UNIVERSITE SAAD DAHLAB DE BLIDA
FACULTE DES SCIENCES
DEPARTEMENT D'INFORMATIQUE

Mémoire en vue d'obtention du diplôme de Master
Spécialité : ingénierie du logiciel

Etude et implémentation d'une ligne de produit pour l'e-Formation

Présentée par :

RAMDANI Abdelouahab

MAZARI Hamza

Promoteur :
M. SIDOUMOU Mohamed Redha

Encadreur :
Mlle. GUENDOUZ Amina

2012/2013

Remerciements

On tient à remercier Allah le tout puissant de nous avoir donné la force, le courage, la volonté, la patience grâce à ses bienfaits on a pu réaliser ce projet.

On tient à exprimer notre profonde gratitude et nos respectueux remerciements à Mr Sidoumou Mohamed Redha notre promoteur, on le remercie de nous avoir guidé, orienté et aidé durant notre recherche.

On tient à remercier Melle Guendouz Amina, notre encadreur qui nous a patiemment guidés dans l'élaboration de ce travail et pour les informations très utiles qu'elle les a mises à notre disposition.

Nos vifs remerciements à toutes les personnes qui ont bien voulues nous aidés de près ou de loin dans notre projet de fin d'études.

Dédicaces

Je dédie ce modeste travail à mes parents qui m'ont mis au monde Grâce à eux et à ses encouragements je suis arrivé à ce stade.

A ma chère mère, la femme la plus adorable dans ma vie.

A mon cher meilleur père.

A mes chers frères et sœurs, je les souhaite tout le bonheur et la prospérité dans toute leur vie.

A celle qui m'a aidée et soutenue durant mes études ma meilleure amie Nassima Belkhir.

A tous mes amis et à ceux qui me connaissent.

Hamza Mazari

REMERCIEMENTS

Avant tout, je tien a remercie Allah en tout-puissant qui m'a aidées à réaliser ce projet.

Je remercie particulièrement ma famille pour le soutien qu'elle m'a apporté et la motivation qu'elle a su me donner lorsque j'en avais le plus besoin.

Je tiens à remercier mon promoteur Mr Mohamed Redha Sidoumou, de nous avoir fait confiance et de nous avoir donné le courage et la force pour vaincre toutes les difficultés.

Je remercie sincèrement notre encadreur Mlle guendouz Amina pour son aide, sa patience et son attention durant cette année, je la remercie pour les informations très utiles qu'elle les a mis à notre disposition. et je souhaite pour elle tout le bonheur et la prospérité.

Je remercie également mes camarades Zinou Houcine Fateh Nacer Salah Hamza et Yucef pour les jours inoubliables que nous avons eues récemment.

Enfin, Je remercie tous ceux qui ont contribué de près ou de loin à notre formation intellectuelle.

RAMDANI Abdelouahab

Résumé

Pour les développeurs des systèmes informatiques d'aujourd'hui la complexité est une problématique importante à gérer, cependant tout le cycle de développement logiciel. Des approches comme les lignes de produits logiciels proposent des solutions aux problèmes de gestion de la complexité, diminution des coûts et amélioration de la qualité du logiciel final en basant sur la réutilisation. La réutilisation s'impose de plus en plus comme une solution méthodologique et technologique pour le développement de systèmes logiciels complexes. Un des principaux défis de la communauté de recherche et des industries et de trouver les concepts, mécanismes et langages adéquats pour une meilleure réutilisation et configuration en masse des systèmes logiciels. Par ailleurs, les lignes de produits logiciels est un paradigme qui prône une vision de modélisation et de développement dans laquelle l'objectif envisagé n'est plus l'obtention d'un système logiciel, mais d'un ensemble de systèmes logiciels possédant des caractéristiques communes.

L'objectif de notre projet de fin d'étude consiste sur le développement d'une ligne de produit pour l'e-formation et la détermination des composants réutilisables. L'architecture est modélisée suivant la discipline orientée composant dans le but est de renforcer la réutilisation. L'approche orientée composant utilisée est IASA cette approche décrit les composants et leurs interconnexions.

Mots Clés

Ligne de Produit (LdP) logiciel, e-formation, Composant, IASA, Réutilisation.

Table des matières

Partie 2 : Conception et réalisation d'une ligne de produit pour le e-formation

Liste des Figures

Introduction Générale

1. Généralités

Actuellement, le domaine du génie logiciel a évolué pour couvrir les nouvelles problématiques introduites par l'extension de la pénétration de l'informatique dans l'industrie. L'e-formation ou formation en ligne suivant plusieurs définitions, désigne de façon globale l'usage des technologies d'information et de communication pour l'apprentissage ; ce terme regroupe à la fois e-éducation, e-Learning, e-training. L'e-formation est une technique de formation reposant sur la mise à disposition de contenu pédagogique via un support électronique et un système de formation ultra-flexible, le stagiaire peut l'apprendre d'où il souhaite et quand il le désire. Il apprend sans se déplacer. Le suivi de la formation est facile et accessible en temps réel grâce aux outils de gestion.

Avec l'extension de l'utilisation des systèmes de E-Learning, nous sommes arrivés à une situation où la problématique n'est plus de développer un seul logiciel à la fois mais plutôt concevoir et développer un ensemble de logiciels destinés à un domaine particulier (e-Formation).

La mise en place d'une ligne de produit pour l'e-Formation va nous permettre de produire des applications pour chaque institut de formation en un temps très cours.

Une ligne de produits logiciels est un ensemble de produits qui :

- partageant un ensemble de fonctionnalités communes.
- satisfaisant des besoins spécifiques pour un domaine particulier.
- développés de manière contrôlée à partir d'un ensemble commun d'éléments réutilisables.

Une ligne de produits a pour but la mise en commun des travaux de développement, de tests et de maintenance de ces éléments logiciels communs de façon à réduire les coûts de production et de maintenance, réduire le temps de production et améliorer la qualité.

2. Présentation du sujet

Notre travail consiste à réaliser les diverses études en vue de mettre sur pied une ligne de produits pour les logiciels de gestion de la formation en ligne, appelée aussi e-formation afin de produire des applications pour chaque institut de formation en un temps très cours.

2.1. Problématique

Les logiciels de gestion de formation dans les diverses institutions publiques ou privés sont destinés aux différents niveaux primaire, moyen, secondaire et universitaire se caractérisent par plusieurs similarités et se distinguent aussi par des aspects particuliers.

Au temps où l'internet joue un rôle très important dans notre vie, on trouve que rare sont les systèmes d'e-formation fonctionnels ou accessibles via internet.

Il devient alors très important de pouvoir profiter des similarités entre ces systèmes, pour développer à la fois un ensemble de systèmes d'e-Formation efficaces.

D'autre part, l'utilisation d'une approche orientée objet pour l'implémentation des applications risque de limiter la possibilité de réutilisation, l'introduction d'autres paradigmes et approches et ainsi devenue nécessaire. Ce qui pose un autre problème qui doit être étudié.

2.2. Objectifs

L'objectif de notre travail consiste à concevoir et développer une ligne de produit qui permet de :

- Optimiser le temps, coût et de minimiser l'effort de développement.
- Identifier les similarités et variabilités entre les applications de l'e-formation.
- Développer les core asset du domaine (élément réutilisable) :
    - Les diagrammes de modélisation du domaine.
    - Le composant réutilisable.
- Développer des applications e-formation en se basant sur les core assets.

- Augmenter la possibilité de réutilisation des éléments logiciels on se basant sur une approche orientée composant.

- Développer des composants spécifiques pour les applications finales s'il existe.

3. Organisation du mémoire

Ce mémoire contient deux parties :

La première partie de ce mémoire comporte trois chapitres.

Le premier chapitre est consacré à une étude sur les Lignes De Produit logiciel (LdP) dans lequel nous citons tout d'abord, quelques définitions et objectifs des LdPs. Puis, nous présentons en détail le processus de développement des LdPs. Ensuite, nous introduisons une notion très importante dans le domaine des LdPs qui est la variabilité, enfin nous présentons les perspectives et défis qui rencontrent le développement des LdPs.

Le deuxième chapitre présente une vue en détail sur les concepts de l'orienté composant, les différents modèles à base de composant, les langages de description d'architecture (ADL), enfin les principaux concepts de l'orienté aspect.

Le troisième chapitre est consacré à une étude du domaine e-formation, considéré comme domaine d'application dans notre projet de fin d'étude. Dans cette partie nous présentons en détails les dispositifs de formation en ligne, les acteurs impliqués dans chaque étape de formation et leurs différents échanges.

La deuxième partie de ce mémoire comporte deux chapitres

Dans le premier chapitre, nous entamons le processus de développement de notre ligne de produit en commençant par l'ingénierie du domaine pour notre domaine d'étude qui est l'e-formation, les résultats de ce processus seront réutilisés lors de l'ingénierie d'application.

Le deuxième chapitre est dédié au processus d'ingénierie d'une application d'e-formation, c'est le deuxième processus dans le développement des lignes de produits dont la tâche principale est la dérivation d'une application d'e-formation et

le développement de cette application en réutilisant les résultats du premier processus.

Partie 1 :

Concepts de base

Chapitre 1 : les lignes de produit logiciel

1. Introduction

À la fin du XIXe siècle, Eli Whitney et Henry Ford ont révolutionné la *production industrielle*. Cette révolution se basait notamment sur un changement radical des méthodes de travail. La production de masse est née avec l'automatisation des processus favorisant l'assemblage d'éléments interchangeables sur des lignes de production. Ces changements ont permis à la fois de mutualiser les efforts, d'augmenter la productivité et de réduire drastiquement les coûts.

La création d'une ligne de produits en utilisant un ensemble de pièces réutilisable n'est pas une idée nouvelle, la société Ford célèbre fabricant automobile a crée la première ligne d'assemblage d'automobiles en utilisant des pièces interchangeables en 1908.

Fondamentalement, l'approche des lignes de produits logiciels propose d'adapter les principes du fordisme au développement logiciel.

L'idée est de transposer les principes de fabrication de nos voitures au développement des logiciels embarqués dans celles-ci. Sauf exception, les véhicules que nous conduisons à l'heure actuelle ne sont plus produits manuellement.

Les processus de fabrication ont été drastiquement rationalisés et ces véhicules sont produits à la chaîne essentiellement afin de maximiser la productivité et de minimiser les coûts.

Différents modèles de voitures sortent des mêmes chaînes de montage en utilisant les mêmes châssis, les mêmes moteurs, les mêmes plans de tests, ...Jusqu'à présent, la *production de logiciels* n'a pas suivi la même évolution. Pourtant, depuis les années 1990 différentes initiatives ont vu le jour afin de promouvoir de nouvelles méthodes de développement basées sur l'assemblage d'éléments réutilisables et sur la systématisation de la réutilisation logicielle. Dans le cadre du développement

logiciel, les principales méthodes qui prônent ces principes ont été regroupés sous un même intitulé : l'approche des « *lignes de produits logiciels* ».

Evidemment, la rentabilité de ces méthodes dépend principalement du nombre de similarités partagées entre les logiciels ainsi que du nombre de logiciels à produire. L'objectif est d'arriver à fournir des logiciels conforment aux exigences du client de manière plus rapide en dépensant moins d'argent et en augmentant la qualité du logiciel.

Le développement des lignes de produits logiciels se focalise sur la production de familles de produits logiciels plutôt que sur la fourniture d'un produit unique.

Ce chapitre présente les notions principales des lignes de produits, il est organisé comme suit : dans la première section nous introduisons l'approche ligne de produits logiciels. Ensuite, dans la deuxième section nous présentons le processus de développement des lignes de produits logiciels qui est structuré en deux sous processus : l'ingénierie du domaine et l'ingénierie des applications. Enfin, nous introduisons la notion de variabilité dans les LdPs.

