Content ne définit pas quand les rôles effectuent les tâches, mais plutôt les relations entre les rôles, les tâches, les produits et conseils associés.

Le processus définit le calendrier des tâches (prédécesseur/ relations successeur)

4.1. Method Content:

Les concepts considérés comme des « method content » sont:

4.1.1 Rôle [3]

- Les rôles définissent un ensemble de compétences et de responsabilités.
- Les rôles ne sont pas des individus.
- Certaines personnes de l'équipe de développement peuvent jouer des rôles multiples.
- Les rôles effectuent des tâches.
- Les rôles sont responsables des produits du travail (Work Products).

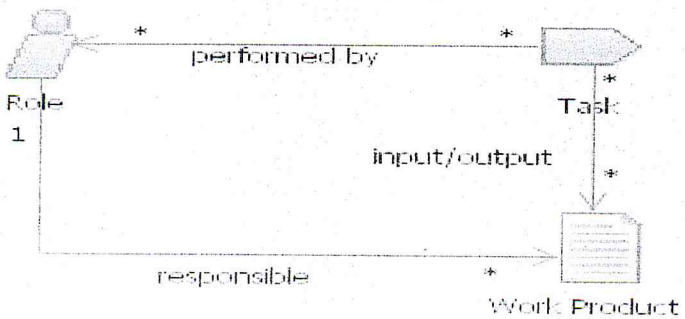


Figure : les liens entre les différents types

4.1.2 Produits de travail (Work Product)

- Les « Work Product » (dans la plupart des cas) représentent les informations tangibles utilisées, modifiées ou produites par une tâche.
- Les rôles utilisent les Work Product pour accomplir des tâches et produire des Work Product dans le cadre de l'exécution des tâches.
- Les Work Product sont la responsabilité d'un rôle.

Il existe trois types de Work Product :

- Artefact: généralement une configuration qui gère des **éléments**
- Deliverable: le client a exigé / intervenants livrable
- Outcome: "immatériel" résultat d'un travail comme un serveur installé ou un outil.

4.1.3 Tâche [3]

- Une tâche définit une unité assignable de travail (généralement quelques heures à quelques jours de longueur).
- Les tâches sont exécutées par des rôles (un primaire et éventuellement d'autres rôles de soutien).
- Les tâches ont un objectif clair, et fournissent une description étape par étape du travail qui doit être fait pour atteindre l'objectif.
- Les tâches permettent de modifier ou de produire des produits de travail.

4.1.4 Guidance (orientation)

Types of Guidance:
- Checklist
- Concept
- Example
- Guideline
- Estimate
- Considerations
- Practice
- Report
- Reusable Asset
- Roadmap
- Supporting Material
- Template
- Term Definition
- Tool Mentor
- Whitepaper

- **Checklists(Liste de contrôle) :** Elle identifie une liste de contrôle d'une série d'éléments qui doivent être complétées ou vérifiées. elles sont souvent utilisées dans des revues telles que des inspections.

- **Concept :**Il est associé avec les principes fondamentaux qui sous-entendent l'élément référencé.

- **Exemple :** Il fournit un exemple de produit achevé ou de tâches effectuées et des activités.

- **Guideline (Ligne directrice) :** Note d'orientation qui fournit des détails supplémentaires sur la façon d'effectuer une tâche particulière ou un groupe de tâches, ou qui fournit des détails supplémentaires, les règles et recommandations sur les produits de travail et de leurs propriétés.

- **Practice :** Représente la pratique d'une manière ou la stratégie éprouvée de faire un travail pour atteindre un objectif qui a un impact positif sur le produit du travail et la qualité des processus.

- **Report :** C'est un modèle pour une description générée automatiquement avec un contenu extrait d'un ou de plusieurs produit de travail.

- **Reusable Asset (Réutilisables Asset) :** Liens préemballés pour une solution à un problème dans un contexte donné.

- **Roadmap (Feuille de route) :** Une procédure pas à pas linéaire d'un processus complexe ou d'une activité. (Ceci est le guide de processus spécifiques.).

- **Supporting Material (Documents à l'appui) :** Un fourre-tout pour d'autres types d'orientation qui ne sont pas expressément définis ailleurs. Il peut être lié à toutes sortes d'éléments de contenu.

- **Template (Modèle) :** Pour un produit de travail, fournit un tableau prédéfini du contenu, des articles, des emballages, et / ou des positions, un format normalisé, ainsi que des descriptions sur la façon dont les sections et les colis sont sensés être utilisés et complétés.

- **Term Definition :** Définit la terminologie et, est utilisé pour construire le glossaire.

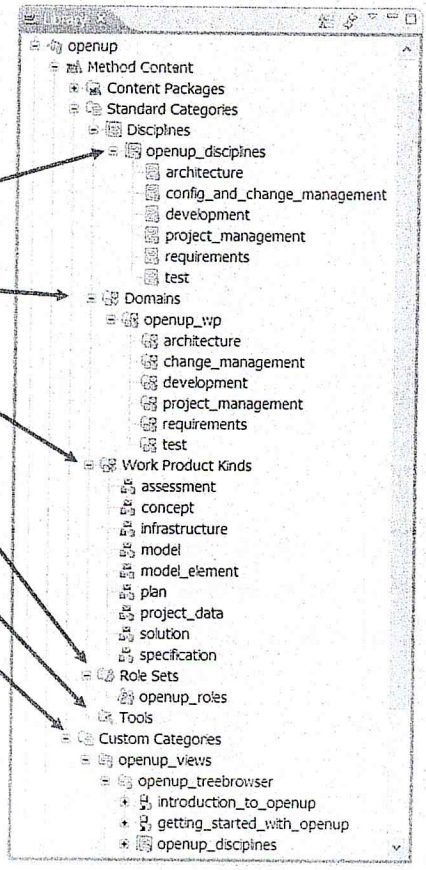
- **Tool Mentor :** Montre comment utiliser un outil spécifique pour accomplir un ouvrage.

- **Whitepaper (Livre blanc) :** semblable au concept, et peut être lu et compris isolément des autres éléments du contenu et des orientations.

4.1.5 Catégories

Les catégories peuvent être utilisées pour catégoriser le contenu fondé sur des critères de l'utilisateur ainsi que, pour définir l'ensemble des arborescences de catégories imbriquées, permettant à l'utilisateur de naviguer de manière systématique et parcourir le contenu du procédé et des processus basés sur ces catégories. Par exemple, un «contrôle», catégorie qui regroupe tous les rôles, les produits de travail, les tâches, et les éléments d'orientation pertinents pour les essais.[3]

- Categories
  - _ Utilisé pour les éléments du groupe des méthodes.
- Il ya 5 catégories Standard
  - **Discipline**: regroupement des tâches liées
  - **Domain**: groupement de WP liés
  - **Work Product Kind**: Similaire au domaine
  - **Role Set** :Groupement des rôles liés
  - **Tool**: Regroupement des outils
- Les catégories peuvent être imbriquées
- Vous pouvez définir vos propres catégories personnalisées
- Les éléments peuvent être classés par leur rédacteur en chef de propriété, ou via les propriétés de la catégorie.
- Utilisé pour créer des vues dans le site Web publié .



4.2 Process Content:

Les concepts de base qui constituent le Process Content sont Capability Patterns et Delivery Process

4.2.1 Capability Patterns (Patterns capacité) : [2]

- Capability Patterns définit la séquence des tâches connexes, réalisées pour obtenir un plus grand objectif.
- La tâche peut être spécialisée pour un contexte donné (ex. supprimer les étapes, work products)

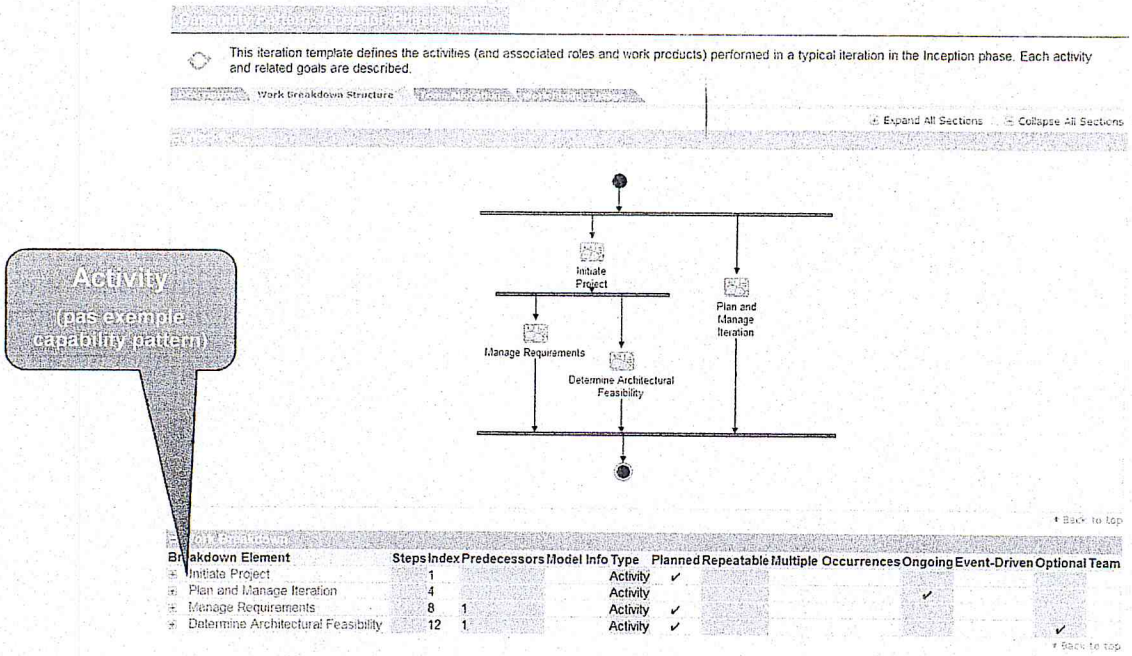


Figure 4 : Capability Pattern : Inception Phase Iteration[2]

4.2.2 Delivery Process (Processus de livraison) : [2]

Un processus de livraison définit une spécification de bout en bout pour atteindre un objectif (ex. une nouvelle version d'un logiciel). Vous pouvez penser que c'est le "haut niveau" d'activité. En plus des activités, les processus de livraison peuvent contenir des étapes, phases et itérations.

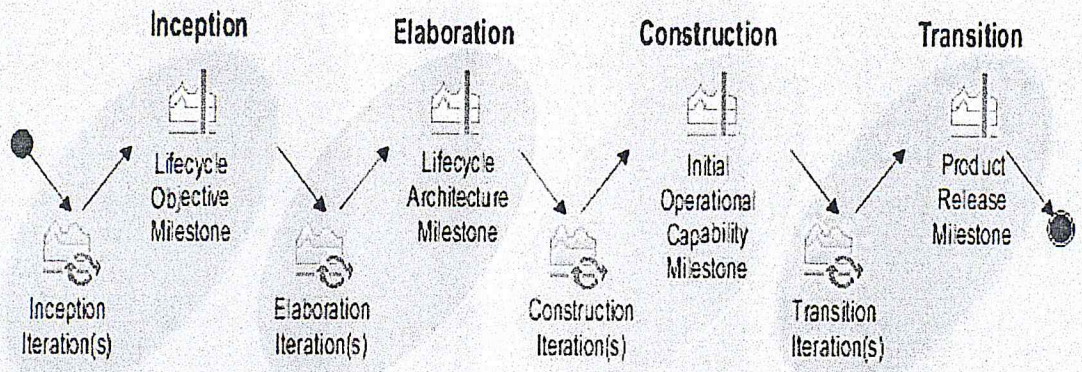
Ce n'est là qu'un exemple d'OpenUp , tout cycle de vie d'autre processus peut être défini.

Delivery Process : OpenUP lifecycle

This delivery process defines an end-to-end software development lifecycle that supports the core principles of OpenUP. It is designed to support small, co-located teams in their daily activities.

Description | Work Breakdown Structure | Team Allocation | Work Product Usage

Workflow



Work Breakdown

Breakdown Element	Steps Index	Predecessors	Model Info Type	Planned	Repeatable
Inception Iteration [1..n]	1		Activity	✔	
Initiate Project	2		Activity	✔	
Plan and Manage Iteration	5		Activity		
Identify and Refine Requirements	9	2	Activity	✔	
Agree on the Technical Approach	13	2	Activity	✔	
Lifecycle Objectives Milestone	15	1	Milestone	✔	
Elaboration Iteration [1..n]	16	15	Activity	✔	✔
Lifecycle Architecture Milestone	46	16	Milestone	✔	
Construction Iteration [1..n]	47	46	Activity	✔	✔
Initial Operational Capability Milestone	68	47	Milestone	✔	
Transition Iteration [1..n]	69	68	Activity	✔	✔
Product Release Milestone	86	69	Milestone	✔	

Figure 5 : Delivery Process : OpenUp lifecycle [2]

5. EPF Composer Import et Export :

EPF Composer nous permet d'importer (et d'exporter) une configuration, un plug-in, ou XML brut. On peut même exporter une configuration, un plug-in, XML brut ou MS Project Template [5]

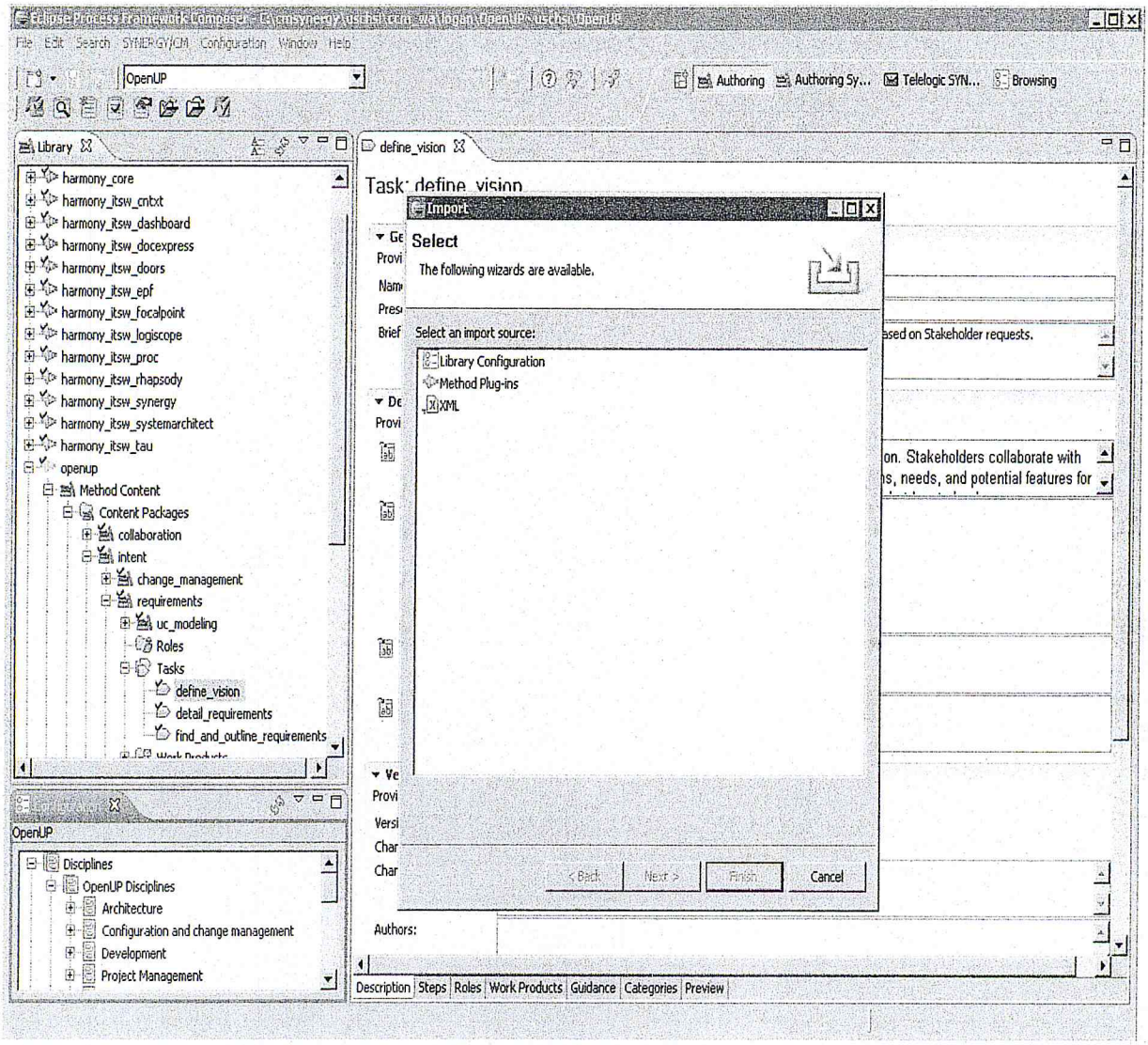


Figure : Import & Export

Remarque : Dans notre application nous allons générer un fichier xml, ensuite l'importer grâce à EPF Composer comme on vient de voir.

Conclusion :

Nous avons introduit dans ce chapitre les notions de base de EPF Composer et les concepts de SPEM utilisés et, à la fin du chapitre on a expliqué comment importer un fichier XML dans EPF composer,

Notre prochain objectif sera la génération d'un fichier XML (grâce à notre programme) qui respecte les règles de EPF Composer pour qu'il puisse être ouvert et lu par EPF Composer.

Et pour cela nous allons expliqué le déploiement d'une architecture logicielle dans le prochain chapitre.

Chapitre 3

Principe de déploiement d'architecture de procédé logiciel

Chapitre 3 : Principe de déploiement d'architecture de procédés logiciels

1.Introduction :

Dans ce chapitre nous présentons le principe du déploiement d'architecture de PL.
La notion d'architecture de PL est récente, très peu de travaux ont abordé cette notion,
Par conséquent nous détaillons la structure des fichiers qui rentre dans notre déploiement.
Architecture de procédé logiciel.

2. Déploiement d'une architecture logicielle :

D'après le "Petit Robert", le verbe déployer a pour définition " développer dans toute son extension (une chose qui était pliée)". Parmi ses synonymes, il est possible de trouver les verbes : déplier, ouvrir et mettre en œuvre. En informatique en particulier, le déploiement consiste donc à développer (dans le sens de déplier) dans toute son extension une entité logicielle pliée (c'est-à-dire packagée).