2. Ingénierie des lignes de produits

L'ingénierie des lignes de produits logiciels est une méthodologie qui permet le développement d'une multitude de produits ou systèmes logiciels avec un gain considérable en termes de coût, de temps et de qualité. C'est une transposition des chaînes de production au monde du logiciel. Il ne s'agit plus de modéliser et de développer des systèmes individuels. L'ingénierie des lignes de produits logiciels s'intéresse plutôt à modéliser et développer une famille de produits.

Une ligne de produits logiciels consiste d'un ensemble de produits représentant des similarités. L'utilisation des lignes de produits logiciels s'appuie sur la réutilisation ainsi que la configuration des systèmes logiciels. La réutilisation a été définie par Krueger comme [1]:

"Le processus de création des systèmes logiciels à partir des logiciels existants au lieu de les créer à partir de zéro"

Dans la section suivante on va citer quelques définitions des lignes de produits logiciels pour donner une idée plus précise sur ce principe.

2.1. Définition des lignes de produits logiciels

La définition des lignes de produits logiciels proposée par Clements et Northrop [2]:

"Une ligne de produits logiciels est un ensemble de produits logiciels qui partagent et gèrent un ensemble de caractéristiques satisfaisant les besoins spécifiques d'un segment de marché particulier ou une mission et qui sont développées à partir d'un même ensemble d'atouts essentiels décrits d'une manière prescrite"

La définition proposée par Krueger :

"A software product line refers to engineering techniques for creating a portfolio of similar software systems from a shared set of software assets using a common means of production." [3]

Cette définition présente une ligne de produits logiciels comme un ensemble de techniques permettant la création de systèmes logiciels similaires s'appuyant sur un ensemble des éléments logiciels partagés. Dans cette définition, on trouve le concept de la portée (*portfolio appelée aussi scope*) d'une ligne de produits. La portée est une notion fondamentale : elle permet de déterminer les produits qui sont inclus dans une ligne de produits et ceux qui n'y appartiennent pas, au moment courant ou dans le futur.

La définition proposée par Lai et Weiss :

"A software product line is a family of products designed to take advantage of their common aspects and predicted variabilities" [4]

Cette définition met en avant la finalité d'une ligne de produits qui est bien de « profiter » d'un ensemble des éléments communs réutilisables. Dans leur proposition, Lai et Weiss mentionnent explicitement la notion de variabilité dans une ligne de produits. Ils expriment le fait que certains aspects sont effectivement

communs mais qu'il existe également des aspects variables au sein des différents artéfacts réutilisables. Enfin, cette définition souligne que les aspects communs et variables sont utilisés de façon explicite.

Jan Bosch propose, la définition suivante :

"A software product line consists of product line architecture and a set of reusable components that are designed for incorporation into the product line architecture. In addition, the product consists of the software products that are developed using the mentioned reusable assets." [5]

Dans cette définition, Jan Bosch met en évidence l'importance de la notion d'architecture logicielle dans une ligne de produits. La ligne de produit est ainsi définie par cette architecture, appelée *architecture de référence* et par un ensemble de composants réutilisables au sein de cette architecture. L'architecture de référence fournit le cadre de développement des composants réutilisables et garantit leur incorporation appropriée. Elle apparaît ainsi comme un guide d'assemblage des artéfacts réutilisables pendant la phase de conception du développement d'un produit. De ce fait, elle permet de conduire à la création de produits individuels par la réutilisation d'artéfacts prévus à cet effet. Cependant, cette définition n'évoque pas le concept de variabilité au sein d'une ligne de produits.

La dernière définition du SEI (*Software Engineering Institute*) [6] :

"A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way." [7]

Cette définition s'applique aux systèmes à forte composante logicielle. Ceci fait référence à des systèmes incluant des aspects matériels mais où le logiciel a un impact majeur sur toutes les étapes du développement telles que la conception, la mise en œuvre, le déploiement et l'évolution [8]. Cette définition précise que les systèmes répondent aux besoins d'un marché existant et s'inscrivent dans une vision globale. Ceci provient du fait qu'une ligne de produit est coûteuse et qu'elle ne peut

être mise en place que pour répondre à un réel enjeu économique. Enfin, cette définition souligne le fait que les produits individuels sont réalisés de manière prescrite en utilisant des artéfacts communs. La manière exacte de construire une famille de produits reste encore floue dans cette définition qui se focalise plutôt sur la perspective économique.

L'ingénierie des lignes de produits logiciels est décomposée en deux phases complémentaires [9]: l'ingénierie de domaine et l'ingénierie d'application. L'ingénierie de domaine adopte la stratégie de développement pour la réutilisation, alors que l'ingénierie d'application s'appuie sur la stratégie du développement avec réutilisation. Le principe derrière ces deux phases rejoint l'idée de compenser le temps de développement des artéfacts communs pendant la première phase par la réutilisation de ces artéfacts pendant la deuxième phase. Ces deux activités seront présentés en détaille dans la section suivante.

2.2. Ingénierie du domaine

L'objectif principal de l'ingénierie du domaine est de produire des artéfacts réutilisables et de fournir les moyens permettant leur utilisation effective pour construire un nouveau produit au sein d'une ligne de produits. On peut voir ces artéfacts comme le résultat du transfert de connaissances avancés sur un domaine (celui de la ligne de produits) dans un monde informatique sous une forme configurable.

Le SEI définit les artéfacts réutilisables comme suit :

"An artifact or resource that is used in the production of more than one product in a software product line. Core assets often include, but are not limited to, the architecture, requirements statements, reference architecture, and reusable software components. More documents can be also included, such as domain models, documentation and specifications, performance models, schedules, budgets estimation, test plans, test cases, work plans, process descriptions." [2]

Chaque artéfact réutilisable est lié à une activité précise du développement de l'ingénierie du domaine. Il apparaît clairement dans cette définition que les artéfacts

réutilisables de lignes de produits ne se limitent pas seulement au code comme un composant logiciel par exemple. Un artéfact réutilisable peut prendre la forme d'un document décrivant des exigences en langage naturel par exemple d'un modèle décrivant par exemple des concepts du domaine (appelés le vocabulaire du domaine) et leurs associations, d'une conception, comme l'architecture de référence de la ligne de produits. Un artéfact peut également correspondre à un plan de production, un plan de test, une description de processus, etc.

La figure [1] présente les principales activités du processus d'ingénierie du domaine ainsi que leurs liens en termes d'entrées requises et de résultats fournis. Cette figure met également en évidence la dépendance entre les processus du domaine. Le développement d'un nouveau produit requiert parfois l'implantation d'un nouvel artéfact pour répondre à certains besoins spécifiques. Dans ce cas, ce nouvel artéfact peut intégrer la plate-forme de la ligne de produits.

Nous détaillons dans les sections suivantes les tâches principales, à savoir l'analyse du domaine, la conception du domaine et la réalisation du domaine.

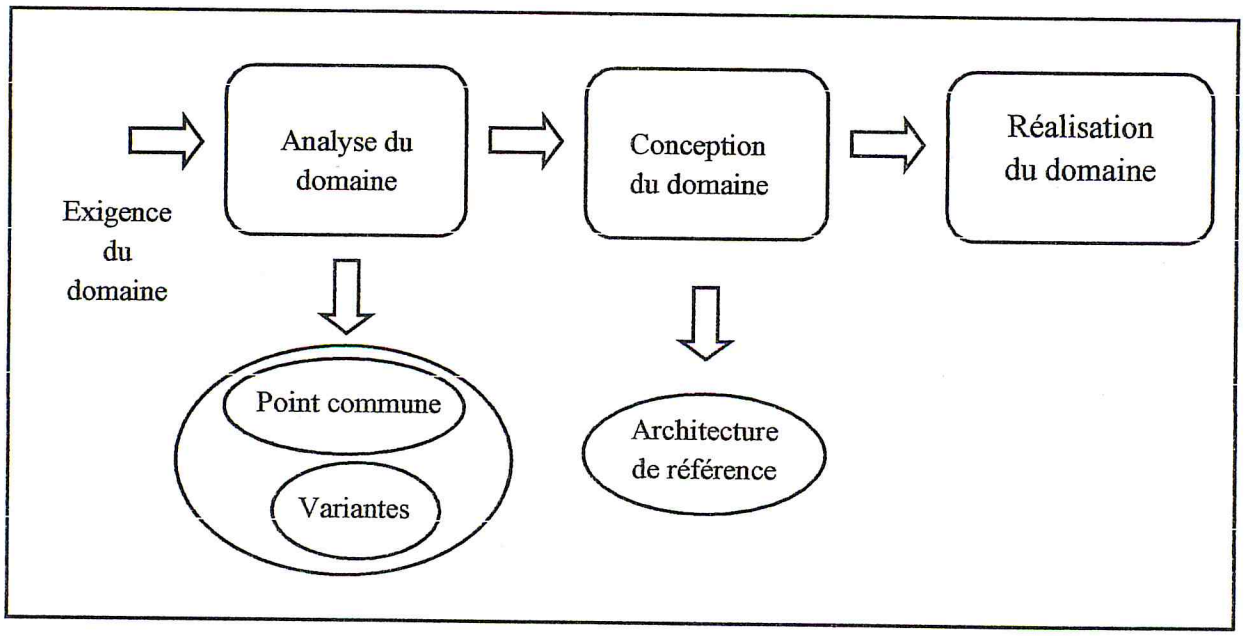


Figure [1] : Le processus de développement de l'ingénierie des domaines [19]

2.2.1. Analyse du domaine

L'analyse du domaine consiste à collecter, extraire et formaliser les informations (connaissances) dans ce domaine, il s'agit d'une transformation des connaissances métier vers des représentations informatique et aussi identifier les règles métier, les contraintes et les exigences.

Dans l'analyse du domaine on extraire les éléments communs pour tous les produits (similarités) et d'autres éléments variables selon les exigences du domaine (variabilités), cette étape est la plus importante parce que le développement des lignes de produit se base sur le concept de variabilités.

L'analyse du domaine est une activité plus complexe que l'analyse des besoins lors du développement d'une application individuelle.

Une activité fondamentale dans cette étape est de déterminer le périmètre fonctionnel (scope), il s'agit de la portée de la ligne de produits, celle-ci définit explicitement l'ensemble des produits à étudier et implicitement les futures produits qui pourront entrer dans la ligne de produits.

2.2.2. Conception du domaine

Le but de la conception du domaine est concevoir les éléments logiciels qui feront partie de la ligne de produit ces éléments sont des composants configurables ainsi que l'architecture de référence.

Clements définit une architecture de référence de la façon suivante [2] :

"Description of the structural properties for building a group of related systems (i.e., product line), typically the components and their interrelationships. The inherent guidelines about the use of components must capture the means for handling required variability among the systems. (This is called a reference architecture)"

Cette définition considère qu'une architecture de référence est la description d'un ensemble de composants et de leurs relations permettant de construire des systèmes logiciels appartenant à une ligne de produits. De plus, l'architecture de

référence doit contenir les informations nécessaires pour gérer la variabilité au sein des systèmes.

Andersson adopte un point de vue similaire [10]. Il met en évidence que l'architecture de référence comporte un processus d'instanciation d'une nouvelle application particulière. Ce processus repose sur l'architecture de référence via les points communs et des points de variabilités.

L'architecture de référence définit de façon abstraite les composants structurels des produits et leurs relations. L'architecture de référence est un élément clé dans une plate-forme de ligne de produits. Généralement, les experts du domaine se focalisent en premier sur ce point. Ils s'appuient sur l'ensemble des modules communs identifiés lors de l'analyse du domaine pour obtenir une première structuration de l'architecture. Cette structuration met en évidence les composants importants et leurs relations. Ensuite, l'architecture est successivement affinée de façon à faire apparaître des points de variation explicites. La création d'un point de variabilité est une décision importante. Dans certains cas, l'architecture de référence permet d'automatiser le développement des produits finaux en définissant de façon systématique comme les artéfacts réutilisables doivent être assemblés. Toutes les architectures des produits spécifiques dans le portefeuille de produits doivent être « conformes » à l'architecture de référence de la ligne de produits.