Notre travail est d'implémenter cette étape de déploiement qui est la dernière étape d'une approche de modélisation de PL à base d'architectures logicielles

Dans notre programme nous allons utiliser une configuration décrite en ACME et un ou plusieurs composants (.xml) et connecteurs (.xml) selon nos besoins (la configuration, les composants et les connecteurs sont définis et, constituent le résultat des étapes précédentes de l'approche), notre programme va nous permettre de déployer l'architecture de PL en un fichier (.xml), et qu'on pourra ouvrir avec Epf-Composer.

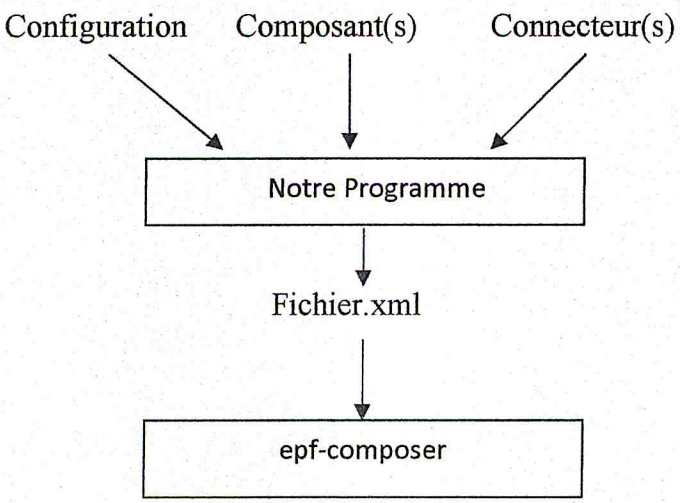


Figure 1 : Déploiement d'architecture de PL.

Dans ce qui suit, nous détaillons les fichiers rentrant des éléments architecturaux utilisés lors du déploiement (Configuration, Connecteur, Composants).

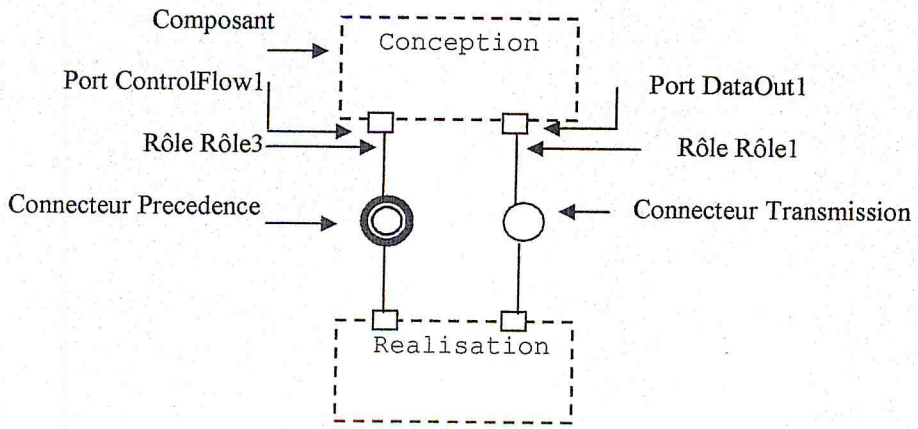
3. Configuration :

Notre configuration est faite grâce au logiciel AcmeStudio qui est un environnement d'édition personnalisable et, outil de visualisation pour les conceptions architecturales de logiciels basés sur le langage de description d'architectures ACME (ADL). Avec AcmeStudio, vous pouvez définir des familles ACME nouvelles et, personnaliser l'environnement de travailler avec ces familles en définissant des styles diagrammes. AcmeStudio est un front-end (programme servant d'interface graphique) adaptable qui peut être utilisé dans une variété d'applications de modélisation et d'analyse. AcmeStudio est implémenté comme un plugin pour Eclipse environnement, une source ouverte en environnement de développement Java intégré. Eclipse fournit un plugin-environnement permettant des extensions faciles d'AcmeStudio avec de nouvelles analyses et de fonctionnalité, et la personnalisation de nouveaux environnements architecturaux adaptés à une organisation particulière. [1]

3.1. La structure de la configuration est la suivante :

```
System System = {

    Component Conception  = {
        Port ControlFlow1  = {}
        Port DataOut1  = {}
    }
    Component Realisation  = {
        Port ControlFlow2  = {}
        Port DataIn2  = {}
    }
    Connector Transmission  = {
        Role Role1  = {}
        Role Role2  = {}
    }
    Connector Precedence  = {
        Role Role3  = {}
        Role Role4  = {}
    }
    Attachment Conception.DataOut1 to Transmission.Role1;
    Attachment Conception.ControlFlow1 to Precedence.Role3;
    Attachment Realisation.DataIn2 to Transmission.Role2;
    Attachment Realisation.ControlFlow2 to Precedence.Role4;
}
```



Les attachements nous donnent les liens entre un composant et un connecteur
Exemple : `Attachment Conception.DataOut1 to Transmission.Role1;`

Ici le composant « Conception » à le port « DataOut1 » qui est relié au Rôle « Rôle1 » du connecteur « Transmission »

Remarque :

La configuration ne décrit pas l'état interne des composants atomiques. Un composant procédé est constitué d'activités, dans notre cas, on ne peut pas savoir combien il y'a d'activités dans chaque composant. C'est la structure des composants (stockés dans un autre fichier) qui nous donne le nombre d'activités de chaque composant.

4. Les Connecteurs :

Il existe 3 catégories de connecteurs procédés : ceux spécifiques à l'envoi des données (DataFlowConnector) comme transmission , fusion , diffusion .... et ceux qui gèrent le déroulement (ControlFlowConnector) comme le connecteur précédence... et d'autres qui font l'envoi en même temps ils gèrent le déroulement comme le connecteur EoTemps.....

Les données du connecteur sont stockées dans un fichier XML qui permettra de décrire ses fonctionnalités. Le connecteur est un fragment de procédé logiciel constitué d'une activité qui porte le nom de la fonctionnalité du connecteur.

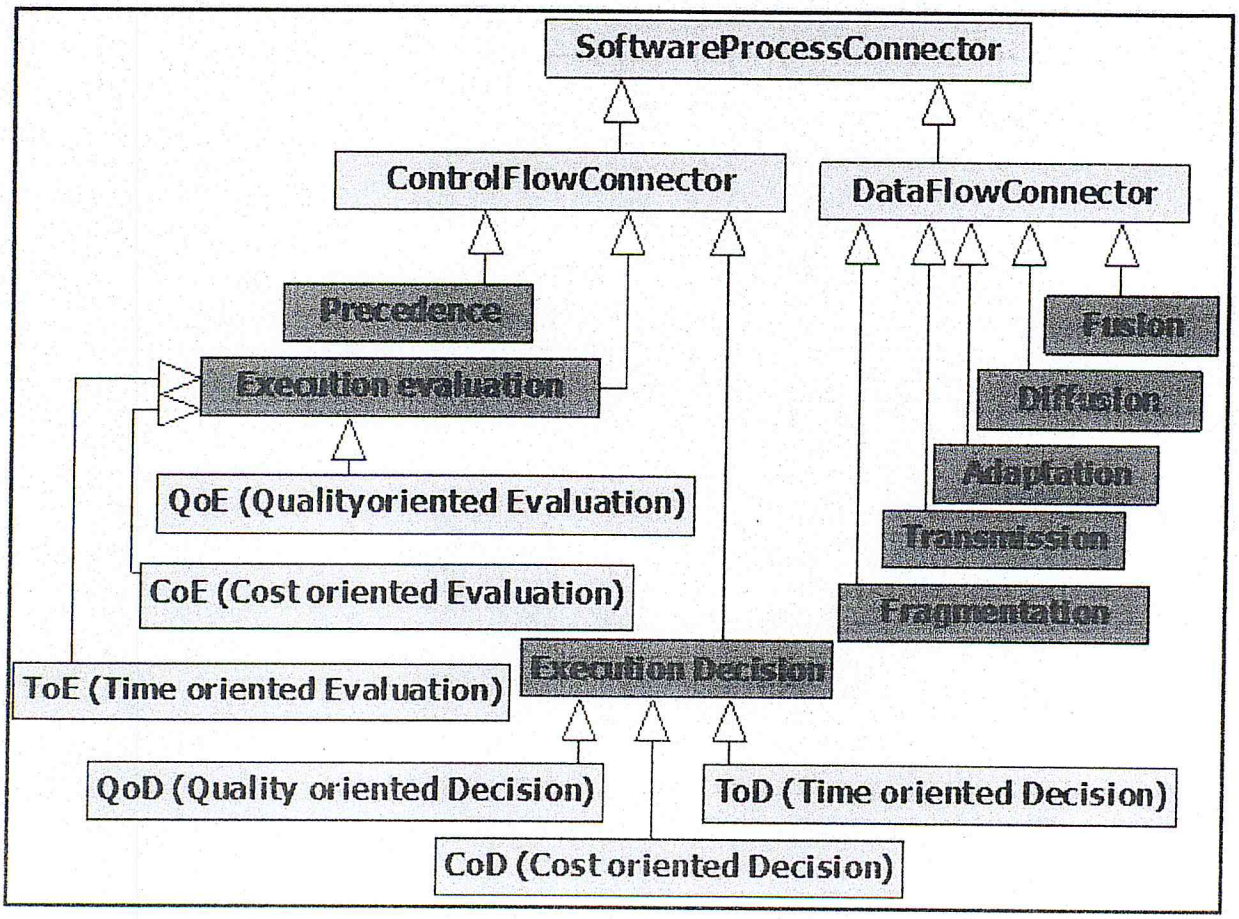


Figure 2 : Taxonomie des connecteurs procédés [2]

Dans ce qui suit nous détaillons les différentes structures possibles des connecteurs existantes dans les 3 catégories puis nous citons à la fin les fonctionnalités des autres connecteurs.

4.1. Connecteur Transmission :

Il s'occupe de la transmission des données (produit) d'un composant à l'autre. Le connecteur transmission va chercher à relier le DataOut1 d'un composant avec le DataIn2 d'un autre composant.

La structure de ce type de connecteur est caractérisée par le faite d'avoir une entrée et une sortie de type data ce qui se traduit par l'utilisation d'une balise <pro_WorkProductUseIn>DataOut1</pro_WorkProductUseIn> Et d'une autre balise <pro_WorkProductUseOut>DataIn2</pro_WorkProductUseOut>.

```
<FragmentDeProcede>
<Activity>
<Task_Use>Transmission</Task_Use>
<Process_Performer>M.Boutarouk</Process_Performer>
<Role_Use>RC1</Role_Use>
<pro_WorkProductUseIn>DataOut1</pro_WorkProductUseIn>
<pro_WorkProductUseOut>DataIn2</pro_WorkProductUseOut>
</Activity>
</FragmentDeProcede>
```

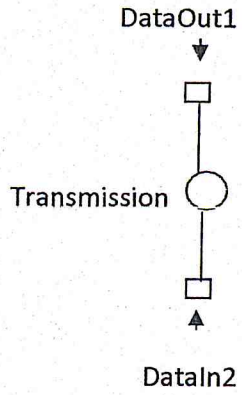


Figure 3 : Structure d'un connecteur de Transmission.

4.2. Connecteur Fusion :

Permet de fusionner un ensemble de produits rentrant en un seul produit sortant.

```
<FragmentDeProcede>
<Activity>
<Task_Use>fusion</Task_Use>
<Process_Performer>M.Boutarouk</Process_Performer>
<Role_Use>RC1</Role_Use>
<pro_WorkProductUseIn>DateOut1</pro_WorkProductUseIn>
<pro_WorkProductUseIn>DateOut2</pro_WorkProductUseIn>
<pro_WorkProductUseOut>DateIn3</pro_WorkProductUseOut>
</Activity>
</FragmentDeProcede>
```

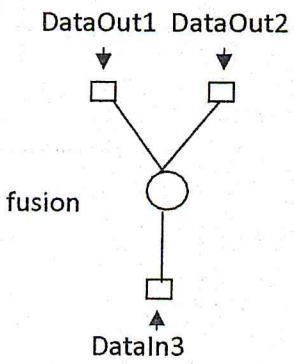


Figure 4 : Structure d'un connecteur de fusion.

On remarque qu'il y'a deux <pro_WorkProductUseIn> et un seul <pro_WorkProductUseOut> qui montre la fusion des deux composants auxquels ils appartiennent le DataOut1 et DataOut2 avec le composant auquel il appartient le DataIn3.

4.3. Connecteur Diffusion :

Il Permet de récupérer les produits d'un composant, puis de les diffuser sur un ensemble de composants

Structure d'un connecteur de diffusion

```
<FragmentDeProcede>
<Activity>
<Task_Use>diffusion</Task_Use>
<Process_Performer>M.Boutarouk</Process_Performer>
<Role_Use>RC1</Role_Use>
<pro_WorkProductUseIn>DateOut1</pro_WorkProductUseIn>
<pro_WorkProductUseOut>DateIn2</pro_WorkProductUseOut>
<pro_WorkProductUseOut>DateIn3</pro_WorkProductUseOut>
</Activity>
</FragmentDeProcede>
```

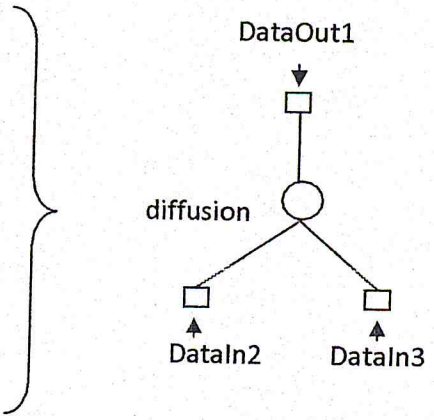


Figure 5 : Structure d'un connecteur de diffusion.

On remarque qu'il y'a deux <pro_WorkProductUseOut> et un seul <pro_WorkProductUseIn> qui montre la diffusion du composant auquel il appartient le DataOut1 avec les composants auxquels ils appartiennent le DataIn2 et le DataIn3.

4.4. Connecteur de Précédence :

Il permet d'ordonner l'exécution d'un ensemble de composants. Dans ce type de connecteur leurs rôles sont de type « control flow », par conséquent les balises utilisées sont :
<predecessor>ControlFlow1</predecessor> et
<successor>ControlFlow2</successor>

Sa structure est la suivante :

```
<FragmentDeProcede>

<Activity>
<Task_Use>Precedence</Task_Use>
<Process_Performer>M.Boutarouk</Process_Performer>
<Role_Use>RC2</Role_Use>
<predecessor>ControlFlow1</predecessor>
<successor>ControlFlow2</successor>
</Activity>
</FragmentDeProcede>
```

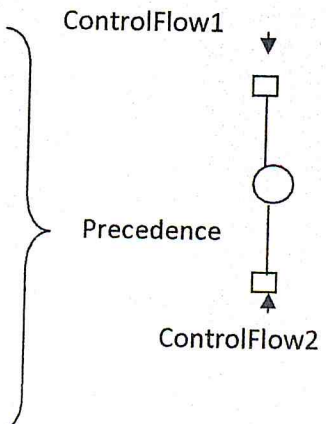


Figure 7 : Structure d'un connecteur de précédence.

Ici, le prédécesseur est ControlFlow1 par conséquent il s'exécute avant ControlFlow2 (successeur).

On dit aussi que ControlFlow1 précède ControlFlow2.

4.5.Connecteur de EoTemps , EoCost, EoQuality, Decision, DecisionCostgarde, DecisionQualitygarde :

Connecteur EoTemps Nom de la tâche : Evaluation of time.
Connecteur EoCost Nom de la tâche : Evaluation du Coût.
Connceteur EoQuality Nom de la tâche : Evaluation de la qualité.
Connecteur DecisionTemps garde le même nom de la tâche.
Connecteur DecisionCost garde le même nom de la tâche.
Connecteur DecisionQuality garde le même nom de la tâche.

Ils permettent l'envoi des données en même temps, gèrent le déroulement Et ne relient que deux composants entre eux (leurs structures comportent un seul <pro_WorkProductUseIn> , un seul <pro_WorkProductUseOut>, et un seul <predecessor> et <successor>).

Voilà le schéma d'un connecteur EOTemps:

```
<FragmentDeProcede>
<Activity>
<Task_Use>EoTemps</Task_Use>
<Process_Performer>M.Boutarouk</Process_Performer>
<Role_Use>R1</Role_Use>
<pro_WorkProductUseIn>DataOut1</pro_WorkProductUseIn>
<pro_WorkProductUseOut>DataIn2</pro_WorkProductUseOut>
<predecessor>ControlFlow1</predecessor>
<successor>ControlFlow2</successor>
</Activity>
</FragmentDeProcede>
```

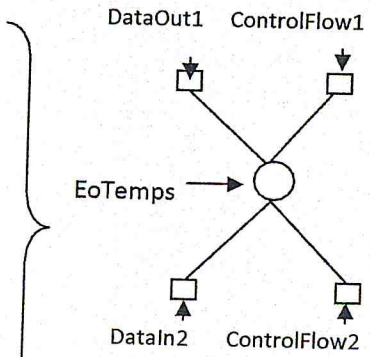


Figure 8 : Structure d'un connecteur EoTemps.

Grâce à <pro_WorkProductUseIn> et <pro_WorkProductUseOut> il va contrôler l'envoi des données et en même temps le déroulement grâce à <predecessor> et <successor>.

4.6. Exemple illustratif avec les connecteurs « Transmissions » et « Précédences »

Voilà un exemple en cycle de vie en V :

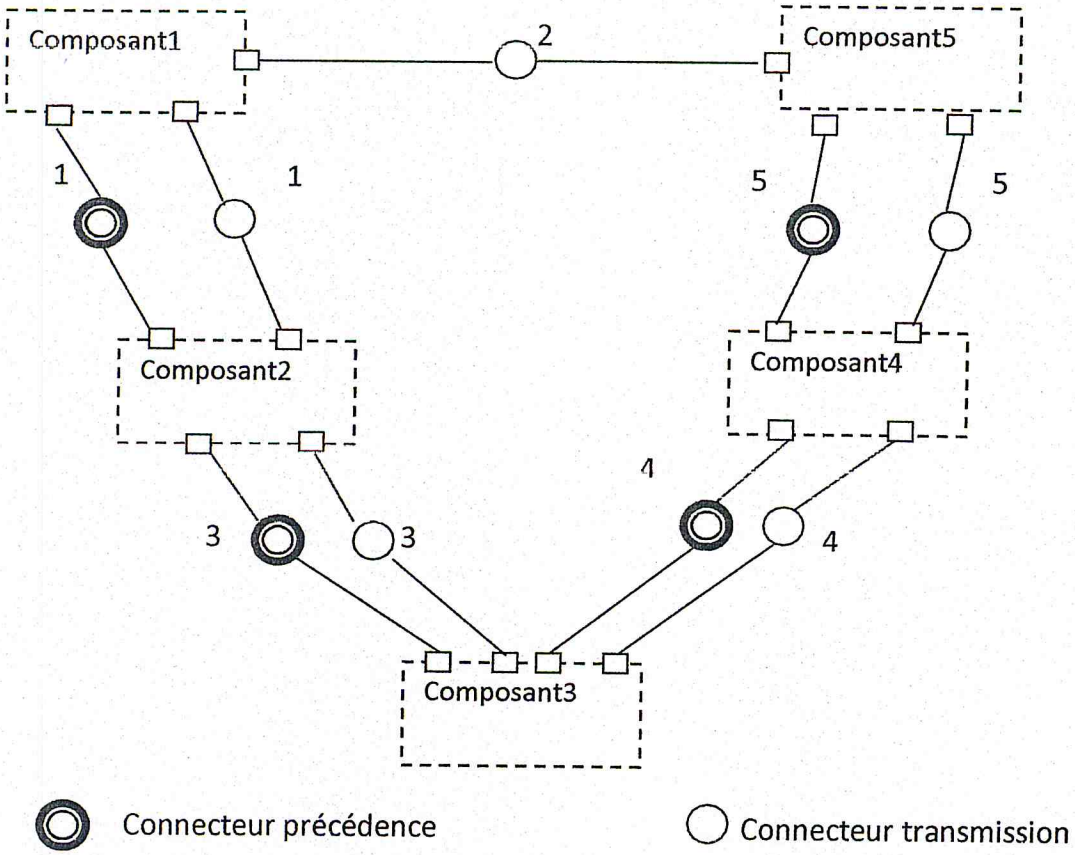


Figure 9 : Structure suivant le cycle de vie en V.

Cet exemple permet d'illustrer la manière d'exploiter les connecteurs procédés définis auparavant. Ainsi, dans cet exemple le composant1 va s'exécuter en premier, et en fin d'exécution, envoie les données à travers transmission 1 et 2 et ainsi déclencher précédence1. Le composant2 reçoit les données grâce à transmission1 et le signal de précédence1 pour s'exécuter, ainsi il peut commencer son exécution.

A la fin de l'exécution du composant1 il envoie les données à composant5 grâce à transmission2, cependant, le composant5 ne va pas s'exécuter car il n'a pas encore reçu le signal de precedence5 et il doit attendre son tour qui se fera après l'exécution du composant4 et ainsi de suite.

5. Les Composants :

Un composant commence toujours par <FragmentDeProcede> et se termine par </FragmentDeProcede>, et peut contenir une ou plusieurs activités.
Un composant a la structure suivante :

<FragmentDeProcede>
<Activity> </Activity>
<Activity> </Activity>
<Activity></Activity>
.....
</FragmentDeProcede>

Exemple d'un composant avec une seule activité :

```
<FragmentDeProcede>
<Activity>
<Task_Use>Conception</Task_Use>
<Process_Performer>M.Boutarouk</Process_Performer>
<Role_Use>R1</Role_Use>
<pro_WorkProductUseOut> DataOut1</pro_WorkProductUseOut>
</Activity>
</FragmentDeProcede>
```

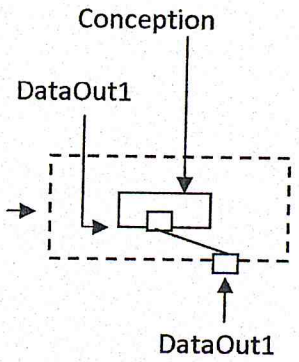


Figure 10 : Structure d'un composant.

Ce composant est une activité Conception qui est sous la responsabilité de M. Boutarouk. Son port est DataOut1.

6. Les Activités :

Une activité commence toujours par <Activity> et se termine par </Activity>
Une activité a la structure suivante :

```
<Activity>
<Task_Use> </Task_Use>
<Process_Performer> </Process_Performer>
<Role_Use> </Role_Use>
<pro_WorkProductUseIn> </pro_WorkProductUseIn>
<pro_WorkProductUseOut></pro_WorkProductUseOut>
<predecessor> </predecessor>
<successor></successor>
</Activity>
```

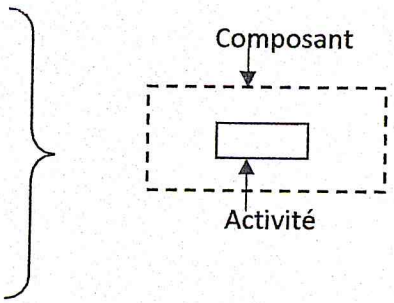


Figure 11 : Structure d'une Activité.

Explication :

Le <Task_Use> signifie la tâche à faire et <Process_Performer> est le réalisateur de cette tâche.

Le <pro_WorkProductUseIn> et le <pro_WorkProductUseOut> sont les liens entre les Activités le « In » pour l'entrée et le « Out » pour la sortie .

Le <predecessor> et le <successor> portent les noms des tâches qu'ils précédent ou succèdent.

Un composant peut avoir plusieurs activités

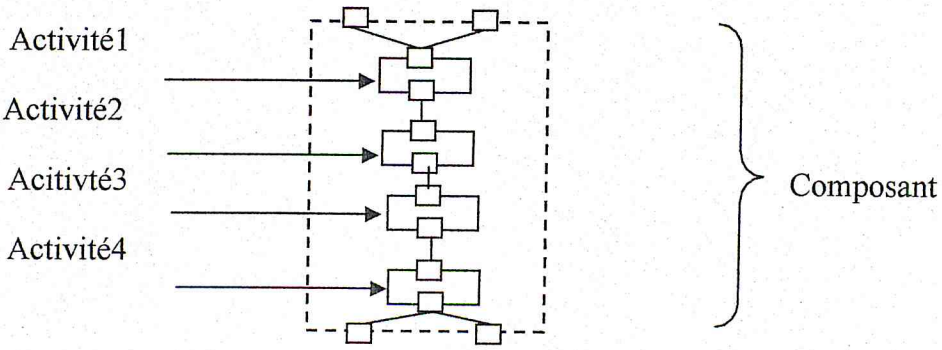
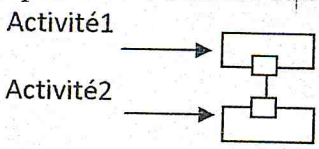


Figure 12 : Structure de plusieurs activités.

Les relations entre les activités :
Dans le cas le plus simple où une activité suit directement une autre



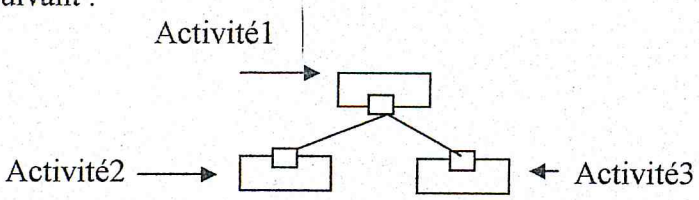
Le <pro_WorkProductUseOut> de la première activité doit avoir le même nom que <pro_WorkProductUseIn> de la deuxième activité.

Et le <successor> de la première activité doit avoir le même nom que le <Task_Use> de la deuxième activité (qui le suit).

Et la même chose pour <predecessor> de la deuxième activité qui doit avoir le même nom que <Task_Use> de la première activité (qui le précède).

```
<Activity>
<Task_Use>Activité1</Task_Use>
<Process_Performer>M.Khaled</Process_Performer>
<Role_Use>R1</Role_Use>
<pro_WorkProductUseOut>Data1</pro_WorkProductUseOut>
<successor>Activité2</successor>
</Activity>
<Activity>
<Task_Use>Activité2</Task_Use>
<Process_Performer>M.Boutarouk</Process_Performer>
<Role_Use>R2</Role_Use>
<pro_WorkProductUseIn>Data1</pro_WorkProductUseIn>
<predecessor>Activité1</predecessor>
</Activity>
```

Dans le cas suivant :

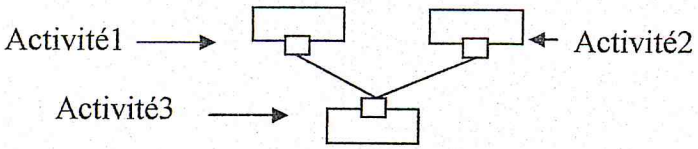


Nous allons avoir deux <successor> dans la première Activité1, un pour l'Activité2 et l'autre pour l'Activité3, et le <pro_WorkProductUseOut> va être équivalent à <pro_WorkProductUseIn> de la deuxième et la troisième Activité.

Et l'Activité2 et 3 vont avoir le même <predecessor> qui est « Activité1 »

```
<Activity>
<Task_Use>Activité1</Task_Use>
<Process_Performer>M.Bourarouk</Process_Performer>
<Role_Use>R1</Role_Use>
<pro_WorkProductUseOut>Data1</pro_WorkProductUseOut>
<successor>Activité2</successor>
<successor>Activité3</successor>
</Activity>
<Activity>
<Task_Use>Activité2</Task_Use>
<Process_Performer>M.Korichi</Process_Performer>
<Role_Use>R2</Role_Use>
<pro_WorkProductUseIn>Data1</pro_WorkProductUseIn>
<predecessor>Activité1</predecessor>
</Activity>
<Activity>
<Task_Use>Activité3</Task_Use>
<Process_Performer>M.Djili</Process_Performer>
<Role_Use>R3</Role_Use>
<pro_WorkProductUseIn>Data1</pro_WorkProductUseIn>
<predecessor>Activité1</predecessor>
</Activity>
```

Dans le cas suivant :



Nous allons avoir dans l'Activité3 deux <predecessor> , un pour l'Activité1 et l'autre pour l'Activité2

```
<Activity>
<Task_Use>Activité1</Task_Use>
<Process_Performer>M.Boutarouki</Process_Performer>
<Role_Use>R1</Role_Use>
<pro_WorkProductUseOut>Data1</pro_WorkProductUseOut>
<successor>Activité3</successor>
</Activity>
<Activity>
<Task_Use>Activité2</Task_Use>
<Process_Performer>M.Korichi</Process_Performer>
<Role_Use>R2</Role_Use>
<pro_WorkProductUseOut>Data1</pro_WorkProductUseOut>
<successor>Activité3</successor>
</Activity>
<Activity>
<Task_Use>Activité3</Task_Use>
<Process_Performer>M.Samir</Process_Performer>
<Role_Use>R3</Role_Use>
<pro_WorkProductUseIn>Data1</pro_WorkProductUseIn>
<predecessor>Activité1</predecessor>
<predecessor>Activité2</predecessor>
</Activity>
```

7. Conclusion :

Dans ce chapitre nous avons expliqué les principes de déploiement d'architecture de procédé logiciel ensuite, on a présenté les fichiers représentant les éléments architecturaux rentrant dans le déploiement d'architecture logicielle. Dans ce qui suit nous présentons la conception de notre application.

Chapitre 4

Conception

1. Introduction :

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

2. Choix de la méthode de conception (processus de développement):

La méthode de conception permet de décrire d'une manière explicite (souvent avec un langage de modélisation) les étapes à utiliser pour suivre la réalisation des fonctionnalités d'un système logiciel.

Le choix de la méthode a été fait selon le domaine et le type de projet, pour notre cas nous avons choisi le cycle de vie en cascade basé sur UML, car c'est un cycle de vie simple et, il convient très bien à notre projet (pas de gestion formelle du projet et pas de planification rigoureuse).

2.1. Présentation générale d'UML : UML signifie Unified Modeling Language. .[1]

UML est un langage de modélisation (Unified Modeling Language), dont la mise au point est supervisée par un consortium de plusieurs entreprises (essentiellement américaines) : l'Object Management Group (OMG). Ce consortium comprend à la fois des éditeurs informatiques,et des entreprises utilisatrices. Ce dernier point est à mon sens important.
A noter que l'OMG n'a pas pour seul souci la maintenance d'UML.
A l'origine, UML est le résultat de la fusion de trois méthodes objets préexistantes :

- OMT de James Rumbaugh ;
- OOSE de Ivar Jacobson ;
- BOOCH de Grady Booch.

Cette initiative de fusionner ces méthodes (à une époque où les méthodes se déversaient sur le marché de manière quasi hémorragique) revient à Rational : éditeur indépendant de l'outil Rose.

UML propose 9 diagrammes :

- Cas d'Utilisation
- Classes
- Objets
- Séquences
- Collaboration
- États et Transitions
- Activité
- Composants
- Déploiement

2.2. Présentation de cycle de vie en cascade : [2]

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

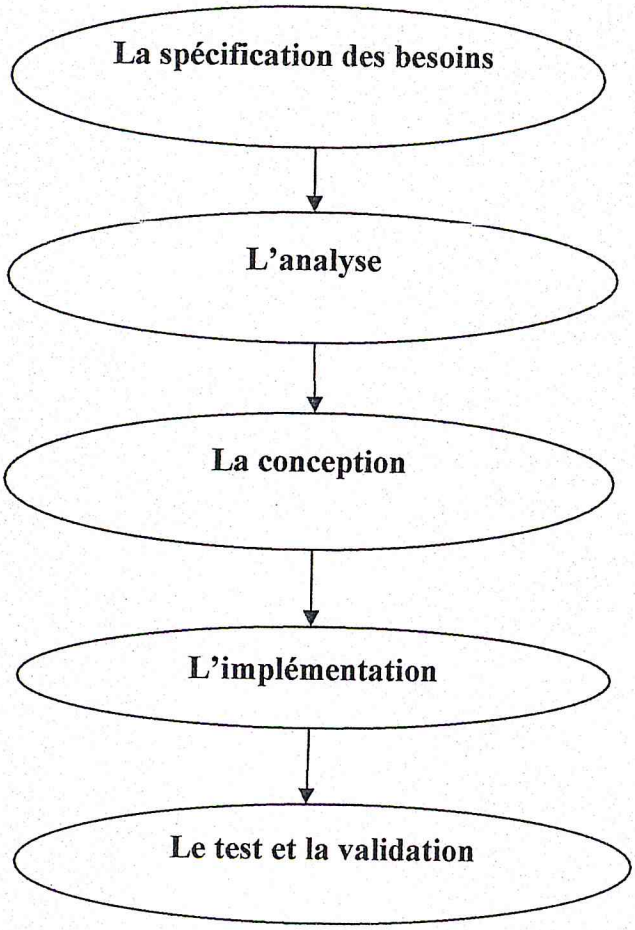


Figure 1 : cycle de vie d'un logiciel (modèle en cascade).

Analyse des besoins :

Cette phase permet d'avoir une vue globale du produit en identifiant tous les besoins d'utilisateur et, d'avoir une vue sur l'architecture du logiciel.

Besoins fonctionnels :

L'application doit réaliser les fonctionnalités suivantes :

L'utilisateur charge les composants et les connecteurs sous forme des fichiers xml en suivant une configuration donnée en ACME et clique sur le bouton générer pour obtenir un fichier epf en xml qui sera publié dans Eclipse Process Framework.

3. La conception :

3.1. Les cas d'utilisation globale :

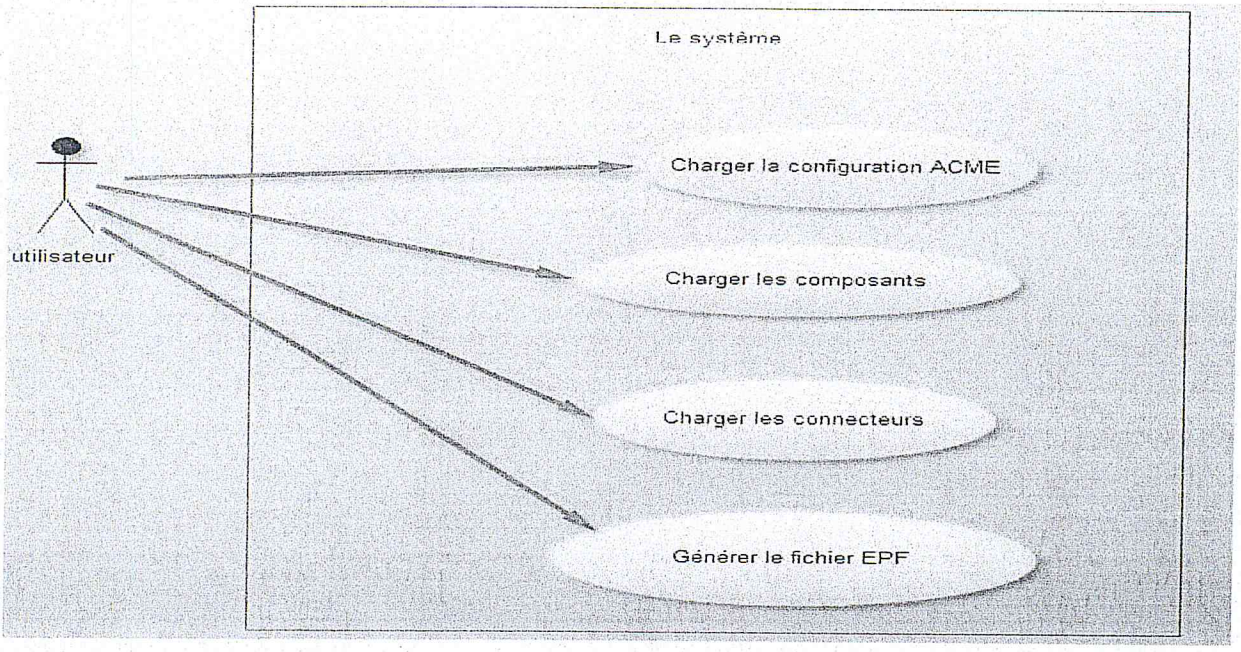


Figure 2 : Diagramme des cas d'utilisation globale.