On définit les composants structurels des produits et leur relation en s'appuyant sur l'ensemble des modules identifiés lors de l'analyse du domaine en premier la structuration de l'architecture de référence en suite l'architecture de référence et successivement affinée de façon à faire apparaître les point de variation. Le nombre des points de variation est très important par ce qu'un grand nombre de point de variation conduit à une ligne de produit inefficace.

Un point de variation peut être caractérisé par un ou plusieurs choix appelés des variantes. On trouve deux types de point de variation :

1- les points de variation « ouvert » : il est possible lors de l'ingénierie d'application d'ajouter une nouvelle variante.

2- les points de variation « fermé » : il n'est pas possible de sortir des choix proposé par le point de variation.

Les points de variation prendre trois formes :

1- Optionnel : la liaison n'est pas indispensable pour le système.

2- Alternatif : la liaison est nécessaire.

3- Multiples : la liaison est nécessaire mais on sélectionne une liaison parmi plusieurs.

2.2.3. Réalisation du domaine

Cette étape consiste à développer les composants principaux constituants de l'architecture, les composants sont adapter a plusieurs contextes d'utilisation. Il existe différentes manières de créer ces composants. Les plus connues sont les techniques suivantes:

- Utilisation de l'héritage en orienté objet. Un composant peut être implanté sous la forme d'un *framework* orienté objet permettant des modifications ou des ajouts de comportement afin de s'adapter à des besoins spécifiques.

- Utilisation de mécanismes d'extension. L'adaptation d'un composant peut être définie par addition dynamique de code comme c'est le cas, par exemple, avec les mécanismes de type « *plug-ins* ». Les *plug-ins* sont parfois communs ou peuvent aussi servir à une application spécifique (il s'agit des variantes).

Dans notre projet nous choisissons de construire un *framework* de composant réutilisable on basant sur le développement orienté composant (qu'on va détailler dans le chapitre 2). Un *framework* fournit un ensemble de fonctions facilitant la création de tout ou d'une partie d'un système logiciel, Il fournit suffisamment de composants logicielles pour pouvoir produire une application rapidement et facile à maintenir, Ces composants sont organisés pour être utilisés en interaction les uns avec les autres.

Pour conclure, la principale activité d'ingénierie du domaine est de fournit les éléments réutilisables du domaine, ces éléments doivent être flexible pour faciliter la maintenance et la réutilisation dans la partie ingénierie d'application.

2.3. Ingénierie d'applications

La deuxième activité concerne la construction du produit logiciel qui consiste en partie à figer certains choix vis-à-vis de la variabilité définie dans la LdP. De toute évidence, certains choix sont incompatibles entre eux. Si l'on reprend l'analogie ci-avant, une voiture comporte généralement un seul moteur, et il faut alors choisir entre une motorisation essence ou diesel. De la même manière, un choix particulier lors de la dérivation d'un logiciel peut exclure certaines variantes.

Une LdP doit donc aussi intégrer des contraintes de cohérence permettant de faciliter les choix lors de la dérivation. Bien que cette activité soit fondamentale à la réussite de l'ingénierie des lignes de produits dans le monde du logiciel, elle n'a pas reçu une attention approfondie que relativement récemment.

De façon générale, l'ingénierie des applications est le processus de création et de mise à disposition des logiciels de la ligne de produits. Ce processus de développement se base sur les artéfacts réutilisables produits lors de la phase d'ingénierie du domaine. Plus précisément, la construction d'un produit logiciel spécifique est un processus d'assemblage et de configuration des artéfacts réutilisables en conformité avec l'architecture de référence.

Comme dans l'ingénierie du domaine, la mise en œuvre de l'ingénierie des applications est divisée en trois activités principales. Ces trois activités du développement sont : l'analyse d'application, la conception d'applications et finalement la réalisation d'applications. La figure [2] illustre ce processus de développement. Ces trois activités majeures de l'ingénierie des applications sont présentées plus en détail dans la suite de cette section.

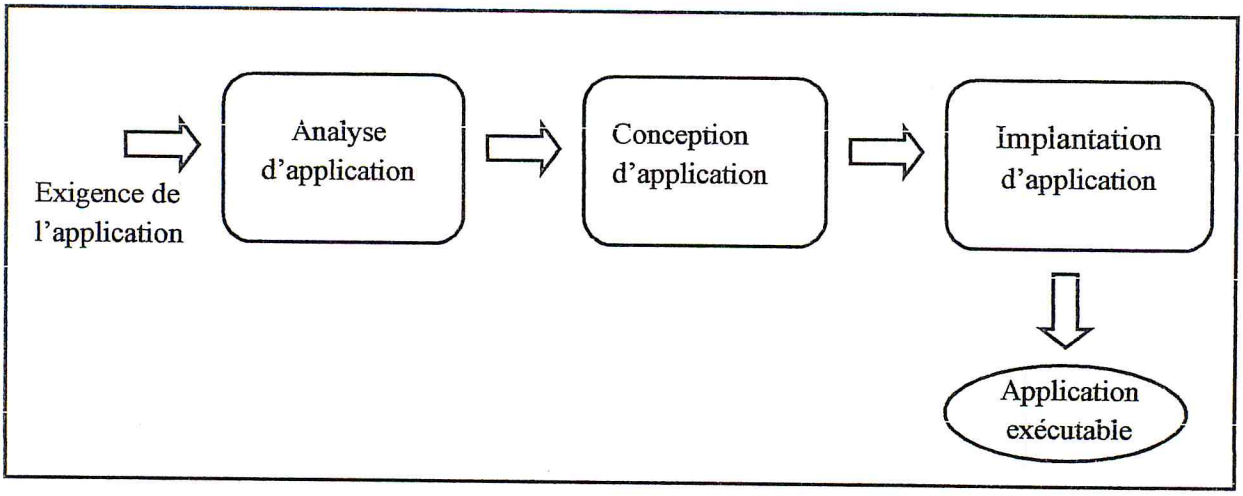


Figure [2]: Processus de développement de l'ingénierie des applications [19]

2.3.1. L'analyse d'application

Cette activité consiste à l'analyse des exigences du produit, ces exigences sont obtenues à partir des interactions avec les différents intervenants du développement de produits tel que des utilisateurs finaux.

On trouve trois types des exigences :

- les exigences du domaine.

- les exigences prévues dans l'analyse du domaine sous forme de variabilités.

- les exigences non prévues dans l'analyse du domaine qui donnent lieu à des configurations particulières.

2.3.2. La conception d'application

Cette activité consiste à obtenir l'architecture du produit à partir de l'architecture globale définit pendant la conception du domaine ainsi que les points de variation définies et planifiés dans la conception du domaine sont instanciés et configurés durant cette étape du développement pour créer une architecture applicative.

2.3.3. La Réalisation d'application

Cette activité consiste à la réalisation du produit concret souhaité. Les composants qui correspondent aux parties sélectionnées dans l'étape précédente sont alors assemblés.

3. La Variabilité

La variabilité dans les lignes de produits a été définie [11] comme le « regroupement des caractéristiques qui différencient les produits de la même famille ». Ce concept est lié à celui de « point de variation ». Un point de variation identifie la partie du système où une variation va être introduite dans la famille de produits, donnant naissance à un nouveau membre ou produit dans la ligne. Nous pouvons également adopter la définition donnée dans [12] d'un point de variation comme « la description de l'ensemble des différences entre les systèmes finaux d'une famille de produits ».

Une classification des différents types de variabilité dans les lignes de produits a été présentée dans [12]:

- Similarité (*Commonality*) : C'est une caractéristique qui est commune à tous les produits de la famille. Cette caractéristique est implémentée comme partie de la plateforme ou noyau commun de la famille.

- Variabilité (*Variability*) : C'est une caractéristique qui peut être commune à quelques produits mais pas à tous les membres de la famille de produits. Cette caractéristique doit être modélisée comme un point de variation et implémentée de manière à ce qu'elle soit présente seulement dans les produits sélectionnés.

- Produit-Spécifique (*Product-specific*) : C'est une caractéristique qui est incluse seulement dans un ou dans un sous ensemble de produits de la famille. Elle est généralement la réponse à une demande spécifique du client pour le produit concerné. Cette caractéristique ne sera pas intégrée dans la plateforme de la ligne de produits mais elle doit être compatible avec celle-ci.

L'activité d'identifier, représenter, exploiter, implémenter et faire évoluer la variabilité dans une ligne de produits, est appelée la « gestion de la variabilité ».

Puisque toute l'architecture de la ligne de produits et ses propriétés dépendent de la gestion de la variabilité, cette activité constitue l'enjeu principal lors de la conception d'une famille de produits [13].

3.1. La gestion de la variabilité

Chaque **produit** appartenant à une ligne de produits logiciels détermine un contexte particulier. Chaque produit se compose d'un ensemble d'éléments logiciels représentés par des « ***features*** ». Chaque *feature* regroupe les exigences devant être satisfaites par les produits logiciels finaux intégrant cette *feature*. L'objectif d'une *feature* est de délimiter un ensemble d'*exigences* fortement connexes et directement réutilisables par différents produits finaux. Les *features* sont successivement décomposées en sous-*features* jusqu'à obtenir des *features* terminales. Idéalement, chaque *feature* terminale est associée à un *élément logiciel réutilisable* (composant, service, ...) implémentant les exigences déterminées par la *feature* correspondante [14].

La gestion de la variabilité joue donc un rôle essentiel afin de déterminer dans quel contexte sous quelles conditions et comment une *feature* (et donc l'élément logiciel associé) peut être réutilisée de manière optimale. La gestion de la variabilité est décomposée en trois activités principales [14]:

• L'identification de la variabilité qui détermine :

- les *features* terminales qui permettent de distinguer les produits logiciels appartenant à la même ligne de produits (**variabilités**).

- les **contraintes** qui existent entre ces *features*.

- où, comment et pourquoi ces variabilités peuvent apparaître (**points de variation**). Les points de variation correspondent souvent aux *features* non terminales qui facilitent la structuration des *features* terminales.

• L'implémentation de la variabilité qui détermine quels mécanismes (héritage, paramétrisation, configuration, génération, template instanciation, plugins, ...) peuvent être utilisés afin de la retranscrire au niveau du code ou de l'architecture.

• L'évolution de la variabilité qui indique comment éviter d'introduire des conflits ou de créer des interactions inattendues entre *features* lorsque de nouveaux points de variation ou de nouvelles variabilités apparaissent.

3.2. Modélisation de la variabilité

Après avoir défini le concept de variabilité, nous avons besoin de trouver un moyen pour l'exprimer de manière claire, c'est-à-dire, de trouver un langage de modélisation qui nous permet d'inclure les concepts liés aux lignes de produits logiciel pour représenter notre système.

De nombreux langages permettent de décrire graphiquement la variabilité. Le premier langage spécifiquement dédié à la modélisation de la variabilité a été proposée par Kang en 1990[15]. Ce langage de modélisation définit différentes constructions (graphiques et textuelles) afin de modéliser les *features* et les relations existant entre elles au moyen d'un modèle particulier appelé « **Feature Diagram** ». Ce langage se voulait avant tout minimal et simple d'utilisation en comparaison avec d'autres langages de modélisation plus complexes tels que UML.

La figure [3] représente un Feature Diagram qui modélise de manière compacte une ligne de produits de téléphones mobiles.

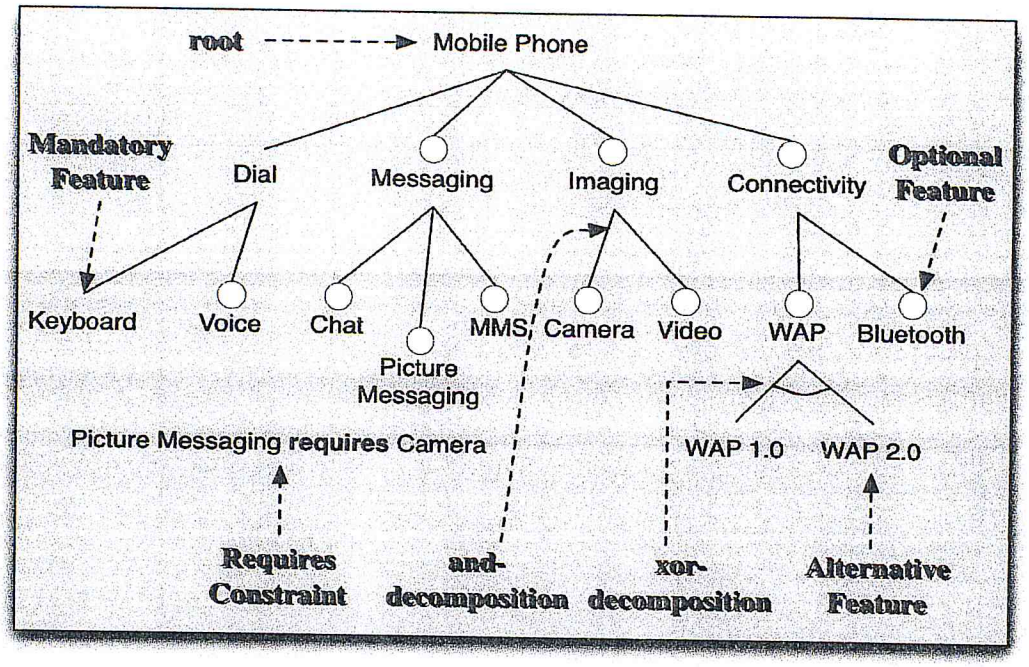


Figure [3]: Feature Diagram Mobile Phone [14]

Un Feature Diagram se présente généralement sous la forme d'un arbre composé de nœuds reliés par des arêtes. Chaque nœud correspond à une *feature* et chaque arête correspond à une relation entre deux *features*. Différents types de *features* existent :

• **La racine** : le Un Feature Diagram est caractérisé et identifié par une *feature* spécifique appelée la racine (*root*). Cette racine détermine le point d'entrée du diagramme qui est l'unique nœud ne possédant pas de parent.

• **Les *features* optionnelles :** *Si* une *feature* est optionnelle (*optional*) alors si un de ses parents est sélectionné elle ne l'est pas nécessairement. Graphiquement, une *feature* optionnelle est décorée par un cercle vide.

• **Les *features obligatoire* :** Si une *feature* est obligatoire (*mandatory*) alors si un de ses parents est sélectionné elle est dès lors aussi sélectionnée. La racine est par définition toujours obligatoire. Graphiquement, une *feature* obligatoire n'est pas décorée par un cercle vide.

Ces features sont reliées par différents types de relations:

• Les **décompositions**. Les *features* peuvent être décomposées en sous-*features* suivant différents types de décompositions. Ces décompositions définissent des contraintes entre des *features* partageant le même parent.

- *And*-decomposition. Cette décomposition signifie que si le parent est sélectionné alors ses enfants aussi.

- *Xor*-decomposition. Cette décomposition signifie que si le parent est sélectionné alors un et un seul de ses enfants peut être sélectionné.

- *Or*-decomposition. Cette décomposition signifie que si le parent est sélectionné alors au moins un de ses enfants peut être sélectionné.

• Les **contraintes**. Dans certains cas, on veut pouvoir exprimer des contraintes entre des *features* qui ne partagent pas le même parent. Ces contraintes peuvent être établies entre toutes les *features* appartenant au même Feature Diagram. C'est pourquoi elles sont représentées de manière textuelle afin d'éviter tout surchargement réduisant la lisibilité du modèle. Les deux contraintes les plus utilisées sont :

- *Requires*. La contrainte *requires* entre deux *features* (*f1 requires f2*) permet d'exprimer que si la *feature f1* est sélectionnée alors *f2* doit nécessairement l'être, mais pas inversement.

- *Excludes*. La contrainte *excludes* entre deux *features* (*f1 excludes f2*) permet d'exprimer que si une des deux *features* est sélectionnée alors l'autre *feature* ne peut pas l'être en même temps pour le même produit.

En pratique, un Feature Diagram décrit la majorité des produits développés par l'entreprise et comprend des milliers de *features* et de contraintes. Ce modèle devient donc stratégique pour l'entreprise.

3.3. Orthogonal Variabilité Modèle (OVM)

OVM est un langage de modélisation permet de déterminer la variabilité d'une ligne de produits logiciels d'une manière orthogonale, en d'autres termes, il fournit une vue en coupe transversale de la variabilité entre tous les objets de la ligne de

produits. OVM inter-relie la variabilité dans les modèles de base telle que les modèles d'exigences, les modèles de conception, les modèles de composants.

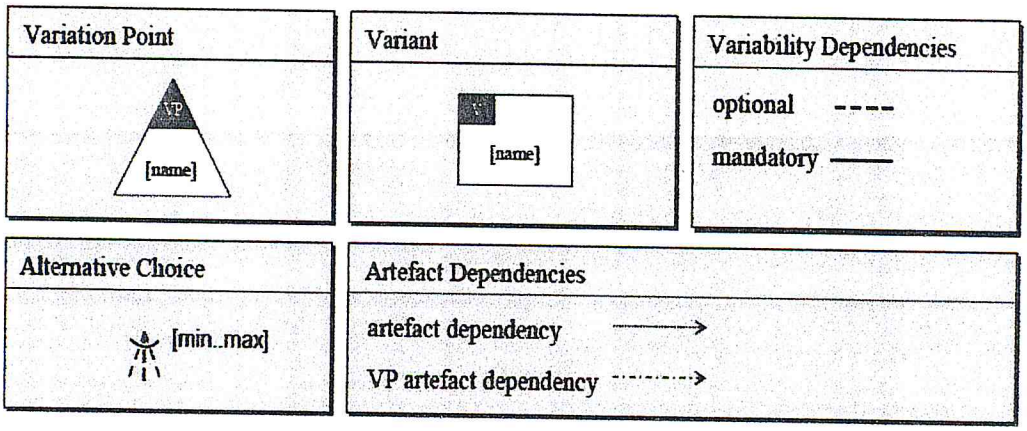


Figure [4]: Les notations graphiques d'OVM [20]

Un OVM est composé de deux éléments principaux: les points de variation (VP) et variantes (V). En cet article, nous nous référons à ces éléments OVM comme éléments variables. Un point de variation est un aspect qui indique une variation qui peut pendre une gamme de produits, ce point de variation et choisis par le client ou un ingénieur de la ligne du produit logiciel. Une variante est liée à un point c'est la façon dont cette point de variation peut varier. La relation entre un point de variation et d'une variante et comme la relation Parent-Child. Les points de variation ont au moins un enfant et chaque variante possède au plus un parent.

Bien que les Feature Diagram et similaire à OVM, elles diffèrent principalement dans leur structure et dans la façon dont ils se rapportent à une information. Un Feature Diagram est uniquement composé d'éléments organisés en un seul arbre et une fonction de racine unique, tandis qu'un OVM se compose de points de variation et de variantes organisés en un ou plusieurs arbres. Dans le Feature Diagram, la fonction racine est toujours obligatoire, c'est à dire qu'il fait partie de tous les produits et par conséquent, il n'y a pas de produit vide. D'autre part, dans OVM, les points de variation peuvent être en option. Cela permet la configuration d'un produit sans aucune variante ou point de variation.

4. Les perspectives et défis

Malgré leurs nombreux avantages, les lignes de produits logiciels ne constituent pas la solution miracle à tous les problèmes de productivité et de réutilisation. Dans cette section, nous distinguons les différentes perspectives et défis inhérents à cette approche.

4.1. Les perspectives

Les principaux avantages associés à une approche de type ligne de produits logiciels concernent principalement :

L'augmentation de la productivité [16] et la diminution des coûts de développement et de maintenance des produits finaux. Les développeurs ont la possibilité de se baser sur des exigences uniformisées et cohérentes et de réutiliser des composants spécifiquement conçus afin de maximiser la réutilisation. De plus, lorsque les produits sont mis en production leur maintenance est beaucoup mieux maîtrisée. L'augmentation de la qualité des produits finaux. Les développeurs réutilisent des composants de meilleure qualité en suivant des standards de programmation communs à toute la ligne de produits. On évite ainsi la duplication de code et on favorise un meilleur contrôle de la qualité logicielle.

4.2. Les défis

Une ligne de produits logiciels ainsi que les bénéfices qui y sont associés n'apparaissent pas comme par magie. Des efforts conséquents doivent être consentis et des défis importants relevés:

L'investissement initial, nécessaire à la mise en place d'une telle approche est relativement important malheureusement le retour sur investissement n'est pas immédiat. Dans la majorité des cas, ces deux facteurs poussent les petites et moyennes entreprises à abandonner ce type d'approche.

L'investissement initial est considérable de même si le projet peut être conduit de manière itérative, il est toujours nécessaire d'avoir une vue d'ensemble non pas sur un logiciel mais sur un ensemble de logiciels incluant à la fois les logiciels passés, présents et à venir.

Le retour sur investissement n'est certainement pas immédiat et la rentabilité ne sera vraisemblablement atteinte qu'à moyen ou long terme.

Généralement, le coût des premiers logiciels développés à partir de la ligne de produits sera même plus élevé que dans la situation précédente.

Il faudra attendre la production d'un certain nombre de logiciels (Break even point) avant de pouvoir générer de réels bénéfices à moyen ou long terme (Figure [5]).

Préalablement, à la mise en place de la ligne de produits logiciels, les coûts et les bénéfices sont pratiquement impossibles à estimer de manière fine.

Ce qui a souvent pour conséquence d'empêcher les décideurs d'évaluer les budgets nécessaires et d'établir des plannings raisonnables.

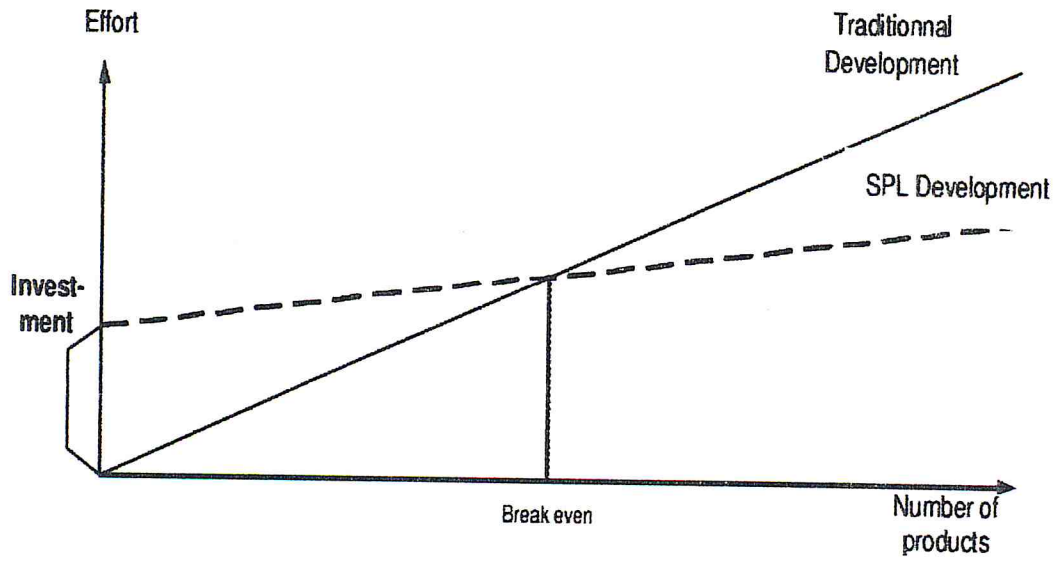


Figure [5]: Rentabilité des lignes de produits logiciels [17]

La manière de concevoir et de développer une ligne de produits logiciels implique une évolution radicale des mentalités et des changements organisationnels importants.