3.2. Les cas d'utilisation et diagrammes de séquence détaillée:

3.2.1. Chargement de la configuration ACME :

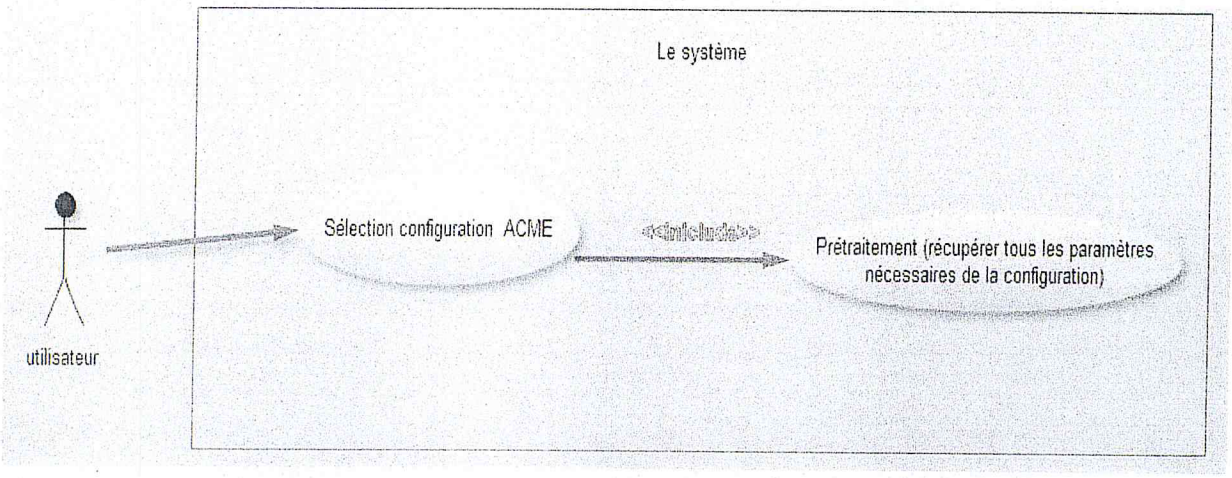


Figure 3 : Diagramme du cas d'utilisation détaillée.

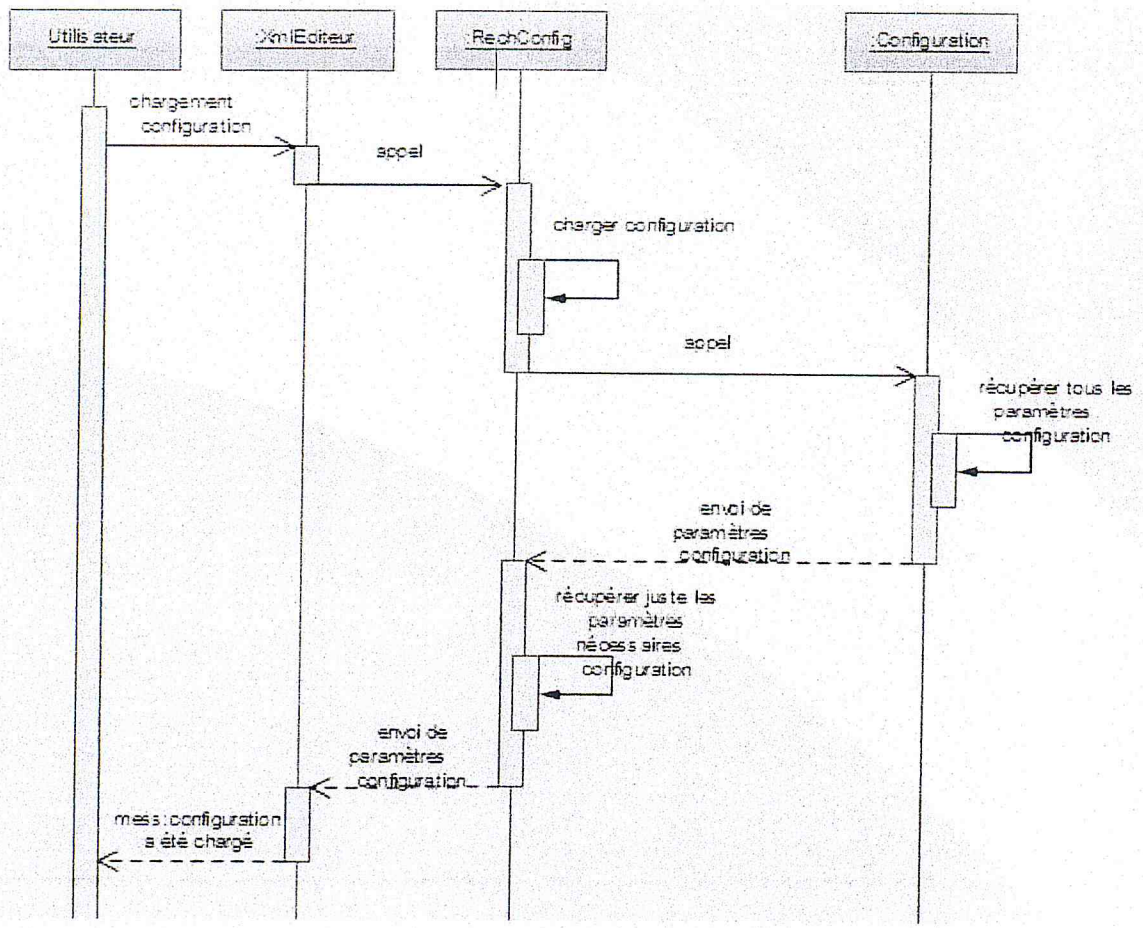


Figure 4 : Diagramme de séquence du cas d'utilisation détaillée.

Cas	Charger la configuration ACME
Résumé	Chargement d'une configuration ACME regroupant un enchaînement d'activités
Acteur principal	Utilisateur
Résultat	Récupérer tous les paramètres nécessaires de la configuration

Description	1. L'utilisateur lance le chargement de la configuration ACME. 2. la classe XmlEditeur fait un appel à la classe RechConfig qui charge la configuration donnée en ACME 3. la classe RechConfig fait un appel à la classe Configuration qui récupère tous les paramètres de la configuration et les envoie à la classe RechConfig. 4. la classe RechConfig récupère juste les paramètres nécessaires de la configuration et les envoie à la classe XmlEditeur .

Tableau 1 : Description du cas d'utilisation détaillée.

3.2.2. Chargement des composants :

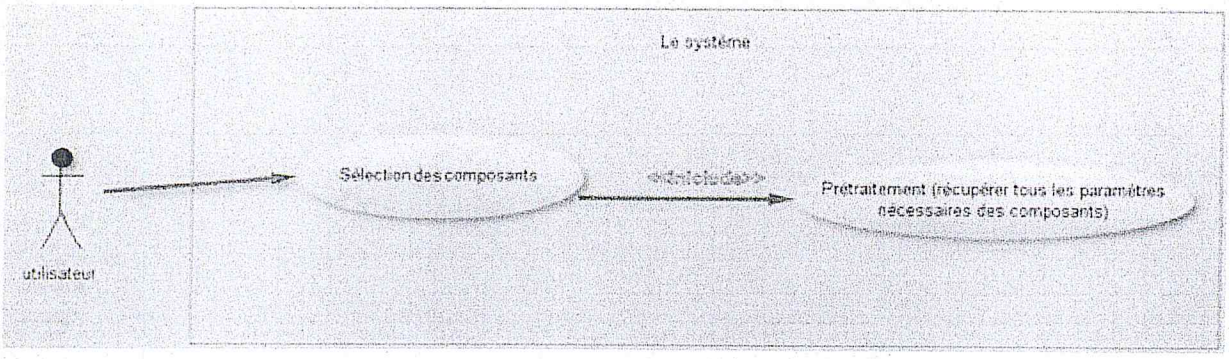


Figure 5 : Diagramme du cas d'utilisation détaillée.

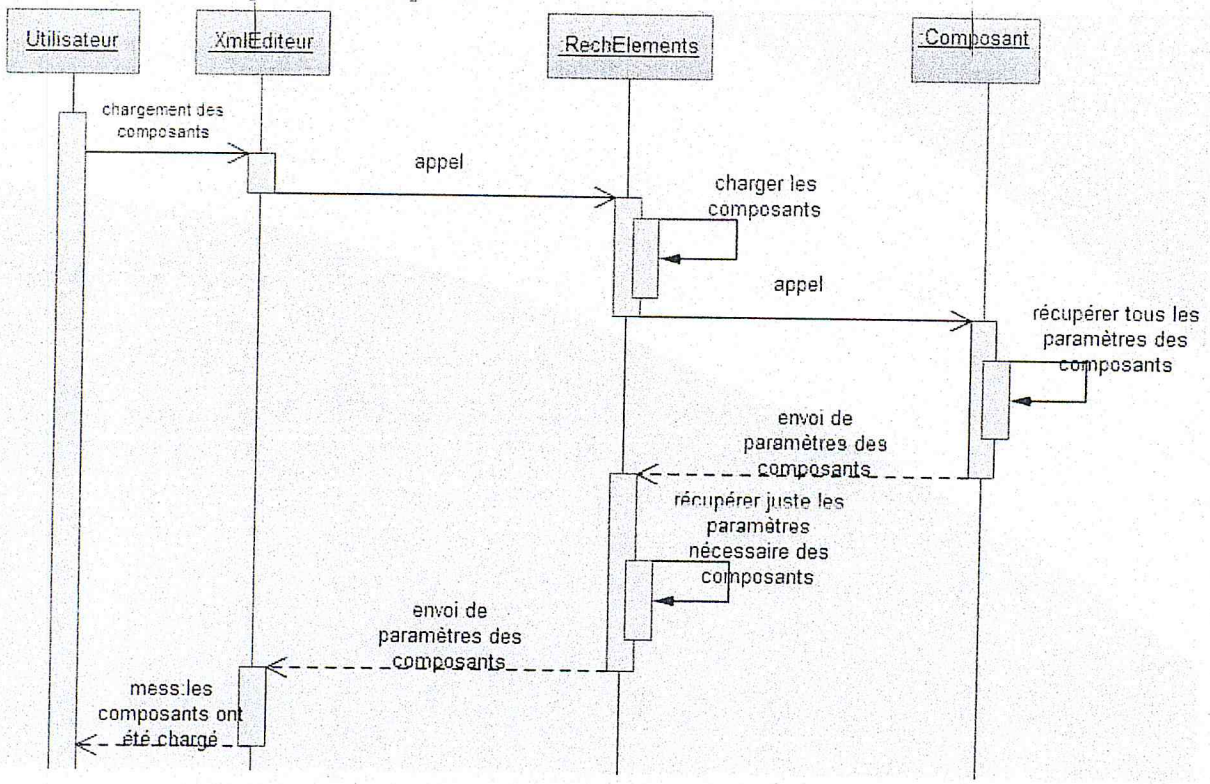


Figure 6 : Diagramme de séquence du cas d'utilisation détaillée.

Cas	Charger les composants
Résumé	Charger tous les composants nécessaires pour le déploiement en suivant la configuration ACME.
Acteur principal	Utilisateur
Résultat	Récupération de tous les paramètres des composants nécessaires pour générer le fichier EPF

Description	5. L'utilisateur lance le chargement des composants associés à la configuration. 6. la classe XmlEditeur fait un appel à la classe RechElements qui charge les composants sous format XML. 7. la classe RechElements fait un appel à la classe Composant qui récupère tous les paramètres des composants et les envoie à la classe RechElements. 8. la classe RechElements récupère juste les paramètres nécessaires des composants et les envoie à la classe XmlEditeur .

Tableau 2 : Description du cas d'utilisation détaillée« Charger les composants».

3.2.3. Chargement des connecteurs :

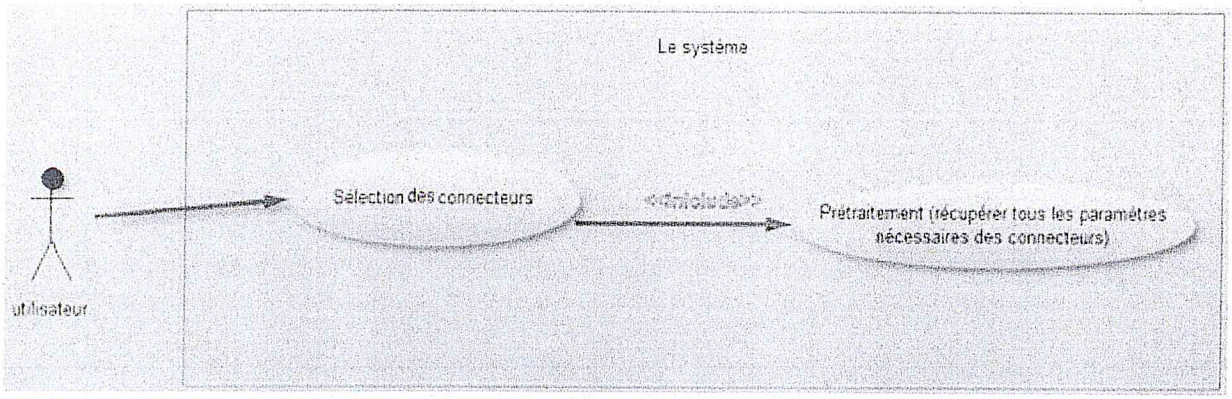


Figure 7 : Diagramme du cas d'utilisation détaillée.

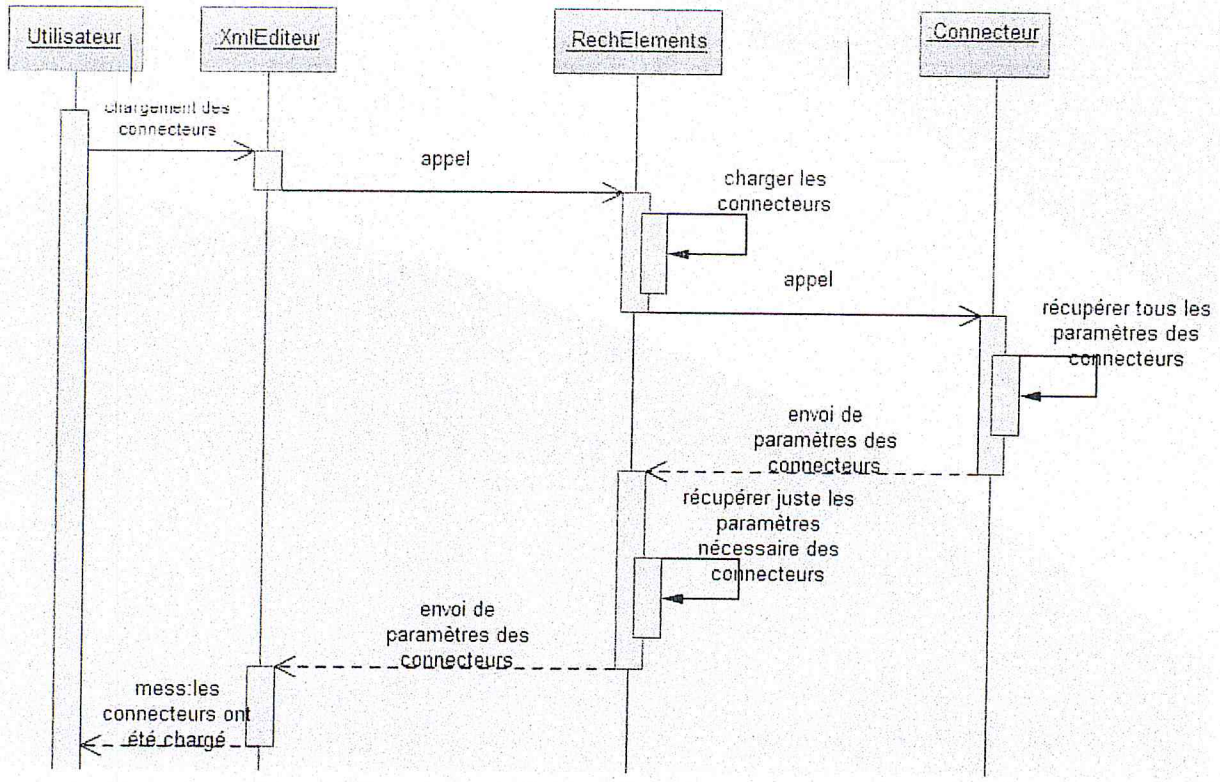


Figure 8: Diagramme de séquence du cas d'utilisation détaillée.

Cas	Charger les connecteurs
Résumé	Charger tous les connecteurs nécessaires pour le déploiement en suivant la configuration ACME.
Acteur principal	Utilisateur
Résultat	Récupération de tous les paramètres des connecteurs nécessaires pour générer le fichier EPF

Description	5. L'utilisateur lance le chargement des connecteurs associés à la configuration. 6. la classe XmlEditeur fait un appel à la classe RechElements qui charge les connecteurs sous format XML. 7. la classe RechElements fait un appel à la classe Connecteur qui récupère tous les paramètres des connecteurs et les envoie à la classe RechElements. 8. la classe RechElements récupère juste les paramètres nécessaires des connecteurs et les envoie à la classe XmlEditeur .

Tableau 3 : Description du cas d'utilisation détaillée.

3.2.4. Génération le fichier EPF (.XML) :

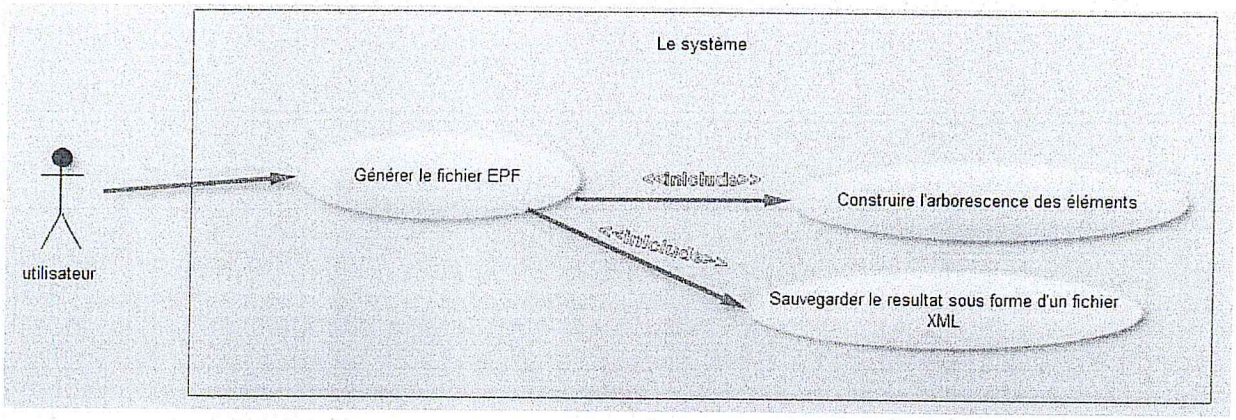


Figure 9 : Diagramme du cas d'utilisation détaillée.