Chaque acteur doit être convaincu des bénéfices potentiels à la fois pour lui-même et pour l'organisation dans son ensemble. Force et de constater que ce

changement de mentalité est généralement plus facile à entreprendre dans des entreprises développant des logiciels embarqués dans des appareils électroniques.

En effet, ces entreprises appliquent depuis des années les principes des lignes de produits au niveau de la conception physique de leurs appareils qui résultent de la composition de différents modules électroniques réutilisables.

En revanche, l'acceptation et l'application des principes des lignes de produits logiciels sont plus délicates dans des entreprises développant des systèmes d'information plus classiques.

Ces difficultés se justifient par les exigences clients beaucoup plus contraignantes et, d'autre part, le principe de réutilisation qui n'est pas encore entré dans les mœurs.

Une attention toute particulière doit être portée sur l'évolution et la gestion du changement dans les lignes de produits logiciels. Gérer l'évolution d'un logiciel est déjà complexe.

Gérer l'évolution d'un ensemble de logiciels dont la cohérence doit être préservée est extrêmement complexe. Tout changement portant sur un élément logiciel partagé par plusieurs produits finaux doit être contrôlé méticuleusement.

Tous les impacts potentiels sur les produits finaux et sur leurs éléments logiciels respectifs doivent être envisagés avant de mettre ces changements en production.

De ce fait, la flexibilité d'une ligne de produits logiciels est souvent réduite afin d'améliorer son efficacité.

La quantité d'information à rassembler afin d'établir une nouvelle ligne de produits est souvent conséquente.

La difficulté consiste à identifier les personnes et les documents permettant de recueillir suffisamment d'information afin de comparer ces produits, d'identifier leurs similarités et leurs variabilités.

Dans la majorité des cas, il est nécessaire de clarifier et de compléter la documentation des produits existants, d'entamer une réingénierie et d'anticiper les différentes avancées et innovations envisageables pour les nouvelles mises en production.

5. conclusion

Dans ce chapitre nous avons présenté les concepts de base des lignes de produits logiciel notamment le concept de la variabilité sa gestion et sa modélisation ainsi que les avantages et défis liés au développement des lignes de produits.

Chapitre 2 : l'orienté composant

1. Introduction

La réutilisabilité et la composabilité ont toujours été des objectifs importants de l'ingénierie du logiciel. En leur temps, les approches procédurales et orientées objet ont chacune apporté leurs solutions. Actuellement, la tendance pour les applications complexes et plus récemment dans le domaine de l'embarqué promeut un développement à base de composants.

Aux prémices de ce paradigme, on retrouve le principe de la séparation des préoccupations déjà énoncé dans les années soixante-dix [21] mais ce n'est que bien plus tard qu'un début de consensus semble se dégager [22] pour caractériser un composant logiciel en tant qu'unité de composition et de déploiement doté d'interfaces contractuelles.

Au-delà, des différences de définitions entre les modèles conçus pour les « applications génériques » ou pour celles plus spécifiques de l'embarqué, l'objectif reste le même : augmenter la réutilisabilité et permettre la création de nouvelles entités par assemblage d'entités existantes.

Dans ce chapitre, nous allons présenter les concepts de base liées à l'orienté composant ainsi que la conception orientée aspect.

2. L'orientés composants

D'une manière générale, un composant est perçu comme une boîte qui abstrait un comportement et doté de points d'entrée et de sortie explicites (ses interfaces) qui lui permettent de communiquer avec d'autres composants de son environnement. L'idée est alors de n'exprimer – ou de n'abstraire – que le strict nécessaire sur ces interfaces pour caractériser précisément le composant ; il s'agit alors d'être en mesure de le déployer de manière indépendante au sein de contextes applicatifs différents.

Considérant les approches orientées composants, nous faisons la distinction entre les deux points suivants [23] :

– Le **modèle de composant** qui spécifie les concepts du langage et les conventions à adopter par le concepteur. En d'autres termes, un tel modèle détermine « l'**espace de conception** » proposé au concepteur pour construire son système.

– Le **canevas à composant** qui désigne l'infrastructure nécessaire à l'étape de conception et d'exécution pour mettre en œuvre le modèle comme par exemple la gestion des ressources nécessaires aux composants, la prise en charge de leurs assemblages et interactions, etc.

Le composant et le connecteur représentent les éléments de base dans la spécification d'architecture logicielle [24] :

2.1. Les composants

Un composant logiciel est une entité responsable de la réalisation d'une ou plusieurs fonctionnalités bien précises dans une architecture à un certain niveau d'abstraction. Il peut être aussi petit qu'une procédure simple ou aussi grande qu'une application [25]. Un serveur, une base de données, une fonction mathématique sont des exemples de composants. Le composant offre une meilleure structuration de l'application et permet de construire un système par assemblage de briques élémentaires en favorisant la réutilisation de ces briques [26].

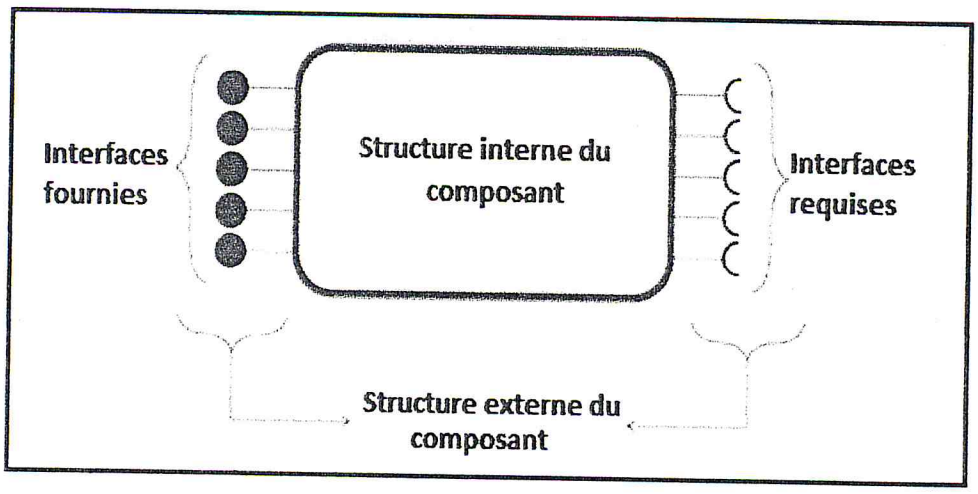


Figure [6] : Représentation d'un composant [24]

Il existe principalement deux parties dans un composant. Une première partie dite extérieure correspond à son interface. Elle comprend la description des interfaces fournies et requises par le composant (Figure [6]). Elle définit les interactions du composant avec son environnement. La seconde partie correspond à son implantation et permet la description du fonctionnement interne du composant. Les caractéristiques globales d'un composant définies par Med vidovic et Taylor [27] sont les suivantes :

- l'interface
- le type
- la sémantique
- les contraintes
- les propriétés non fonctionnelles

2.1.1. L'interface d'un composant

L'interface d'un composant est la description de l'ensemble des services offerts et requis par le composant sous la forme de signature de méthodes, de type d'objets envoyés et retournés, d'exceptions et de contexte d'exécution. L'interface est un moyen d'expression des liens du composant ainsi que ses contraintes avec l'extérieur. L'interface peut englober une description comportementale souvent référencée par le terme typage comportemental. Le typage comportemental indique les règles de mise en œuvre d'une interface.

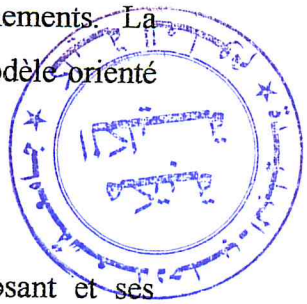
2.1.2. Le type d'un composant

Le type d'un composant est un concept représentant l'implantation des fonctionnalités fournies par le composant. Il s'apparente à la notion de classe que l'on trouve dans le modèle orienté objet. Ainsi un type de composant permet la réutilisation d'instances de même fonctionnalité soit dans une même architecture, soit dans des architectures différentes. En fournissant un moyen de décrire de manière explicite les propriétés communes à un ensemble d'instances d'un même

composant la notion de type de composant introduit un classificateur qui favorise la compréhension d'une architecture et sa conception.

2.1.3. La sémantique d'un composant

La sémantique du composant est exprimée en partie par son interface. Cependant, l'interface telle que décrite ci-dessus ne permet pas de préciser complètement le comportement du composant. La sémantique doit être enrichie par un modèle plus complet et plus abstrait permettant de spécifier les aspects dynamiques ainsi que les contraintes liées à l'architecture. Ce modèle doit garantir une projection cohérente de la spécification abstraite de l'architecture vers la description de son implantation avec différents niveaux de raffinements. La sémantique d'un composant s'apparente à la notion de type dans le modèle orienté objet.



2.1.4. Les contraintes d'un composant

Les contraintes définissent les limites d'utilisation d'un composant et ses dépendances intra composants. Une contrainte est une propriété devant être obligatoirement vérifiée sur un système ou une de ces parties. Si celle-ci est violée le système est considéré comme un système incohérent et inacceptable. Elles permettent ainsi de décrire de manière explicite les dépendances des parties internes d'un composant comme la spécification de la synchronisation entre composants d'une même application (dépendance intra composant).

2.1.5. Les propriétés non fonctionnelles d'un composant

Les propriétés non fonctionnelles (propriétés liées à la sécurité, la performance, la portabilité, etc.) doivent être exprimées à part, permettant ainsi une séparation dans la spécification du composant des aspects fonctionnels (aspects métiers de l'application) et des aspects non fonctionnels ou techniques (aspects transactionnel, de cryptographie, de qualité de service).

2.2. Les Connecteurs

Le connecteur correspond à un élément d'architecture qui modélise de manière explicite les interactions entre un ou plusieurs composants en définissant les règles qui gouvernent ces interactions [28]. Par exemple un connecteur peut décrire des interactions simples de type appel de procédure ou accès à une variable partagée mais aussi des interactions complexes telles que des protocoles d'accès à des bases de données avec gestion des transactions la diffusion d'événements asynchrones ou encore l'échange de données sous forme de flux [29].

Un connecteur comprend également deux parties. La première correspond à la partie visible du connecteur c'est-à-dire son interface qui permet la description des rôles des participants à une interaction. La seconde partie correspond à la description de son implantation. Il s'agit là de la définition du protocole permettant la mise en œuvre du protocole associé à l'interaction. Les exemples de connecteurs incluent des formes simples d'interaction comme des pipes des appels de procédure et l'émission d'évènements. Les connecteurs peuvent également représenter des interactions complexes comme un protocole client/serveur ou un lien SQL entre une base de données et une application. Six caractéristiques importantes définies par Medvidovic et Taylor [27] sont à prendre en compte pour spécifier de manière exhaustive un connecteur. Ces caractéristiques sont les suivantes :

- l'interface
- le type
- la sémantique
- les contraintes
- l'évolution
- les propriétés non fonctionnelles

2.2.1. L'interface

L'interface d'un connecteur appelée aussi rôles dans certains langages de description d'architecture tel que Wright [30] définit les points connexions entre

connecteurs et composants. Elles servent à déclarer les participants à l'interaction décrite par le connecteur. Comme celles des 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.

2.2.2. Le type

Le type d'un connecteur correspond à sa définition abstraite qui reprend aux mécanismes de communication entre composants. Il permet la description d'interactions simples ou complexes de manière générique et offre ainsi des possibilités de réutilisation de protocoles. Par exemple la spécification d'un connecteur de type RPC qui relie deux composants définit les règles du protocole RPC.

2.2.3. La sémantique

Comme pour les composants la sémantique des connecteurs est défini par un modèle de haut niveau spécifiant le comportement du connecteur. A l'opposé de la sémantique du composant qui doit exprimer les fonctionnalités déduites des buts ou des besoins de l'application, la sémantique du connecteur doit spécifier le protocole d'interaction. De plus, celui-ci doit pouvoir être modélisé et raffiné lors du passage d'un niveau de description abstraite à un niveau d'implantation.

2.2.4. Les contraintes

Les contraintes permettent de définir les limites d'utilisation d'un connecteur c'est-à dire les limites d'utilisation du protocole de communication associé. Une contrainte est une propriété devant être vérifiée sur un système ou sur l'une de ses parties. Si celle-ci est violée le système est considéré comme un système incohérent et inacceptable. Par exemple le nombre maximum de composants interconnectés à travers le connecteur peut être fixé et correspond alors à une contrainte.

2.2.5. L'évolution d'un connecteur

Le changement des propriétés (interface, comportement) d'un connecteur doit pouvoir évoluer sans perturber son utilisation et son intégration dans les applications

existantes. Il s'agit de maximiser la réutilisation par modification ou raffinement des connecteurs existants.

2.2.6. Les propriétés non fonctionnelles

Les propriétés non fonctionnelles d'un connecteur concernent tout ce qui ne découle pas directement de la sémantique du connecteur. Elles spécifient des besoins qui viennent s'ajouter à ceux déjà existants et qui favorisent une implantation correcte du connecteur. Par exemple elles peuvent concerner la performance ou la sécurité. La spécification de ces propriétés est importante puisqu'elle permet de simuler le comportement à l'exécution, l'analyse, la définition de contraintes et la sélection des connecteurs.

3. Les modèles à base de composants

3.1. Le modèle à composants *Fractal*

Fractal [31], réalisé par France Telecom R&D et par l'INIA vise à autoriser une définition, une configuration et une reconfiguration dynamique d'une architecture à base de composants. Un composant Fractal est formé d'une membrane et d'un contenu.

La membrane définit par l'intermédiaire d'un ensemble d'interfaces, les services requis et fournis par le composant. Le contenu d'un composant Fractal permet de différencier deux sortes de composant : les primitifs et les composites. Le contenu d'un composant primitif identifie un module logiciel (classe, ensemble de fonctions, *etc.*) permettant de réaliser les services du composant primitif. Le contenu d'un composite définit un assemblage de composants permettant de mettre en œuvre les services du composite.

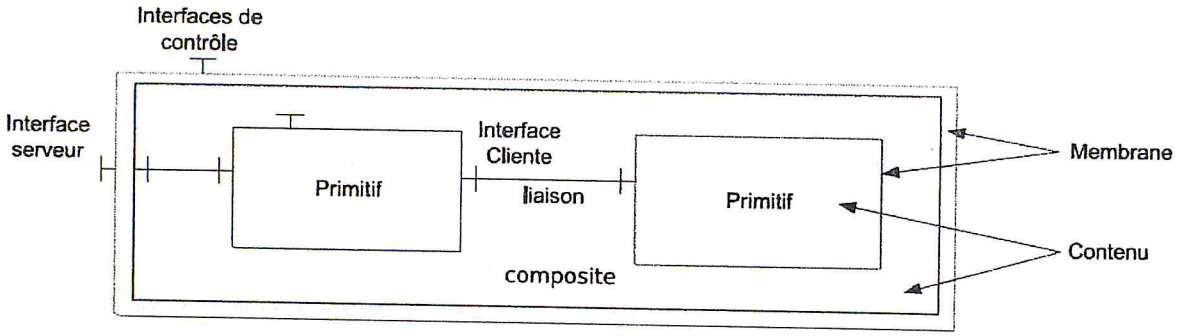


Figure [7]: Modèle de composant Fractal [33]

Fractal met à disposition un ensemble d'interfaces de contrôle qui sont la gestion du cycle de vie (*life-cyclecontroler*), du nom (*name-controller*), des liaisons (*binding-controler*), des attributs (*attribute-controler*), des parents (*super-controller*) et du contenu (*content-controler*) pour les composites. Le modèle étant extensible, il est possible de définir ses propres interfaces de contrôle.
Une interface Fractal métier est du type client ou serveur. Une interface serveur identifie les services offerts par un composant alors qu'une interface client spécifie les fonctionnalités qu'un composant requiert pour son fonctionnement.
Fractal ne possède pas de notion de connecteur explicite avec une sémantique de processus. Cependant, il utilise la notion de *liaison* pour spécifier les interactions entre composants. Une liaison Fractal est définie comme un lien orienté entre une interface client et une interface serveur. Ce lien permet aux composants d'interagir.

3.2. SOFA

SOFA (*SOFtware Appliance*) [32] est un modèle de composant développé par l'université Charles (Prague). Son objectif est la construction d'application à partir d'une composition de composants.

SOFA propose les éléments caractéristiques des langages de description d'architecture à savoir le composant et le connecteur.

Dans SOFA les composants sont primitifs ou composites. Un composite est défini comme un assemblage de sous-composants alors qu'un primitif ne peut contenir de

sous-composants. Un composite ne contient pas de fonctionnalités propres toutes les fonctionnalités sont définies au sein des primitifs.

Les composants de SOFA sont définis par leur *frame* et leur *architecture*. La notion de *frame* est à rapprocher à la notion de type de composant. C'est une approche boîte noire du composant. Un *frame* définit les interfaces requises et fournies par le composant. Elle déclare aussi de manière optionnelle certaines propriétés permettant de paramétrer le composant.

SOFA fournit trois types de connecteur prédéfinis : CSProcCall avec une sémantique d'appel de procédure, EventDelivery avec une sémantique d'envoi de message et DataStream pour une sémantique d'échange de flux. Il fournit aussi un type de connecteur utilisateur qui permet de définir sa propre sémantique.

3.3. Le modèle à composants IASA

Un composant IASA (Integrated Approach to Software Architecture) est vu de l'extérieur comme une boite noire qui communique avec le monde externe grâce à des ***Ports*** qui définissent les services qu'il peut offrir ou requérir. La vue interne d'un composant primitif est inaccessible tandis que celle d'un composant composite est bien définie, elle consiste en trois parties : *Partie Opérationnelle*, Partie *Aspect et Partie Contrôle.* [33]

- **Partie Contrôle** : a pour rôle de charger et configurer les composants internes et satisfaire la demande des composants externes en délégant ces demandes à un composant interne.

- **Partie Aspect** : C'est les fonctionnalités techniques qui ne font pas partie de la logique du métier de l'application.

- **Partie Opérationnelle** : Regroupe les composants métier (Opérations principales pour le fonctionnement de l'application)

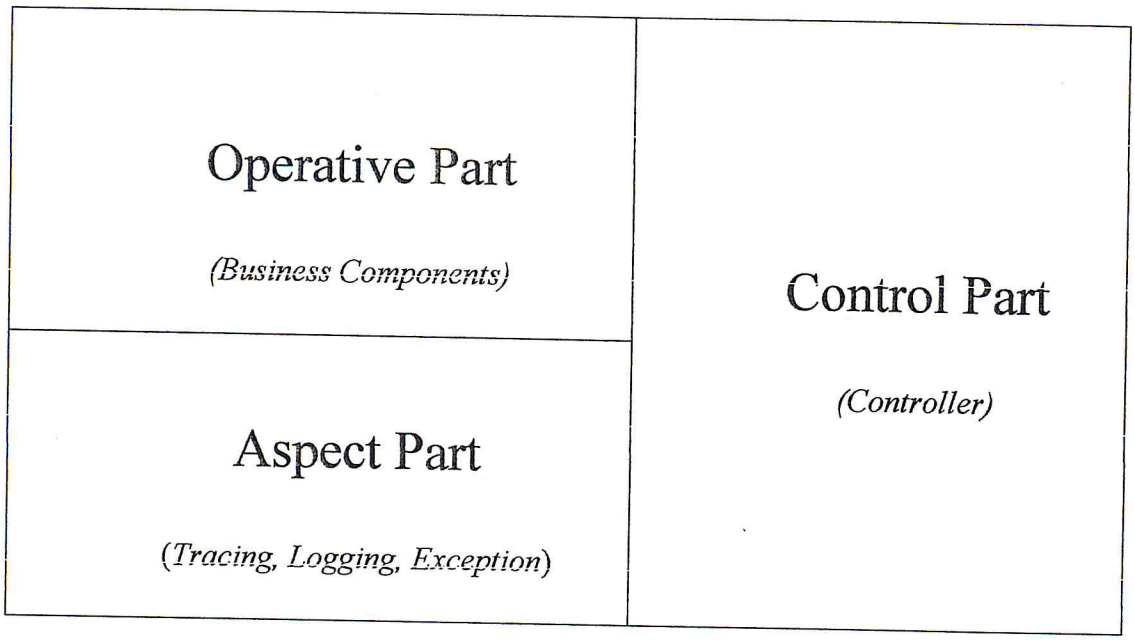


Figure [8] : les différentes parties de la vue interne d'un composant selon IASA

L'instanciation d'un composant se fait dans le contexte du concept d'enveloppe qui sert principalement à isoler la vue interne d'un composant de son environnement d'exploitation, l'enveloppe détient toutes les ressources dont il a besoin pour le support de la communication et de la spécification de la connexion inter-composants. Un point d'accès est le plus petit élément dans la spécification d'une application il définit les ressources requises ou fournies qui peuvent être des données ou des opérations. On peut regrouper un ensemble de points d'accès dans le concept de Port. [34]

Description de l'architecture avec IASA

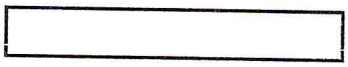
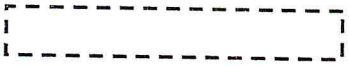
Le composant composite dont on veut décrire la vue interne est représenté par un cadre qui englobe les autres éléments, la partie de contrôle est située en bas à droite et la partie aspect en bas à gauche.

Une interface (port) est représentée avec IASA sous la forme d'un petit carré de couleur grise pour les interfaces fournies et de couleur blanche pour les interfaces requises.

- Interface fournie 
- Interface requise 

Les sous composants faisant partie du métier sont définis dans la partie opérationnelle ils sont définis sous forme de boites contenant le nom du sous composant et ses interfaces (sur le contour). Les boites ayant un contour pointillé sont des composants externes qui ne font pas partie du composant décrit mais comme ce dernier les utilise alors on les intègre dans sa description pour que son fonctionnement devienne plus clair. Les aspects sont définis de la même façon que les sous composants mais à l'intérieur de la partie aspect.

Un connecteur est défini par une ligne joignant deux interfaces et la coupe d'un aspect est définie par une ligne pointillée entre l'aspect et le point de jonction (un port spécifique).

- Le composant interne 
- Un composant externe 
- Un connecteur
- Une coupe

Dans notre projet de fin d'étude nous avons choisi d'utiliser le model de composant de l'approche IASA.

4. Introduction à la Conception Orientée Aspect

Les premières idées sur le concept des aspects existent depuis longtemps déjà. Pour la première apparition du concept, tel qu'il est à l'heure actuelle il faut remonter à 1997. Il s'agit essentiellement d'une idée issue de la communauté des chercheurs. Gregor Kiczales est souvent cité comme étant un des créateurs du concept orientée aspect. Il dirigeait l'équipe du PARC (Palo Alto Research Center) de Xerox qui a développé la POA (Programmation Orientée Aspect) [35].

Le développement Orienté Aspect (Aspect-Oriented Software Development) vise à améliorer la séparation des préoccupations. C'est une propriété cruciale pour développer des systèmes plus compréhensibles et plus facile à la gestion et la maintenance [33].

Parmi les principaux avantages de la POA nous pouvons citer [35]:

• Augmentation de la productivité: L'utilisation des aspects évite d'avoir à tout réécrire (redondance) et concevoir de nouveau certaines parties d'un programme.

• Diminution de la complexité du projet: En séparant le projet en diverses sections (modularité) la complexité est réduite.

• Amélioration de la qualité logicielle à différents points de vue (maintenance, test, réutilisation).