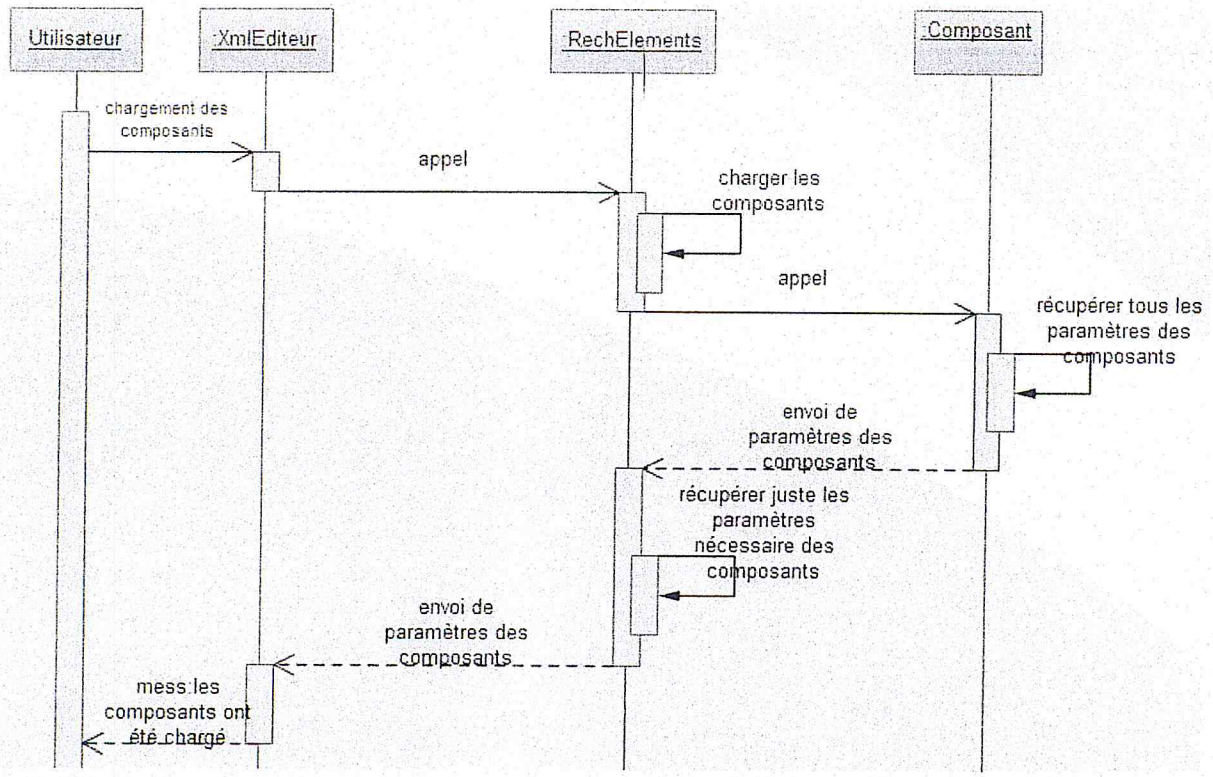


Figure 10 : Diagramme de séquence du cas d'utilisation détaillée.

Cas	Générer le fichier EPF
Résumé	Générer un fichier EPF sous format XML qui regroupe l'ensemble des éléments chargés.
Acteur principal	Utilisateur
Résultat	Un fichier EPF sous format XML regroupant un enchainement d'activités

Description	1. l'utilisateur génère le fichier EPF. 2. la classe XmlEditor fait un appel à la classe RechConfig pour comparer le produit d'entrée (ProdIn) au composant chargé par l'utilisateur. 3. la classe RechConfig cherche l'activité (act1) qui a comme entrée le ProdIn . 4. la classe RechConfig fait la comparaison entre le composant chargé par l'utilisateur et le composant(component) de la configuration ACME. 5. trouver le prodOut de l'activité act(1) du composant. 6. trouver l'activité act(2) qui a comme entrée le prodOut. 7. la classe RechConfig compare le connecteur associé au composant avec le connecteur (connector) associé au component de la configuration ACME. Si le connecteur et le connector sont équivalents, la classe RechConfig fait un appel à la classe XmlEditeur pour relier entre act(1) et act(2) du composant. 8. la classe XmlEditeur construit l'arborescence des éléments. 9. la classe XmlEditeur sauvegarde le résultat sous format d'un fichier XML.

Tableau 4 : Description du cas d'utilisation détaillée.

4. Diagramme de classe : [3]

Le diagramme de classe permet d'appréhender d'un point de vue logique la structure statique du système en indiquant :

- La structure des objets composants le système.
- Les structurations entre ces objets.

Le diagramme suivant présente les classes utilisées dans notre système et les relations entre elles.

Les interactions décrites dans les trois diagrammes de séquences permettent de déduire un diagramme de classe

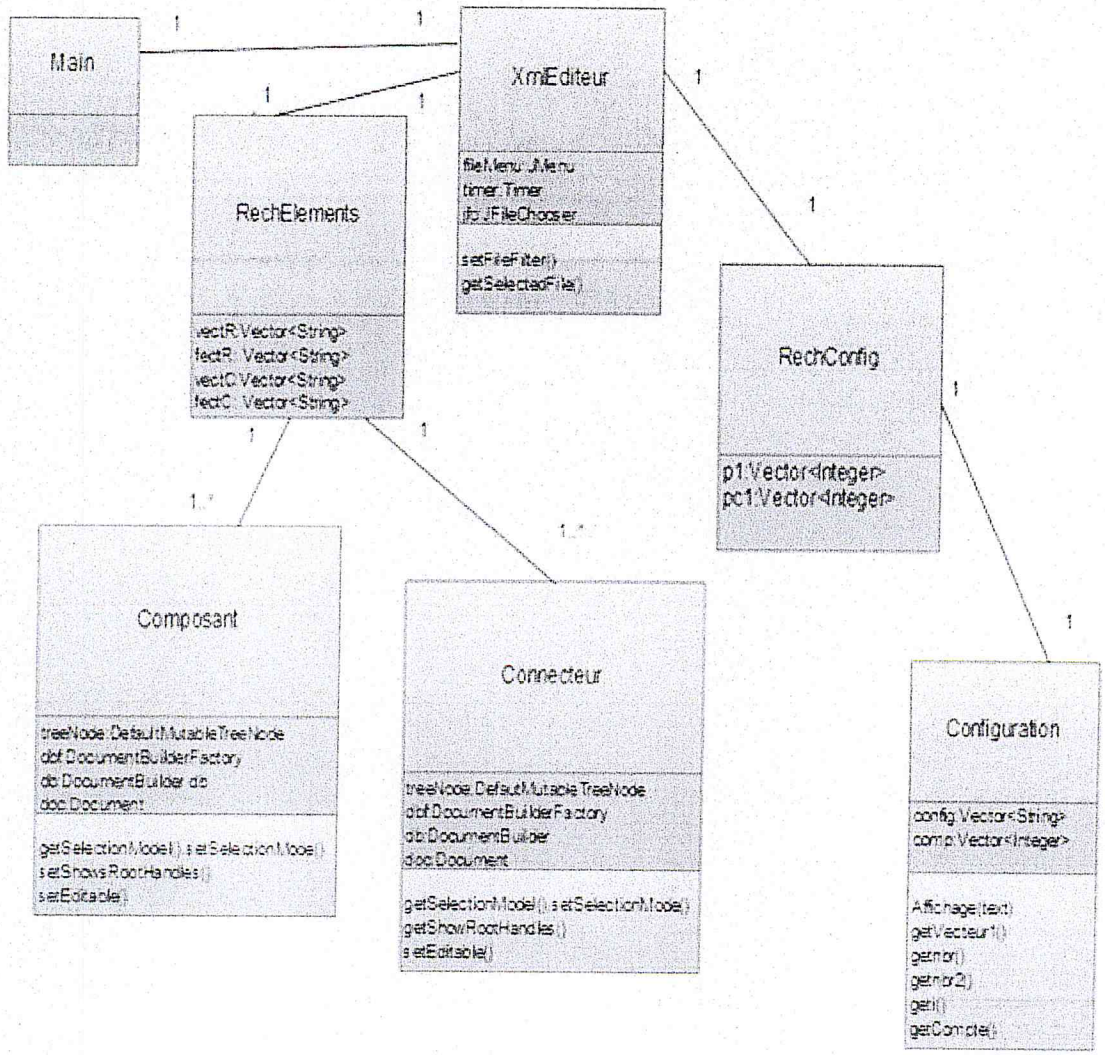


Figure 11 : Diagramme des classes de la conception.

Les classes représentées ci-dessus sont : soit la fenêtre principale, soit la classe XmlEditeur qui appelle les deux classes RechElements ,RechConfig .

La classe RechElements récupère les paramètres d'un seul ou plusieurs composants chargés par la classe Composant et récupère les paramètres d'un seul ou plusieurs connecteurs chargés par la classe Connecteur.

La classe RechConfig récupère les paramètres de la configuration chargée par la classe Configuration.

5. Conclusion :

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

Nous avons commencé par décrire le diagramme des cas d'utilisation globale et la description des cas d'utilisation détaillée. Ensuite, pour chaque cas, nous avons établi le diagramme des séquences correspondantes. Enfin, nous présentons le diagramme des classes de notre conception.

La prochaine étape, est l'implémentation de notre application. Ainsi, le chapitre suivant présente notre application et décrit les outils utilisés pour la réalisation de cette dernière.

Chapitre 5

Implémentation

1. Introduction :

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

Notre application permet d'analyser une configuration ACME, l'objectif du travail est de générer un fichier XML, en utilisant des composants, des connecteurs sous forme de fichiers XML et une configuration ACME, afin que le fichier XML (résultat du déploiement) soit ouvert et lu par EPF Composer.

Nous présentons également dans ce chapitre les différents outils et langages de programmation utilisés pour la réalisation de notre application.

2. Outils utilisés :

2.1. Eclipse IDE Java 2EE : [1]

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

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

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

2.2. Eclipse Process Framework Composer (EPF Composer): [2]

EPF Composer, outil développé dans le cadre du sous-projet officiel Eclipse Process Framework (EPF), est une application riche construite sur le framework Eclipse et utilisant abondamment les « plug-in » de haut niveau de l'écosystème.

Le but d'EPF Composer est d'offrir un outil de modélisation de processus complexes aux méthodologues, aux chefs de projets, aux directeurs de projet qui ont en charge la maintenance et l'implémentation d'un processus quelconque. Accessoirement, EPF Composer est utilisé au sein de l'outil Jazz pour modéliser les processus utilisés par l'application.

EPF Composer est un outil permettant à divers intervenants de décrire un processus, de quelque nature qu'il soit, même si l'intention première est de fournir un outil de support méthodologique à des approches telles que RUP, OpenUP, Scrum, XP ou encore EclipseWay.

2.3 .DOM: [3]

DOM est l'acronyme de Document Object Model. C'est une spécification du W3C pour proposer une API qui permet de modéliser, de parcourir et de manipuler un document XML. Le principal rôle de DOM est de fournir une représentation mémoire d'un document XML sous la forme d'un arbre d'objets et d'en permettre la manipulation (parcours,recherche et mise à jour).

A partir de cette représentation (le modèle), DOM propose de parcourir le document mais aussi de pouvoir le modifier. Ce dernier aspect est l'un des aspects les plus intéressants de DOM.

DOM est défini pour être indépendant du langage dans lequel il sera implémenté.DOM n'est qu'une spécification qui, pour être utilisée, doit être implémentée par un éditeur tiers. DOM n'est donc pas spécifique à Java.

Le parseur DOM pour JAVA le plus répandu est Xerces .

JDOM utilise DOM pour manipuler les éléments d'un Document Object Model spécifique (créé grâce à un constructeur basé sur SAX).

JDOM permet donc de construire des documents, de naviguer dans leur structure, ajouter, modifier, et supprimer leur contenu.

3. Présentation de l'application :

L'application que nous avons développée permet :

1. Le chargement de la configuration ACME, connecteurs (.XML) et composants (.XML).
2. La génération d'un fichier (.XML).
3. On doit pouvoir ouvrir notre résultat du déploiement grâce à EPF Composer (et pour cela on doit respecter la structure de EPF Composer).

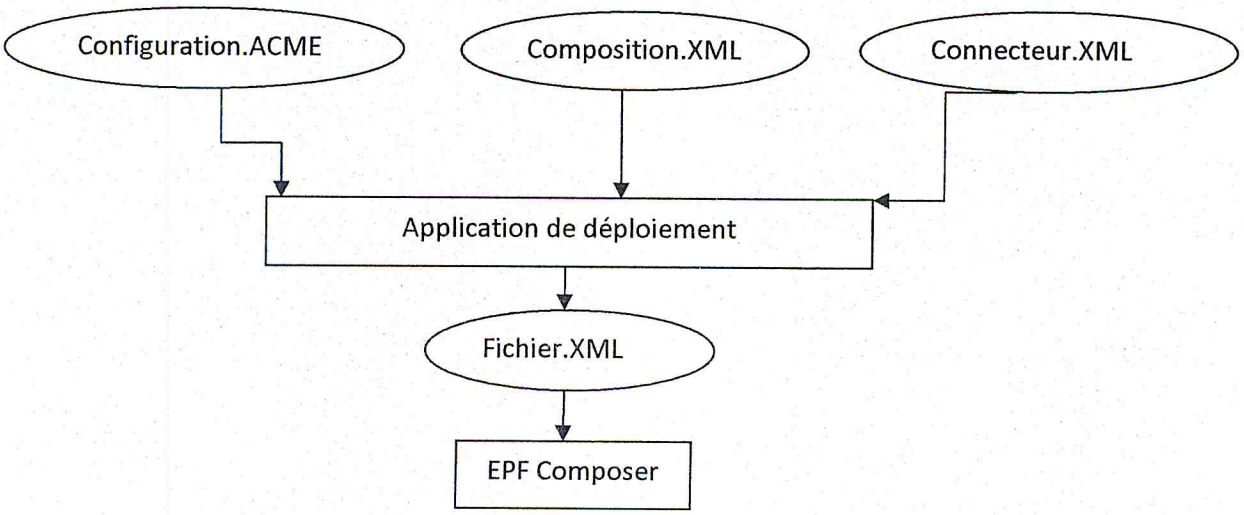


Figure 1 : Schéma de l'application

Nous suivons le schéma sur la figure 2 pour la génération du fichier (.XML).

NbrDem : Nombre de composants nécessaires à notre application (le nombre est récupéré de la configuration).

NbrComp : Il va s'incrémenter à chaque passage.

NbrDem2: Nombre de connecteurs nécessaires à notre application (le nombre est récupéré de la configuration).

NbrConn : Il va s'incrémenter à chaque passage.

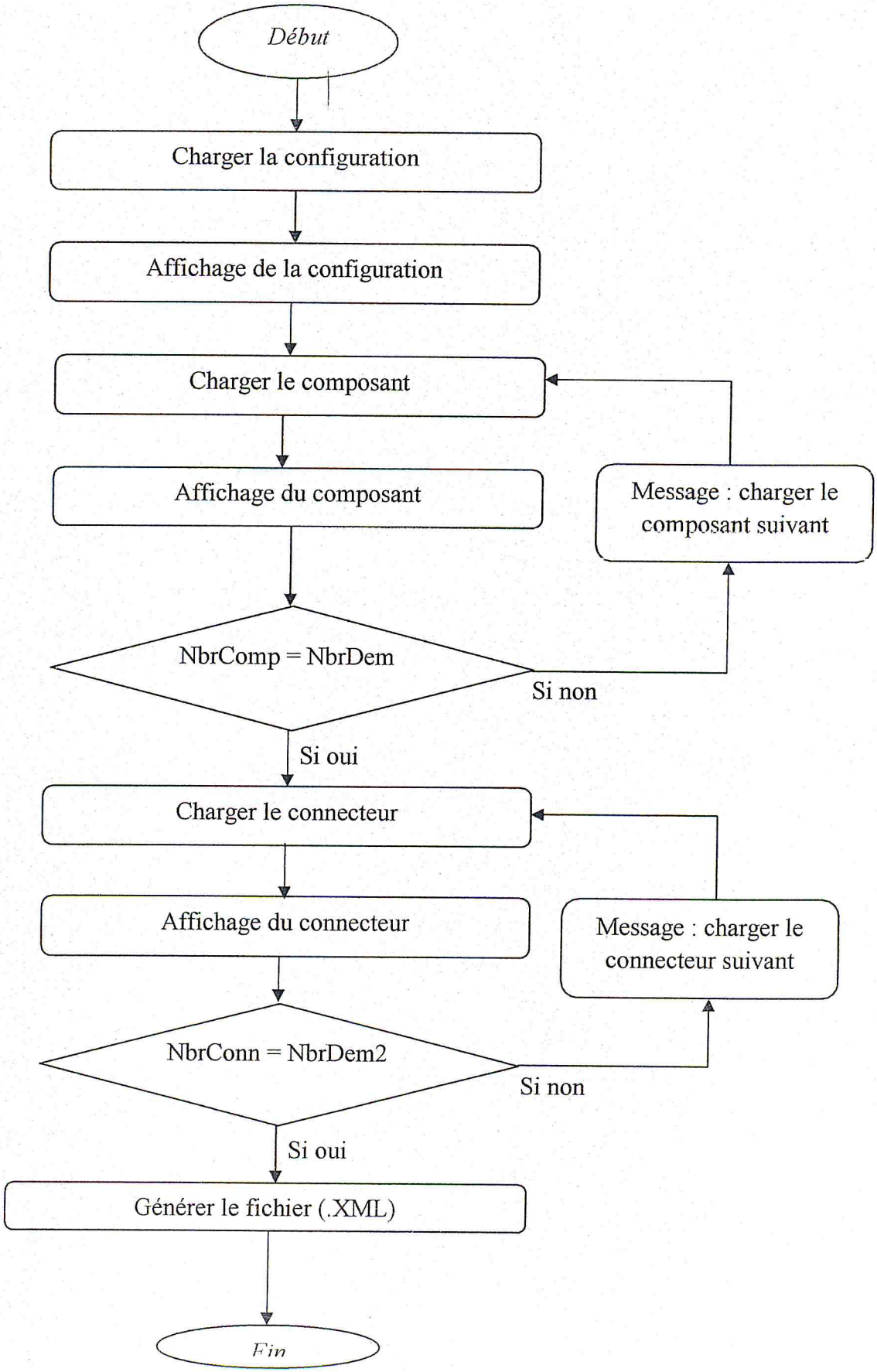


Figure 2 : Schéma à suivre.