AOSD (Aspect Oriented Software Development) introduit une ou plusieurs dimensions de décomposition qui permettent la modularisation des préoccupations transverses. Elle permet d'intégrer ces solutions en introduisant les nouveaux concepts d'aspect, point de jonction, greffon (advice code), et de coupe [33].

Une application Orientée Aspect consiste en deux parties [33] :
• Métier : constituée des classes qui font la base de l'application et implémentent la logique du métier de l'application.
• Aspects : intègre les éléments supplémentaires (classes, méthodes et données) qui correspondent aux besoins non-métiers.

5. Les concepts de base de la Conception Orientée Aspects

5.1. Aspect

Un aspect est une entité logicielle qui capture une fonctionnalité transversale à une application [33].

5.2. Point de Jonction (join Point)

Il s'agit de balises dans le code du programme pour l'exécution de l'aspect. Un appel de méthode, un appel de constructeur, un accès en lecture ou écriture constituent des exemples de points de jonctions.

5.3. Point de coupure (Pointcut)

C'est une construction qui permet de désigner un ensemble de points de jointure et de capturer le contexte spécifique de chacun de ces points.

5.4. Code Greffon (Advice Code)

Il s'agit du code à exécuter selon les différentes conditions contenues dans les points de coupure. Il existe 3 types de advices:

• *Before (avant)* : s'exécute avant que les points de jointure ne soient exécutés.

• Around (autour) : s'exécute autour ou avec le point de jointure et peut avoir le contrôle sur son exécution.

• After (après) : s'exécute après le point de jointure.

5.5. Tissage (Weaving)

Le tissage est le processus qui prend en entrée un ensemble d'aspects et une application de base et fournit en sortie une application dont le comportement et la structure sont étendu par les aspects [33].

6. Conclusion

L'approche orientée composant est une approche prometteuse en termes de la réutilisation des éléments logiciels. Dans ce chapitre, nous avons présenté les différents modèles à base de composants et nous avons choisi le modèle IASA pour la conception de l'architecture de notre ligne de produit eFormation.

Chapitre 3 : introduction du domaine d'application

1. Introduction

Avant de parler de l'e-formation, il faut d'abord comprendre c'est quoi une formation. Une formation est un ensemble de connaissances acquises en principe avant d'entrer dans la vie active en tant qu'élève dans une école primaire, un CEM ou un lycée, étudiant dans une université ou un stagiaire dans un centre de formation professionnelle.

L'e-formation, ou formation en ligne est une technique de formation reposant sur la mise à disposition de contenus pédagogiques via un support électronique : Cd-rom, Internet, intranet, extranet…

L'e-formation désigne les outils, les applications et l'ensemble des contenus mis à disposition d'un stagiaire dans le but d'une formation pédagogique. Longtemps réduites à des supports cd-rom, l'e-formation a évolué et utilise dorénavant le web et différentes applications. Contrairement à ce que l'on pourrait penser, la formation en ligne n'est pas synonyme d'isolement. Cette solution permet aussi de nombreuses possibilités d'échange et d'interactivité.

2. Le domaine d'application

Le domaine du e-formation est vu comme un processus d'apprentissage, où la pratique pédagogique se focalise d'avantage sur l'apprenant en mettant à sa disposition des dispositifs de formation en ligne et interactifs. La formation est adaptée aux besoins et au niveau de l'apprenant en lui proposant un environnement où il peut progresser à son rythme et bénéficier d'un suivi personnalisé.

La situation actuelles des logiciels de e-formation est principalement due à une approche très cloisonnée dans le contexte de la conception d'une application informatique. Ainsi la conception d'une gestion de scolarité pour un lycée se basera

directement sur les concepts très liés au monde du lycée. De même pour un logiciel dédié au primaire ou un logiciel dédié au LMD.

Dans la réalité ces logiciels traitent en grande partie des concepts similaires mais ayant juste des noms distincts dans chaque institution. A titre d'exemple au niveau universitaire on parle d'étudiant, au niveau primaire, moyen et secondaire nous parlons d'élève, au niveau d'une entreprise de formation nous parlons de stagiaire. En réalité les étudiants ou élève ou stagiaire sont des apprenants.

L'étude de la scolarité dans les diverses institution de formation public ou privé nous a mené à la conclusion suivante : Il y a beaucoup de similarité entre les diverses formations. La plupart des concepts sont les même et souvent ce n'est que l'appellation qui change. Cette remarque a mené à l'idée suivante : la formation peut être considérée comme un domaine d'application et les divers logiciels sont des cas particuliers d'applications de formation.

3. Le fonctionnement du système

3.1. Gestion des inscriptions

Dans cette partie, l'apprenant demande la création d'un compte en remplissant tous les champs présenter. Dans notre cas l'application permet aux utilisateurs que ce soit apprenants ou enseignants de s'inscrire sans préciser leurs rôles, le rôle affecté par défaut c'est celui d'apprenant. Pour les enseignants c'est à l'administrateur de leur changer le rôle en enseignant.

3.2. Gestion des cours

L'enseignant peut créer un cours, l'organiser en sous-section (parties, chapitres...). Poster des documents dans son cours qui peuvent être par la suite consultés ou téléchargés par les apprenants.

Une fois inscrit, un apprenant peut consulter les différentes catégories de cours. Il peut choisir le cours qui l'intéresse. Puis, pour accéder au contenu de ce cours il doit s'inscrire au cours. Les contraintes d'inscription à un cours sont définis par l'enseignant ayant créé ce cours, on peut trouver ainsi:

- Inscription libre (sans contrainte).

- L'inscription par clé : nécessite une clé d'inscription défini par l'enseignant et communiqué par celui-ci aux apprenants. Cette clé peut être modifiée par l'enseignant.

- Il n'est pas possible de s'inscrire à un cours : L'enseignant a fermé l'inscription.

Une fois inscrit, l'apprenant peut consulter ces cours ainsi qu'un ensemble d'outils (groupes, exercices, chat, forum…) la visibilité des outils est définie par l'enseignant.

3.3. Gestion des groupes

L'enseignant peut organiser les apprenants de son cours en groupes, il peut aussi leur permettre de s'inscrire dans ces groupes ou les inscrire lui -même.

3.4. Gestion des évaluations

Il y a deux types d'outils permettant l'évaluation des apprenants : les tests et les travaux:

Les travaux: L'enseignant crée, au niveau de son cours, un travail à faire par les apprenants. Le travail peut être un ensemble d'exercices à résoudre en ligne ou bien un rapport de travail à rendre.

L'enseignant peut spécifier le délai de remise des travaux et même les personnes concernés par le travail.

Les travaux peuvent être soumis par un apprenant (travaux individuel) ou un groupe d'apprenants (travaux de groupes)

Les tests: un test est un ensemble de questions. Les tests peuvent correspondre à des examens en lignes leur durée est généralement courte par rapport aux travaux (correspond à la durée d'un examen).

Les scores peuvent être calculés automatiquement par le système ou manuellement par l'enseignant.

3.5. Gestion des annonces

L'enseignant peut lancer une annonce concernant son cours pour l'ensemble des apprenants.

3.6. Gestion des délibérations

L'enseignant peut visualiser les résultats (scores) d'un test ou travail. Il peut également afficher les résultats pour être consulté par les apprenants.

4. Conclusion

De nos jours, les applications d'e-formation sont de plus en plus utilisées dans divers institutions. Ce chapitre introduit l'e-formation et donne une vue abrégée sur les fonctionnalités d'une application d'e-formation.

Partie 2 :

Conception et réalisation d'une ligne de produit pour le e-formation

Chapitre 4 : Ingénierie de domaine

1. Introduction

Comme on a déjà présenté dans le chapitre 1, l'ingénierie du domaine permet de gérer une ligne de produits logiciels dans sa globalité et non pas comme un ensemble de produits séparés où chaque produit est traité indépendamment des autres. La ligne de produits logiciels doit alors inclure les besoins spécifiés par toutes les catégories d'utilisateurs visées.

Il est possible de définir la portée de la ligne de produits logiciels c'est-à-dire l'ensemble des produits planifiés. La définition des membres de la famille de produits planifiée permet d'identifier et de planifier l'implémentation des points communs réutilisables ainsi que les points de différence entre eux. C'est ainsi que cette phase adopte la stratégie de développement pour la réutilisation. Et comme nous avons déjà dit l'objectif principal de l'ingénierie du domaine est de produire des artéfacts réutilisables comme par exemple les exigences, les composants et les classes sont construits de façon à ce qu'ils soient réutilisés dans les produits planifiés.

Pour accomplir ces objectifs l'ingénierie de domaine est composée de trois activités qui sont l'analyse de domaine, la conception du domaine et la réalisation du domaine.

Dans ce chapitre, on va appliquer ces activités dans le cadre de notre domaine d'étude l'e-formation.

2. Analyse de domaine

Le but de l'analyse du domaine est d'étudier le domaine de la ligne de produits et d'identifier les similarités et les variabilités entre les produits pour cela on va

étudier les exigences du domaine. Les exigences sont analysées pour identifier celles qui sont communes à tous les produits et celles qui sont spécifiques à des produits bien particuliers.

Tout d'abord, c'est quoi un domaine de ligne de produits ? Un domaine est un secteur de métier ou de technologies ou des connaissances caractérisées par un ensemble de concepts et de terminologies compréhensibles par les utilisateurs de ce secteur. Dans notre cas, c'est le domaine de la LdP est l'e-formation. Les applications d'e-formation dans les diverses institutions publiques ou privés est destinés aux différents niveaux primaire, moyen, secondaire et universitaire se caractérisent par plusieurs similarités et se distinguent aussi par des aspects particuliers c'est ce qu'on appelle la *variabilité* (chapitre 1 – section 3).

Pour analyser les similarités et variabilités de notre ligne de produits nous allons utiliser un **Feature Diagram**. La figure suivante représente le **Feature Diagram** de notre ligne de produit.