Nous devons toujours commencer par faire entrer la configuration en premier dans notre application, et de cette configuration on récupère le nombre de composants et de connecteurs, puis on fait entrer les composants et connecteurs nécessaires à notre déploiement, et enfin on génère notre fichier (.XML).

3.1. Les classes de notre application :

L'objectif de notre travail est la génération d'un fichier (.XML) à partir d'une configuration ACME, afin qu'il soit déployable et ouvrable dans EPF Composer.

L'utilisateur charge la configuration ACME en entrée, et les composants et connecteurs associés à cette configuration sous forme de fichiers XML. Ensuite on appuie sur le bouton « Générer le fichier EPF » (qui est dans notre application) pour obtenir un fichier sous forme XML représentant un enchainement de composants grâce aux connecteurs et à la configuration, comme dernière étape notre fichier (.XML) doit être lu et ouvert par EPF Composer et pour cela on doit respecter sa structure lors du déploiement.

Pour atteindre cet objectif nous avons développé les classes suivantes :

- **Configuration :** elle récupère toutes les données nécessaires pour la structure des composants.
- **Composant :** Il analyse chaque composant et regroupe les données dans des tableaux.
- **Connecteur :** Il analyse chaque connecteur et regroupe les données dans des tableaux.
- **Id:** Elle nous permet de générer des Ids (identifiant) selon la demande de notre application.
- **Main :** Pour lancer l'application.
- **RechConfig :** Elle recherche dans la configuration les liens entre composants et connecteurs pour la classe XlmEditeur.
- **RechElement :** Elle récupère les données des composants et connecteurs pour la classe.
- **XmlEditeur :** Elle regroupe toutes les données pour la génération du fichier (.XML).

4. Exemple sur le déploiement :

Comme exemple nous déployons l'exemple suivant :

Dans cet exemple la configuration à déployer est constituée de deux composants « Conception » et « Realisation » (.xml) et deux connecteurs Précédence et Transmission (.xml) suivant la Configuration (.ACME) voir figure 3.

4.1.Configuration ACME :

```
System System = {

  Component Conception = {
    Port ControlFlow1 = {}
    Port DataOut1 = {}
  }
  Component Realisation = {
    Port ControlFlow2 = {}
    Port DataIn2 = {}
  }
  Connector Transmission = {
```

```
    Role Role1 = {}
    Role Role2 = {}
  }
  Connector Precedence = {
    Role Role3 = {}
    Role Role4 = {}
  }
  Attachment Conception.DataOut1 to Transmission.Role1;
  Attachment Conception.ControlFlow1 to Precedence.Role3;
  Attachment Realisation.DataIn2 to Transmission.Role2;
  Attachment Realisation.ControlFlow2 to Precedence.Role4;
}
```

Le schéma de notre exemple modélisé dans l'environnement acme studio :

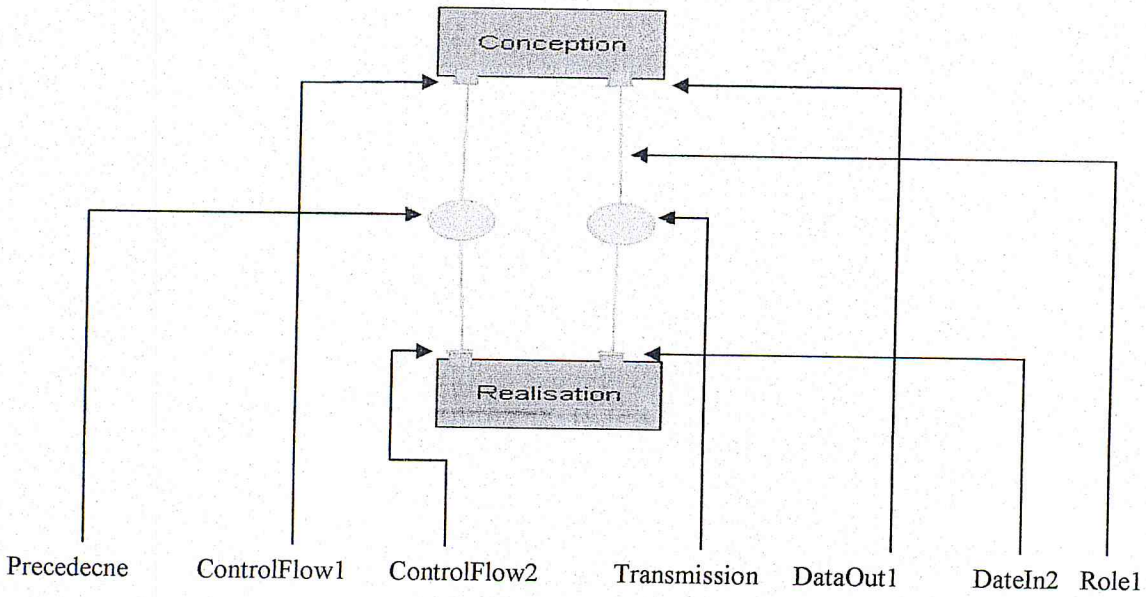


Figure 3: L'exemple a déployer

Nous suivons l'algorithme décrit précédemment en montrant les traitements effectifs à l'aide d'imprimé écran de notre application.

4.2. Chargement de la configuration dans notre application:

Voici les étapes à suivre pour charger la configuration :

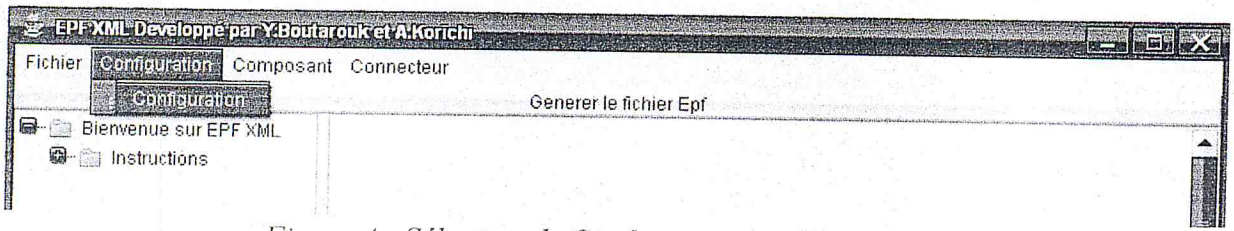


Figure 4 : Sélection de Configuration sur notre application

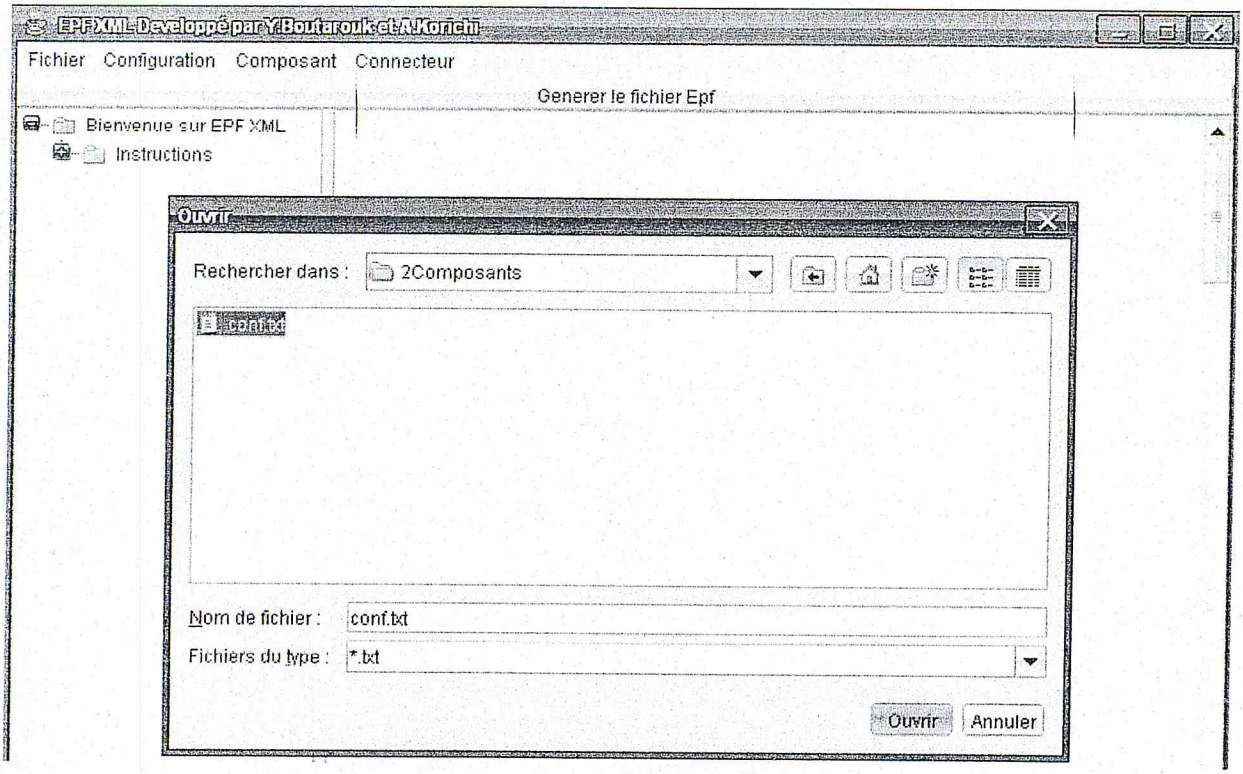


Figure 5 : Récupération de la configuration

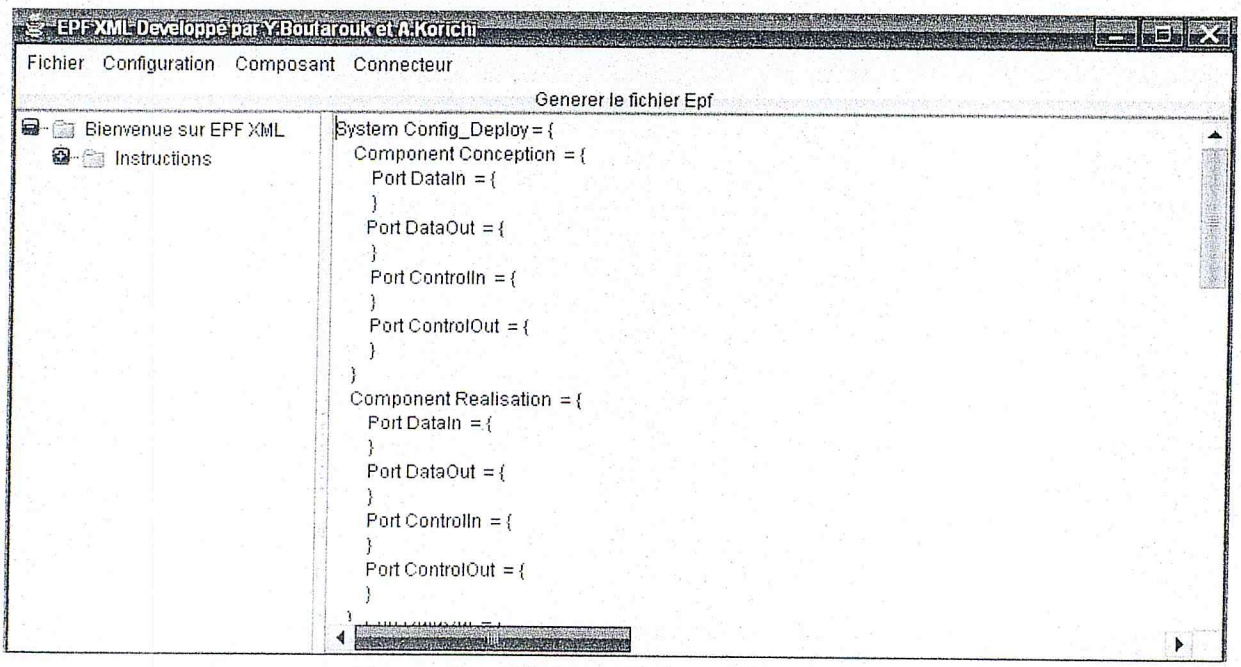


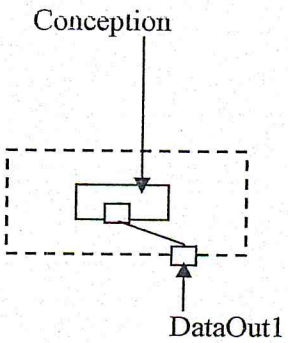
Figure 6 : Affichage de la configuration

Notre programme nous permet de parcourir nos documents pour récupérer notre configuration, et ensuite l'afficher.

4.3. Les composants (.XML) :

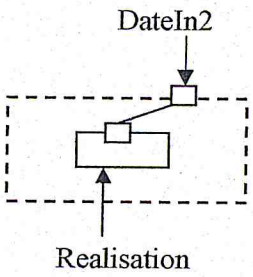
Composant 1 :

```
<FragmentDeProcede>
<Activity>
<Task_Use>Concepetion</Task_Use>
<Process_Performer>M.Boutarouk</Process_Performer>
<Role_Use>R1</Role_Use>
<pro_WorkProductUseOut>DataOut1</pro_WorkProductUseOut>
<successor> Realisation</successor>
</Activity>
</FragmentDeProcede>
```



Composant 2 :

```
<FragmentDeProcede>
<Activity>
<Task_Use>Realisation</Task_Use>
<Process_Performer>M.Korichi</Process_Performer>
<Role_Use>R2</Role_Use>
<pro_WorkProductUseIn>DataIn2</pro_WorkProductUseIn>
<predecessor>Conception </predecessor>
</Activity>
</FragmentDeProcede>
```



4.4. Chargement des composants :

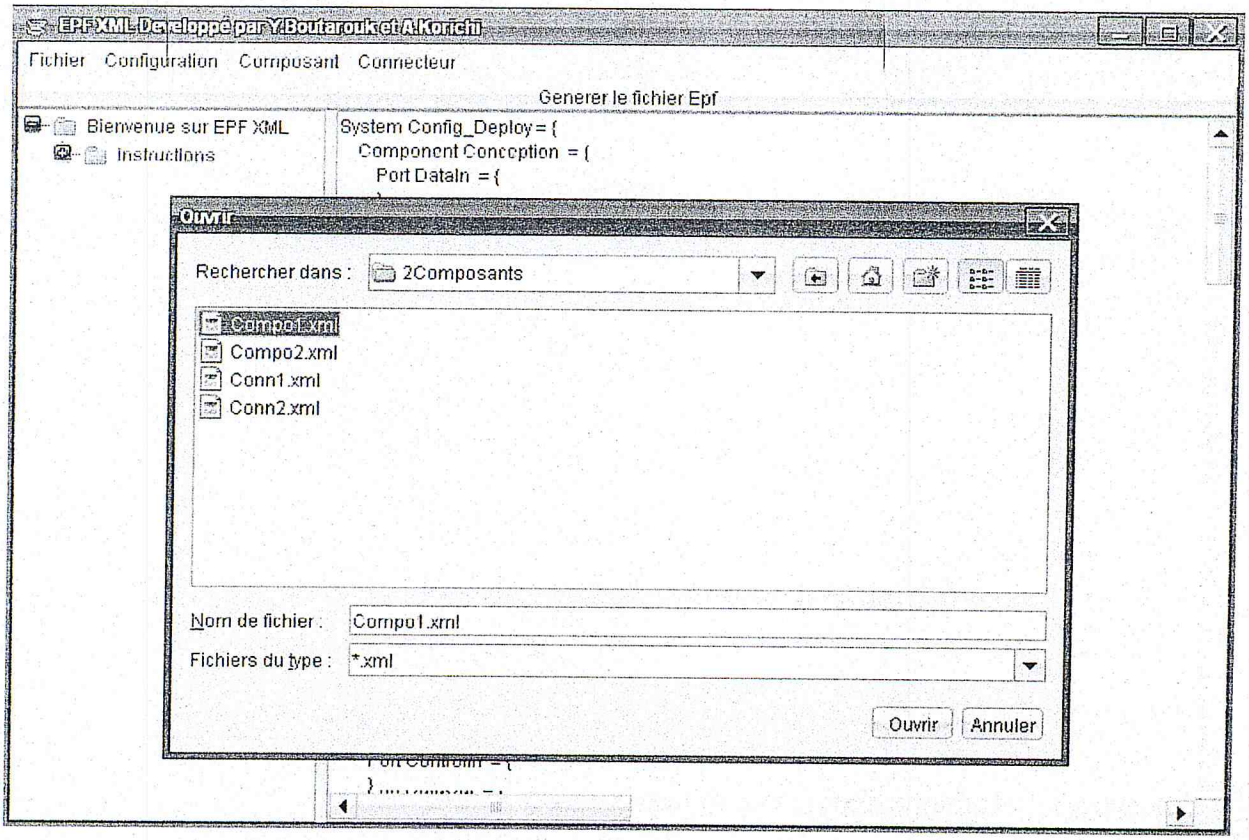


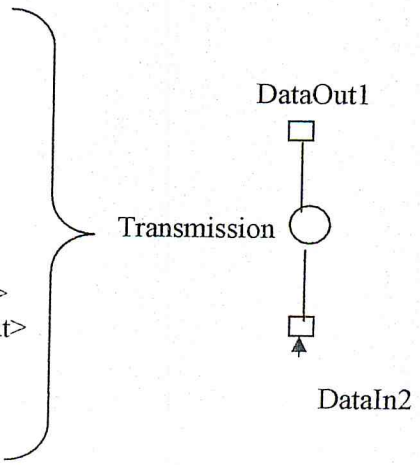
Figure 7 : Chargement des composants sous format XML.

Le chargement des composants se fait de la même manière que la configuration, parfois on a besoin de charger plusieurs composants alors, on refait la procédure.

4.5. Les connecteurs :

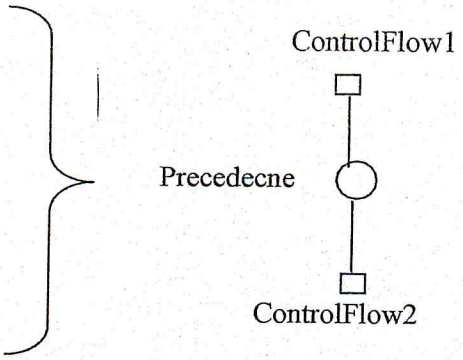
Connecteur Transmission :

```
<FragmentDeProcede>
<Activity>
<Task_Use>Transmission</Task_Use>
<Process_Performer>M.djili</Process_Performer>
<Role_Use>RC1</Role_Use>
<pro_WorkProductUseIn>DataOut1</pro_WorkProductUseIn>
<pro_WorkProductUseOut>DataIn2</pro_WorkProductUseOut>
</Activity>
</FragmentDeProcede>
```



Connecteur Précédence :

```
<FragmentDeProcede>
<Activity>
<Task_Use>Precedence</Task_Use>
<Process_Performer>M.Bassor</Process_Performer>
<Role_Use>RC2</Role_Use>
<predecessor>ControlFlow1</predecessor>
<successor>ControlFlow2</successor>
</Activity>
</FragmentDeProcede>
```



Voici un schéma plus détaillé de notre exemple :

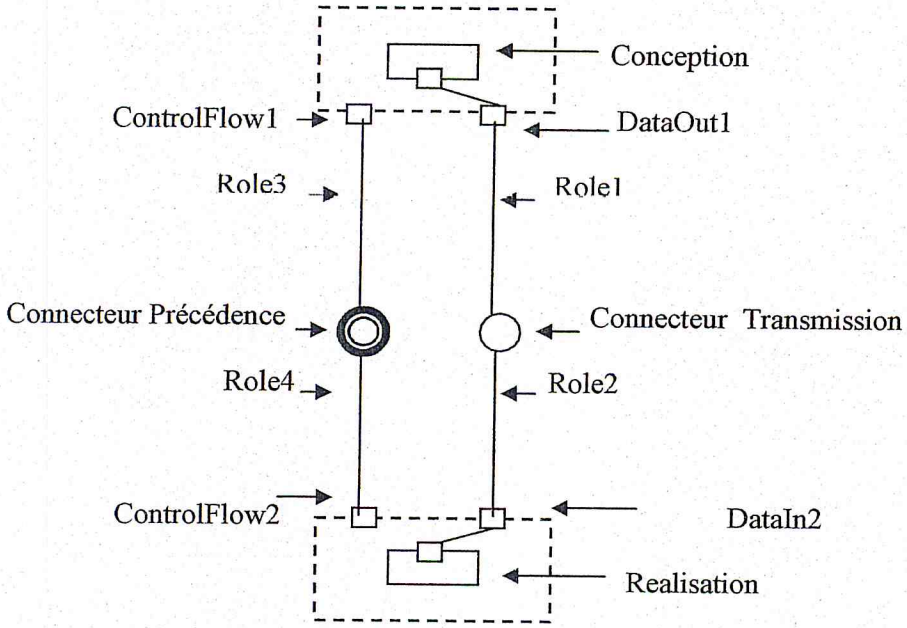


Figure 8 : Schéma plus détail.

Le schéma montre la relation entre connecteurs et composants et les noms des composants.

4.6. Chargement des connecteurs :

Le chargement se fait de la même manière que les composants.

4.7. Les outils utilisés pour la génération du fichier (.XML) :

Pour pouvoir générer le fichier XML, nous allons utiliser certaines fonctionnalités d'Eclipse qui vont nous permettre d'analyser les fichiers composants et connecteurs (.xml) et récupérer les données nécessaires pour notre déploiement et la génération du fichier (.xml).

JDOM (*Document Object Model*) :

JDOM est une API du langage Java développée indépendamment de Sun Microsystems. Elle permet de manipuler des donnés XML plus simplement qu'avec les API classiques.

Elle permet de modéliser, de parcourir et de manipuler un document XML , et pout cela JDOM utilise un parseur SAX . Ce type de parseur utilise des événements pour piloter le traitement d'un fichier XML,[2][3]

JTree : elle va nous permettre de visualiser (Afficher) l'arborescence.
Les données sont affichées ligne par ligne, chaque ligne contenant exactement une donnée qui est un *Nœud.*
Un arbre a une racine.
Les nœuds qui n'ont pas de fils sont des feuilles.[4]

DocumentBuilderFactory : C'est ce constructeur de documents qui va construire notre document XML,il va représenter les données sous forme d'arbre "d'objets".
[http://www.mkyong.com/java/how-to-create-xml-file-in-java-dom/]

Exemple : Nous allons prendre un exemple simple, pour montrer la façon à suivre pour pouvoir générer le fichier (.XML) .
On a le fichier XML suivant :

```
<USDB>
        <Departement id="1">
                <Nom>Boutarouk</Nom>
                <Prenom>Younes</Prenom>
                <Matricule>0906030187</ Matricule >
        </Departement>
</USDB>
```

Notre programme va analyser la structure suivante grâce aux fonctionnalités qu'on a définit précédemment, il va récupérer le « nom » et l'associer à « Boutarouk » , et ainsi de suite jusqu'à la fin du fichier, puis il va regrouper les données suivant une arborescence bien précise.
Notre programme commence par « USDB » comme racine et « Departement » comme son prédécesseur, ensuite il va ajouter le « Nom », « Prenom », « Matricule » comme des prédécesseurs de « Departemment ».
Voici un aperçu de notre programme qui explique ce qu'on vient de définir :

```
DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

        //racine
        Document doc = docBuilder.newDocument();
        Element rootElement = doc.createElement("USDB");
        doc.appendChild(rootElement);

        //Departement
        Element Departement = doc.createElement("Departement");
        rootElement.appendChild(Departement);

        //Identifiant
        Attr attr = doc.createAttribute("id");
```

```
attr.setValue("1");
Departement.setAttributeNode(attr);

//Nom
Element firstname = doc.createElement("Nom");
firstname.appendChild(doc.createTextNode("Boutarouk"));
Departement.appendChild(firstname);


//Prenom
Element lastname = doc.createElement("Prenom");
lastname.appendChild(doc.createTextNode("Younes"));
Departement.appendChild(lastname);


//Matricule
Element matricule = doc.createElement("Matricule");
matricule.appendChild(doc.createTextNode("0906030187"));
Departement.appendChild(matricule);

// écrire le contenu dans un fichier xml
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(doc);
StreamResult result =  new StreamResult(new File("C:\\NotreFichier.xml"));
```

Pour que ce code respecte la structure de « EPF Composer » on doit toujours commencer par la structure suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<uma:MethodLibrary xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:uma="http://www.eclipse.org/epf/uma/1.0.6" name="library2" briefDescription=""
id="_dK-9sGEqEeCKPJCz43JaLQ" orderingGuide="" presentationName=""
suppressed="false" authors="" changeDescription="" version="" tool="epf=1.5.0">
```

Remarque : On trouve ces structures en exportant des projets d'EPF Composer on XML.

4.8. Génération du fichier (.XML) à partir de notre application :

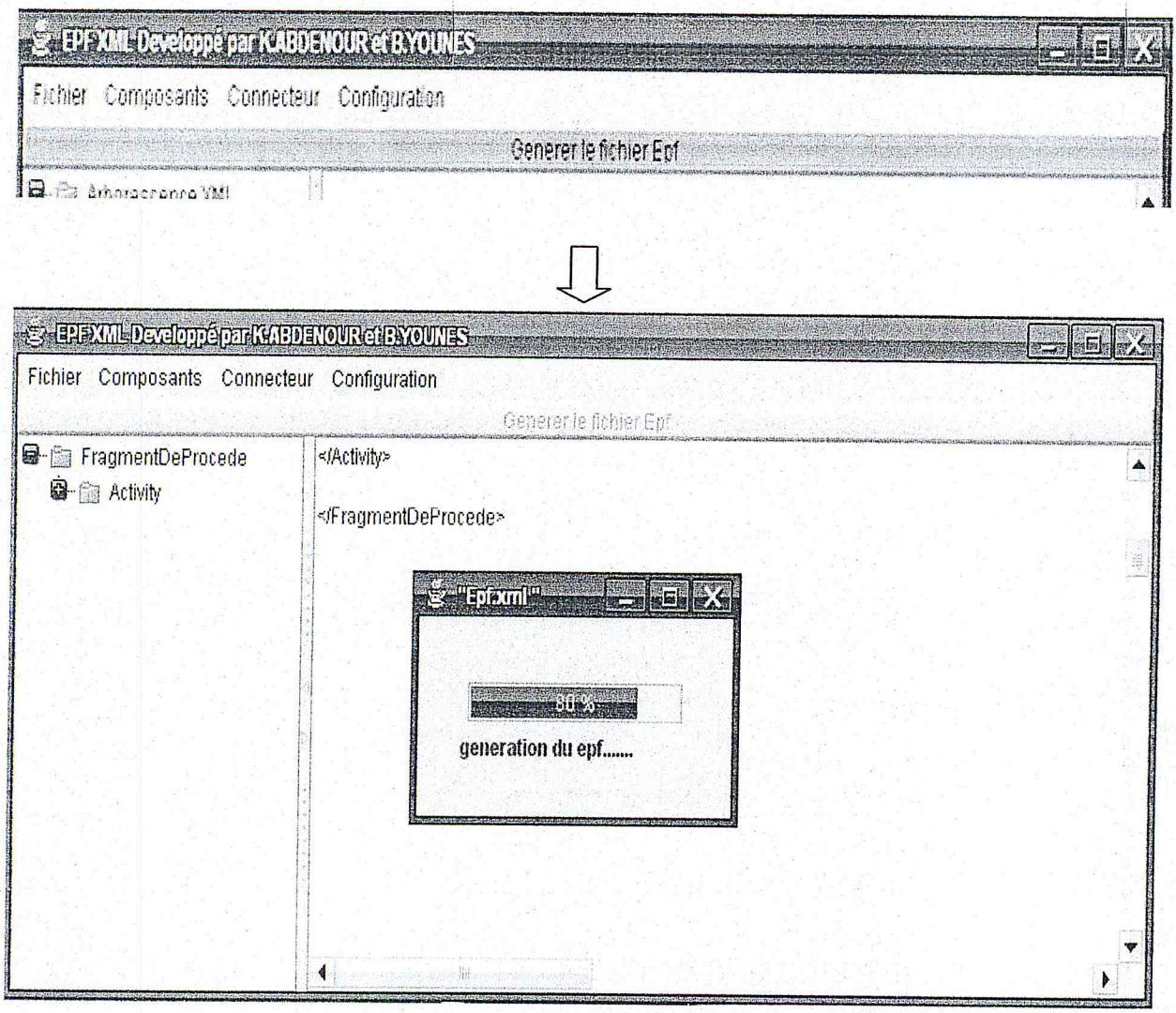


Figure 9 : La génération du fichier (.XML).

Après la génération de notre fichier (.xml) on le trouve dans la partie « c » de notre Windows (c'est nous qui avions choisi ce chemin).

Après l'ouverture de notre fichier on trouvera, tout ce qu'on a chargé comme données, exemple « M.Boutarouk .... », va être structuré dans des balises qu'on a implémenté dans notre programme, et ce fichier respectera la structure EPF Composer.

Le fichier (.xml) généré a une taille de 15 ko (la taille du fichier augmente à chaque fois qu'on ajoute des composants et connecteurs), soit 151 lignes.

Voici le fichier généré :

Figure 10 : Le fichier (.XML) après génération.

Remarque : ceci n'est qu'une partie du fichier.

4.9 Notre fichier (.XML) dans EPF Composer :

Après ouverture de notre fichier (.xml) avec EPF Composer.
Voici des aperçus de ce qu'on voit sur EPF Composer.

1-

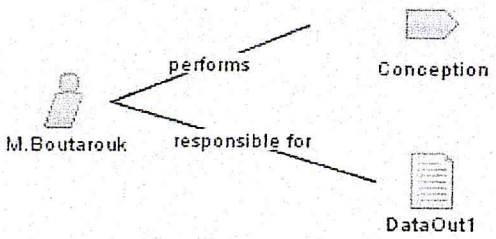


Figure 11 : Preview du rôle EPF Composer

Explication : on voit ci dessus que « M.Boutarouk » effectue la tâche « Conception » et est responsable de l'artefact « DataOut1 ».

On a pu faire ces liens grâce aux Ids (identifiants) de Epf composer qui nous permettent d'affecter des tâches et responsabilités (et pour cela on doit garder le même Id), et d'ajouter la balise <PerformedBy> dans le rôle, en lui donnant le même Id de la tâche « Conception ».

2-

Presentation Name	In...	Predecessors
Systeme_comp	0	
Conception	1	
Realisation	2	1,3
Transmission	3	1

Figure 12 : Process_comp (EPF Composer)

Explication : Dans la figure 12 on voit le rôle joué par les connecteurs.

La tâche « Conception » n'a pas de prédécesseurs alors, elle s'exécute en premier, ensuite elle envoie le signal à « Transmission » puis à « Realisation » via le connecteur « Precedence ». La tâche « Realisation » s'exécute en dernier après avoir reçu le signal de « Conception » et « Transmission ».

Cela se fait grâce à la balise <Predecessor> en ajoutant le Id du prédécesseur.

3-

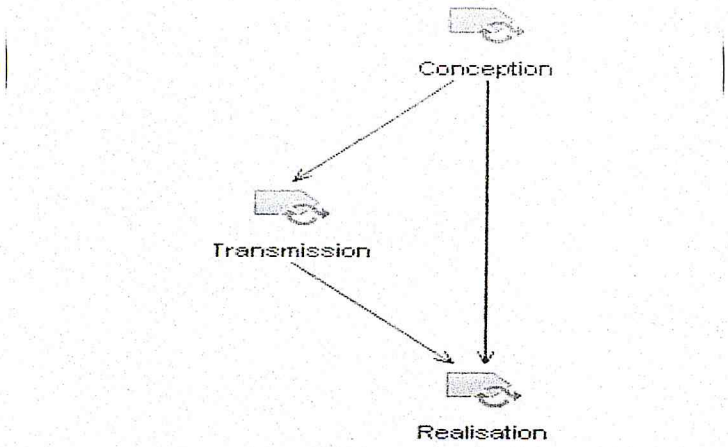


Figure 13 : Delivery Process(EPF Composer)

La figure 13 représente d'une manière plus simple ce qu'on a expliqué dans la figure 12.

4-

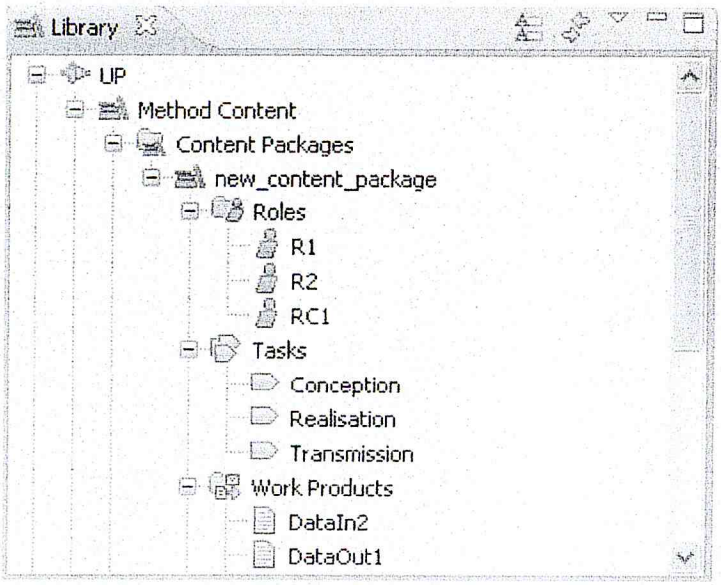


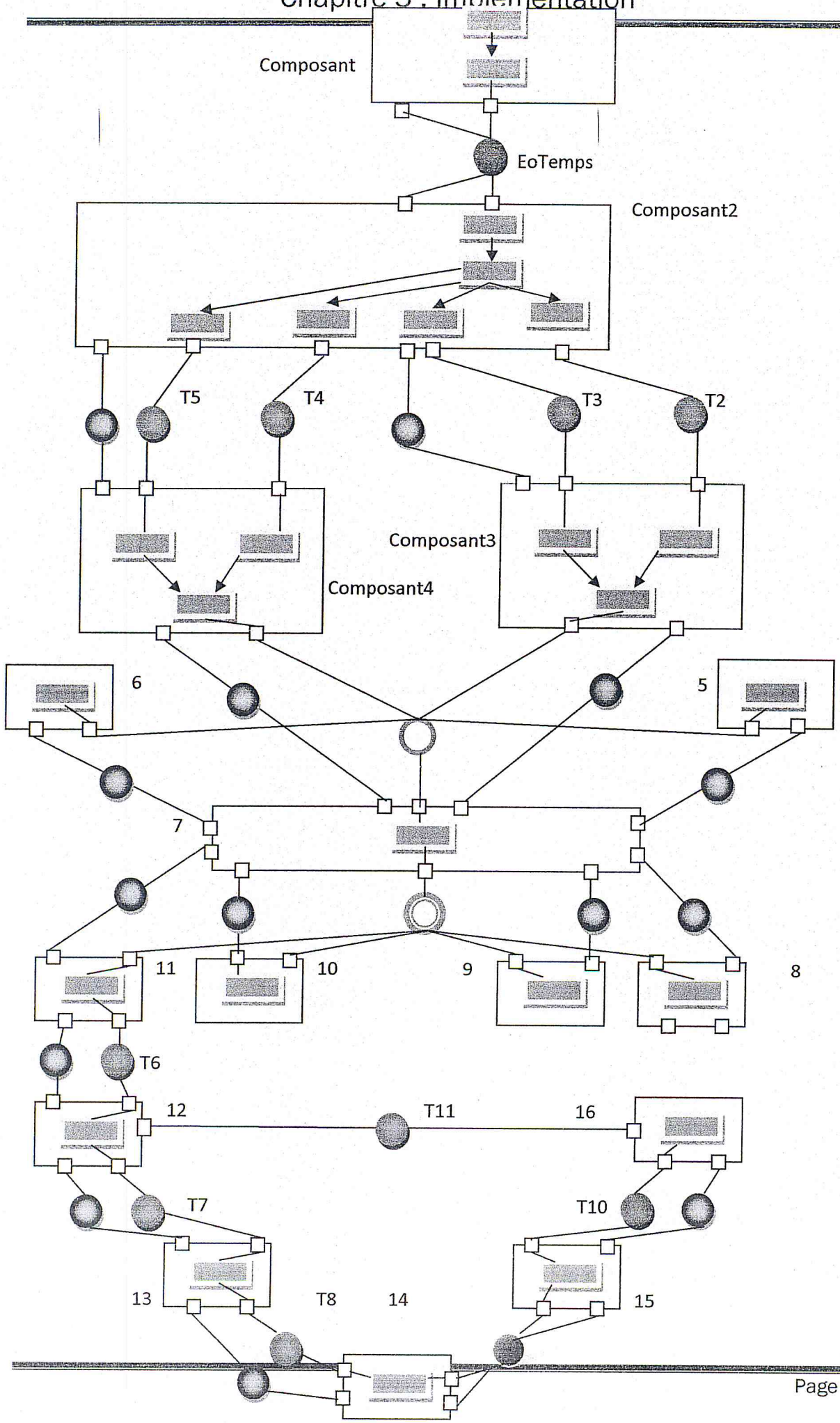
Figure 14 ; Library content (EPF Composer)

Dans Library on trouve tous les rôles et tâches et artefacts de nos fichiers, par exemple le premier composant « Conception » on le trouve dans « tasks » et son rôle R1 dans « Roles » et ses deux artefacts dans « Work Products ».

5. Exemple qui comporte l'ensemble des composants et connecteurs étudiés :

Nous prenons un exemple regroupant un ensemble de composants et connecteurs différents qu'on a vu précédemment.

L'exemple regroupe 16 composants et 27 connecteurs (9 Transmissions, 14 Précédences, 1 fusion ,1 diffusion ,1 EoTemps, 1 EoQuality),


Composant
EoTemps
Composant2
T5
T4
T3
T2
Composant3
Composant4
6
5
7
11
10
9
8
T6
12
T11
16
T7
T10
13
T8
14
15

Composant

Activity

Connecteur Precedence

Connecteur Transmission

Connecteur Fusion

Connecteur Diffusion

Connecteur EoTemps

Connecteur EoQuality

Figure 15 : Exemple général.

Après déploiement (exécuté par notre application)

Le résultat est un modèle de PL utilisable sous EPF. EPF Composer va nous donner le tableau suivant qui montre les enchaînements :

Presentation Name	In...	Predecessors
Composant	1	
Composant(2)	2	1
Composant2	3	27
Composant2(2)	4	3
Composant2(3)	5	4
Composant2(4)	6	4
Composant2(5)	7	4
Composant2(6)	8	4
Composant3	9	5,6,7,8,28
Composant3(2)	10	5,6,7,8,29
Composant3(3)	11	9,10
Composant4	12	5,6,7,8,30
Composant4(2)	13	5,6,7,8,31
Composant4(3)	14	12,13
Composant5	15	
Composant6	16	
Composant7	17	11,14,15,16,32
Composant8	18	17,33
Composant9	19	17,33

Presentation Name	In...	Predecessors
Composant10	20	17,33
Composant11	21	17,33
Composant12	22	21,34
Composant13	23	22,35
Composant14	24	23,36
Composant15	25	37
Composant16	26	25,38,39
EoTemps	27	2
Transmission2	28	5
Transmission3	29	6
Transmission4	30	7
Transmission5	31	8
fusion	32	11,14,15,16
diffusion	33	17
Transmission6	34	21
Transmission7	35	22
Transmission8	36	23
EoQuality	37	24
Transmission10	38	25
Transmission11	39	22

Figure 16 : tableaux des prédécesseurs (image prise de « EPF Composer »).

Dans le tableau on remarque que « Composant » qui est à l'index « 1 » n'a pas de prédécesseur alors il va s'exécuter en premier (la même chose pour « composant5 » et « Composant6 ») , ensuite il va déclencher le « composant(2) » qui a comme prédécesseur l'index « 1 » qui est « Composant » et ainsi de suite ,jusqu'au dernier composant.

6. Le diagramme du composant :

Le diagramme du composant permet de décrire l'architecture physique et statique du système en termes de modules (fichier source exécutable, librairie, .etc). Les dépendances entre composants permettent d'identifier les contraintes de compilation ou d'exécution et mettent en évidence les contraintes de réutilisation des composants.

Le diagramme du composant suivant montre l'architecture physique de notre application.

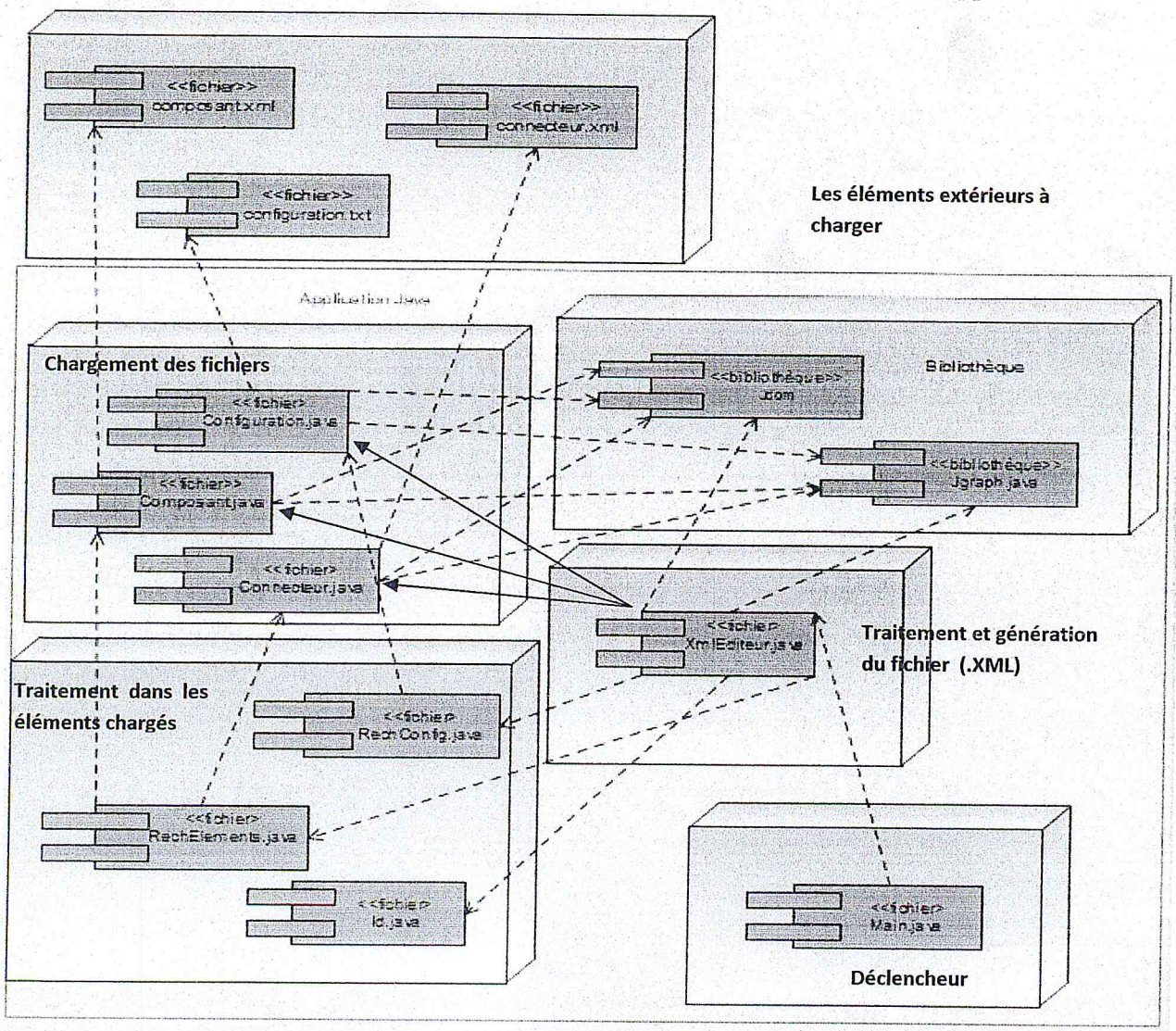


Figure 17 : Diagramme des composants.

Le « Main » déclenche notre application, ouvre « XmlEdireur » ce dernier va charger les fichiers (configuration, composants, connecteurs) via le « Chargement des fichiers », et commence à traiter les éléments tout en utilisant les bibliothèques et les classes de traitement.

7. Conclusion :

Nous avons présenté dans ce chapitre l'application que nous avons développée ainsi que les outils et les langages de programmation utilisés. Toutes les fonctionnalités de l'application ont été décrites, avec à chaque étape une illustration de notre interface.

Conclusion générale

Ce travail nous a permis d'élargir nos connaissances dans les domaines de génie logiciel.

En effet, nous avons travaillé sur les procédés logiciels, notre objectif principal était la réutilisation des PLs. Cette réutilisation consiste à extraire puis déployer des architectures de PLs.

Et pour cela, nous avons implémenté une partie de cette approche. Notre travail consistait à déployer des architectures de PL, en exploitant des connaissances existantes préparées pour le déploiement.

Le déploiement a été fait en exploitant des configurations de PL décrites en langage de description d'architecture logiciel ACME, ainsi que des fichiers XML décrivant les composants et connecteurs de PLs. Le résultat était en « EPF Composer ».

Nous avons choisi « EPF Composer » parce qu'il respecte le méta modèle SPEM, qui est un standard défini par l'OMG. Aussi, « EPF Composer » est gratuit et accessible à tout le monde.

Ainsi, notre application permet de réutiliser des modèles de procédés logiciels respectant le méta-modèle SPEM, nous déployons des architectures de PLs.

Les perspectives futures de recherche seraient de permettre différents types de déploiement. Nous avons pu jusque là, implémenter le déploiement sélectif qui se fait partiellement en sélectionnant les composants à déployer.
Cette possibilité nous permettra dans le futur de définir d'autres types de déploiement qui exprimeront les caractéristiques des PLs tel que le déploiement itératif, incrémental ou distribué.

Bibliographie

Introduction générale :

[1] Aoussat, F., M. Ahmed-Nacer, et M. Oussalah (2010). Reusing approach for software Processes based on software architectures. In ICEIS.

Chapitre 1 :Procédés logiciels et architectures logicielles.

[1] Sonia, Jamel, Environnement de procédé extensible pour l'orchestration Application aux services web, Thèse de Doctorat, Université Joseph Fourier de Grenoble ,Décembre 2005.
[2] Ma. Lizbeth ,Gallardo, Une approche à base de composants pour la modélisation des procédés logiciels,DEA, préparé dans l'équipe ADELE, au laboratoire LSR, Septembre 2000.
[3] Sergio, Garcia-Camargo, Ingénierie Concurrente en Génie Logiciel : Céline, Thèse de doctorat de l'université Joseph Fourier de Grenoble, Décembre 2006.
[4] Amiour, Mahfoud, Vers une fédération de composants interopérables pour les environnements centrés procédés logiciels, Thèse de doctorat de l'université Joseph Fourier, Grenoble I, juin 1999.
[5] Hanh ,Nhi Tran, Modélisation de procédés logiciels à base de patrons réutilisables, Thèse de d octorat de l'université Toulouse II- Le Mirail, Novembre 2007.
[6] SPEM-OMG (2008). Software Systems Process Engineering Metamodel, v2.0, Object Management Group OMG.
[7] Mehta, N. R., N. Medvidovic, et S. Phadke (2000). Towards a taxonomy of software connectors. In ICSE '00: Proceedings of the 22nd international conference on Software engineering, New York, NY, USA, pp. 178–187. ACM.
[8] Medvidovic, N. et R. N. Taylor (1997). A framework for classifying and comparing architecture description languages. In ESEC / SIGSOFT FSE, pp. 60–76.
[9] Aoussat, F., M. Ahmed-Nacer, et M. Oussalah (2010). Reusing approach for software processes based on software architectures. In ICEIS, pp. 366–369.
[10] Gary, K., T. Lindquist, H. Koehnemann, et J. Derniame (1998). Component-based software process support. Automated Software Engineering, International Conference on 0, 196.
[11]Belkhatir, N. et J. Estublier (1996). Supporting reuse and configuration for large scale software process models. In ISPW '96: Proceedings of the 10th International Software Process Workshop, Washington, DC, USA, pp. 35. IEEE Computer Society.
[12] Avrilionis, D., N. Belkhatir, et P. Y. Cunin (1996). A unified framework for software process enactment and improvement. In ICSP '96: Proceedings of the Fourth International Conference on the Software Process (ICSP '96), Washington, DC, USA, pp. 102. IEEE Computer Society.
[13] Medvidovic, N., P. Grünbacher, A. Egyed, et B. W. Boehm (2003). Bridging models across the software lifecycle. J. Syst. Softw. 68, 199–215.
[14] Choi, J. et W. Scacchi (2000). Modeling and simulating software acquisition process architectures. Journal of Systems and Software 59, 343–354.
[15] Dami, S., J. Estublier, et M. Amiour (1998). Apel: A graphical yet executable Formalism forprocess modeling. Automated Software Engg. 5, 61–96.
[16] Coulette, B., T. D. Thu, X. Crégut, et B. T. Dong Thi (2000). Rhodes, a process component centered software engineering environment. In ICEIS, pp. 253–260.

[17]Dai, F., T. Li, N. Zhao, Y. Yu, et B. Huang (2008). Evolution process component composition based on process architecture. In IITAW '08: Proceedings of the 2008 International Symposium on Intelligent Information Technology Application Workshops,Washington, DC, USA, pp. 1097–1100. IEEE Computer Society.

[18] Borsoi, B. T. et J. L. R. Becerra (2008). A method to define an object oriented software process architecture. Software Engineering Conference, Australian 0, 650–655.

[19]Boehm, B. et S. Wolf (1996). An open architecture for software process asset reuse. In ISPW '96: Proceedings of the 10th International Software Process Workshop, Washington, DC, USA, pp. 2. IEEE Computer Society.

[20] Sunghwan, R., K. Kyungrae, et J. Taewoong (2004). Architecture modeling language based on uml2.0. In APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference, Washington, DC, USA, pp. 663–669. IEEE Computer Society.

[21]Zarras, A., V. Issarny, C. Kloukinas, et N. V. K. (2001). Towards a base uml profile for architecture description. In Proceedings of the ICSE Workshop on Architecture and UML, pp. 22–26. IEEE Computer Society.

[22] Alti, A., A. Boukerram, A. Smeda, S. Maillard, et M. Oussalah (2010). Cosabuilder and cosainstantiator : An extensible tool for architectural description. International Journal of Software Engineering and Knowledge Engineering 20(3), 423–455.

[23]Hudaib, A. et C. Montangero (2002). A uml profile to support the formal presentation of software architecture. In COMPSAC, pp. 217–223.

[24]Alti, A., A. Boukerram, A. Smeda, S. Maillard, et M. Oussalah (2010). Cosabuilder and cosainstantiator : An extensible tool for architectural description. International Journal of Software Engineering and Knowledge Engineering 20(3), 423–455.

[25] Métamodélisation architecturale des procédés logiciels Fadila Aoussat, Mourad Oussalah, Mohamed Ahmed-Nacer

[26] Architecture logicielle et conception avancée Partie ½ Yann-Gaël Guéhéneuc 2009

[2 7] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Addison-Wesley, 1998.

[28] [ÉCOLE DOCTORALE STIM « SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATÉRIAUX » Extraction d'une architecture logicielle à base de composants depuis un système orienté objet Présentée et soutenue publiquement par Sylvain CHARDIGNY] Année 2010

[29]Une approche pour les architectures logicielles à composants multimédia Makhlouf Derdour, Philippe Roose ,Marc Dalmau , Nacéra Ghoualmi Zine , Adel Alti Université d'Annaba.

[30]M E M O I R E Pour obtenir le diplôme de : Magister en Informatique Expression et Vérification des Contraintes non Fonctionnelles d'une Architecture SADL de Mr LATRECHE FATEH

[31] Dewayne E. PERRY et Alexander L. WOLF. Foundations for the study of software architecture. SIGSOFT Softw. Eng. Notes, 17(4) :40–52, 1992.

[32] D. GARLAN et D.E. PERRY. Introduction to the special issue on software architecture. IEEE Transactions on Software Engineering, 21(4) :269–274, 1995.

[33]David GARLAN. Software architecture : a roadmap. In ICSE '00 : Proceedings of the Conference on The Future of Software Engineering, pages 91– 101, New

York, NY, USA, 2000. ACM.
[34] Plate-forme de composants logiciels pour la coordination des adaptations multiples en environnement dynamique Djalel CHEFROUR INRIA Rennes (2001-2004) Projet RNRT Cyberté
[35] David GARLAN et Mary SHAW. An introduction to software architecture. Rapport technique, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.
[36] Nenad Medvidovic. « Mae – A System Model and Envoronment for Managing Architectural Evolution » ACM Transactions on Software Engineering and Methodology. Accepted for publication, 2004.
[37] Rapide Design Team Program ANALYSIS et Verification GROUP. Guide to the rapide 1.0 language reference manuals, Rapport technique, Computer Systems Lab Stanford University, Stanford, USA, july 1997.
[38]Extraction d'une architecture logicielle à base de composants depuis un système orienté objet THÈSE DE DOCTORAT Sylvain CHARDIGNY 23 octobre 2009
[39][http://www.omg.org/spec/SPEM/2.0/]

Chapitre 2 : EPF Composer.

[1] http://www.eclipse.org/projects
[2] www.mountainview.ca
[3] Eclipse Process Framework (EPF) Composer (Installation, Introduction, Tutorial and Manual) 8-Mars-10 http://epf.eclipse.org/uploads/14.pdf
[4] Eclipse Process Framework Composer Part 1: Key Concepts de Peter Haumer, Avril 2007.
[5] Eclipse Process Framework Composer: Part 2: Authoring method content and processes Avril 2007 de Peter Haumer.
[6] Ivar Jacobson et al, "The Unified Software Development Process", Addison Wesley, 1998.
[7] Présentation de la méthode OpenUp 1.0 Livre Blanc 19/10/2007 par Xavier MEHAUT
[8] www.eclipse.org/epf/

Chapitre 3 : Principe de déploiement d'architecture de procédé logiciel.

[1] www.cs.cmu.edu/~acme/AcmeStudio/index.html
[2] Métamodélisation architecturale des procédés logiciels Fadila Aoussat_, Mourad Oussalah_, Mohamed Ahmed-Nacer_Département d'informatique, Université Saad Dahlab Blida, (2008)

Chapitre 4 : Conception.

[1] Antoine Clave, article sur UML et la modélisation, Consultant indépendant en gestion de projet et en modélisation fonctionnelle, La Lettre d'ADELI n°53 – Octobre 2003.
[2] Anne-Marie Hugues, Génie logiciel, DIFFERENTS MODELES DE CYCLE DE VIE, 19/12/02.
[3] Pierre-Alain Muller, Modélisation objet avec UML,Eyrolles,1997