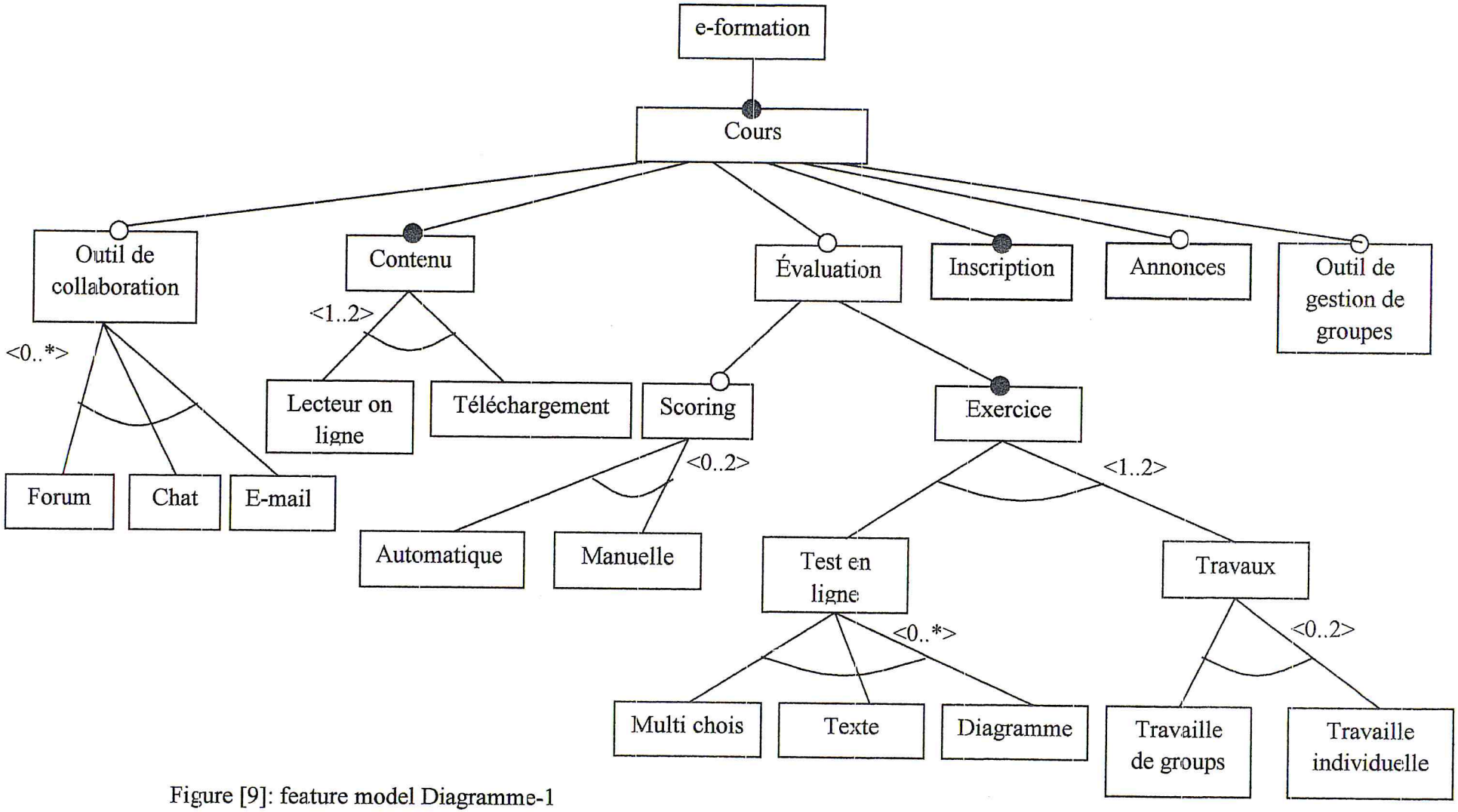


Figure [9]: feature model Diagramme-1

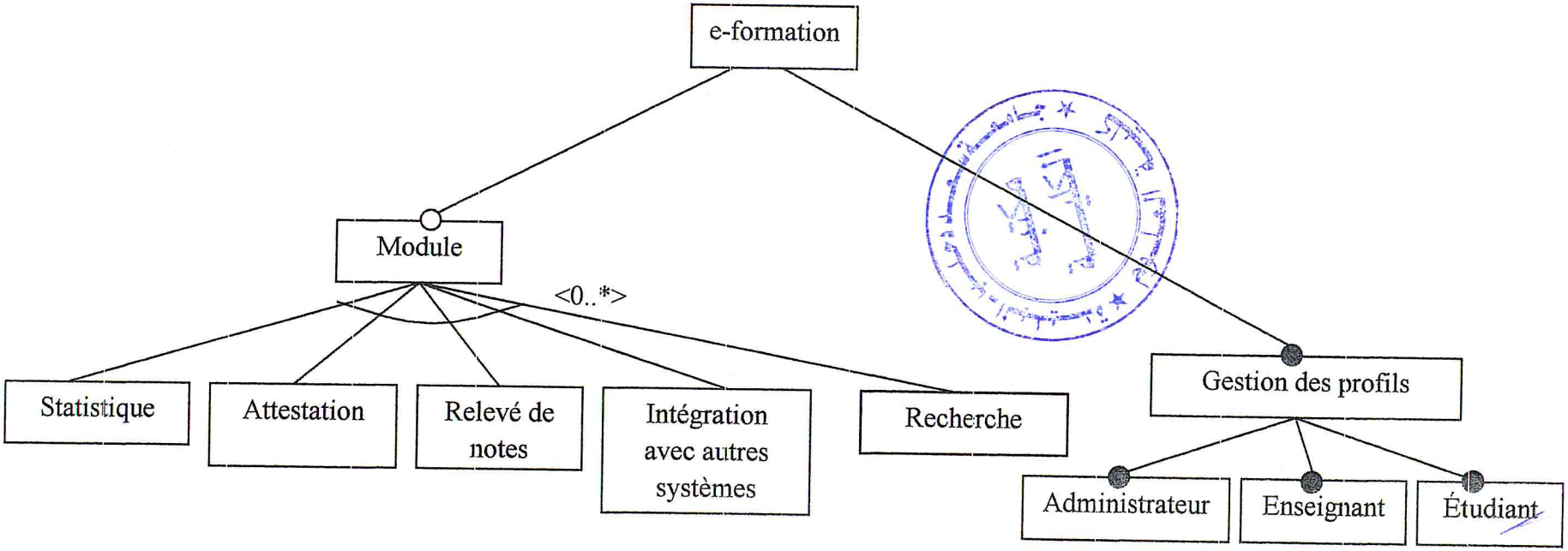


Figure [10]: feature model Diagramme-2

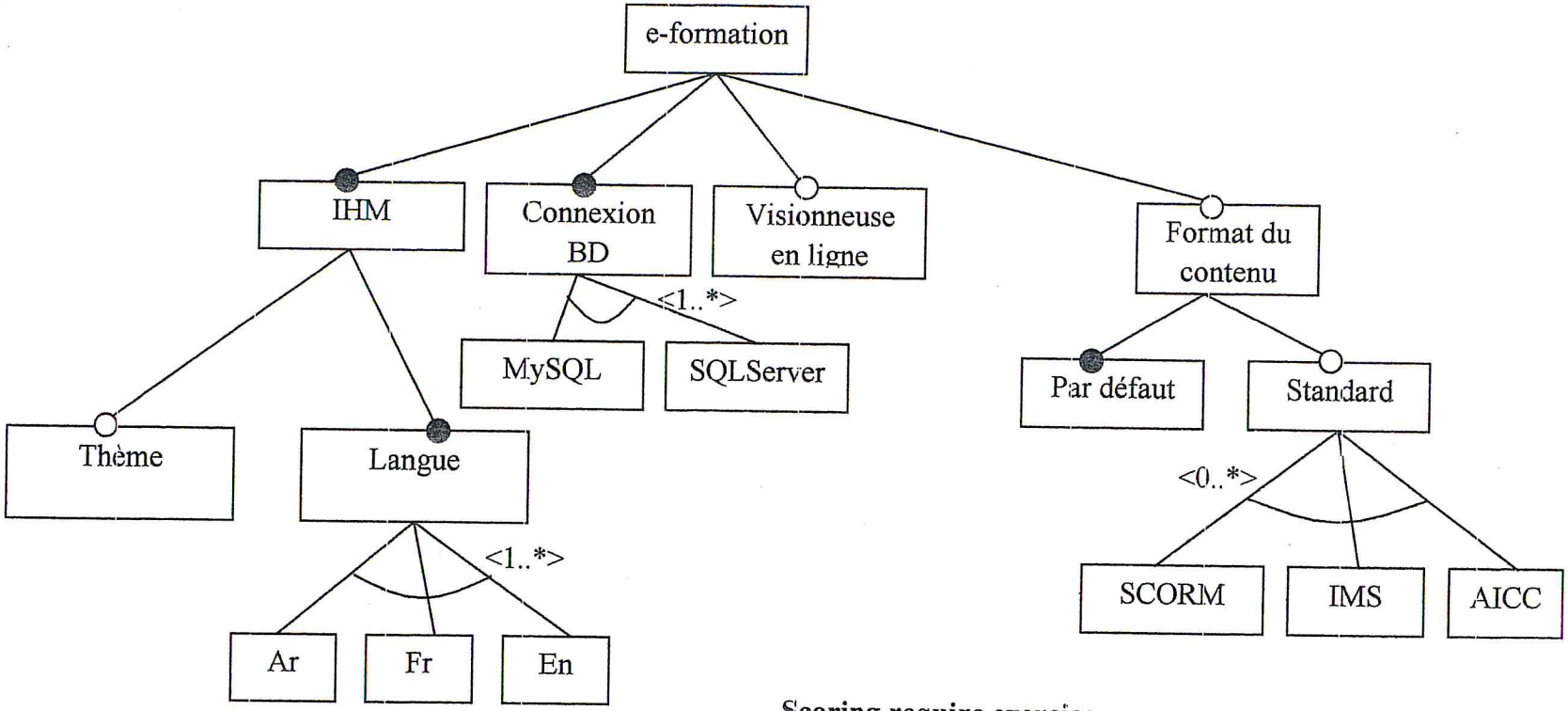


Figure [11]: feature model Diagramme-3

Scoring require exercice

Travaille de Groupe require outils de gestion de groupe

Relevé des notes require scoring

3. Conception du domaine

Le but de la conception de domaine est d'établir une architecture logicielle générique de la ligne de produits. La variabilité identifiée pendant l'analyse de domaine doit être spécifiée explicitement dans l'architecture de ligne de produits.

Dans cette partie, nous allons présenter une conception orientée composant pour la ligne de produits e-formation en utilisant la notation IASA, Nous commencerons par donner un aperçu global du système puis nous procéderons par raffinement successif.

3.1. Conception du System E-formation

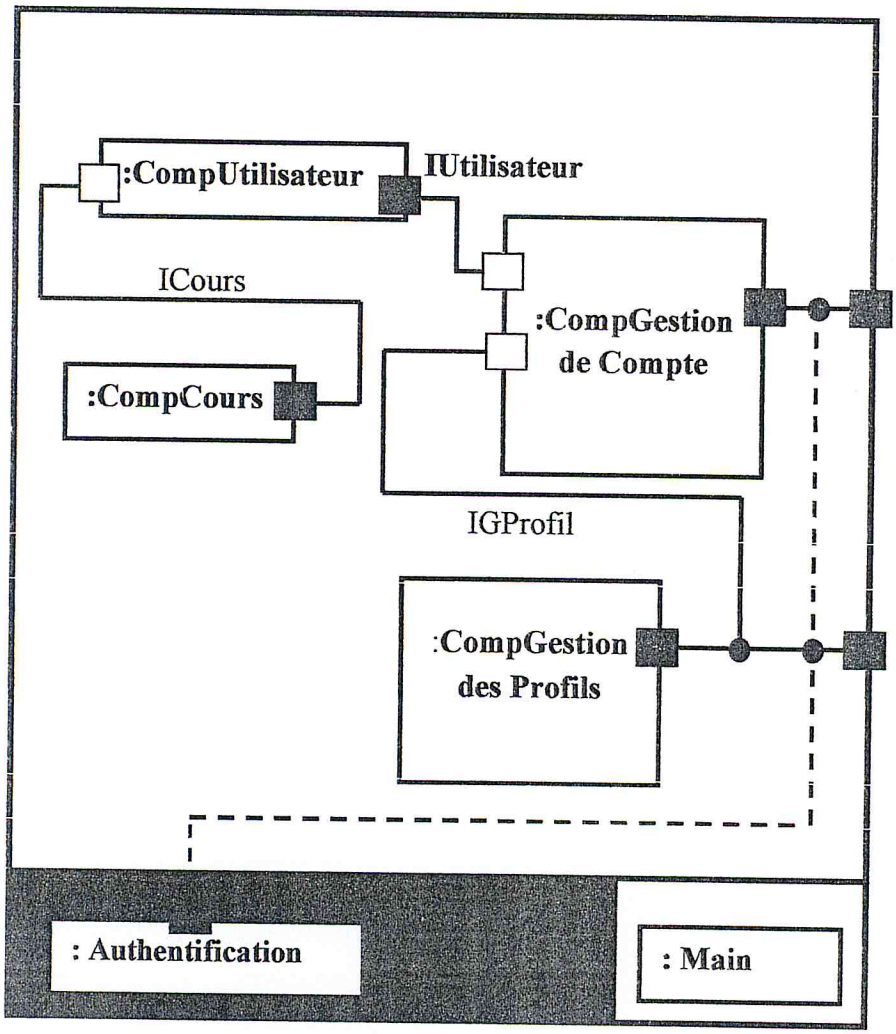


Figure [12]: Architecteur Globale E-formation

3.2. Raffinement des composants d'E-Formation

3.2.1. Composant Cours

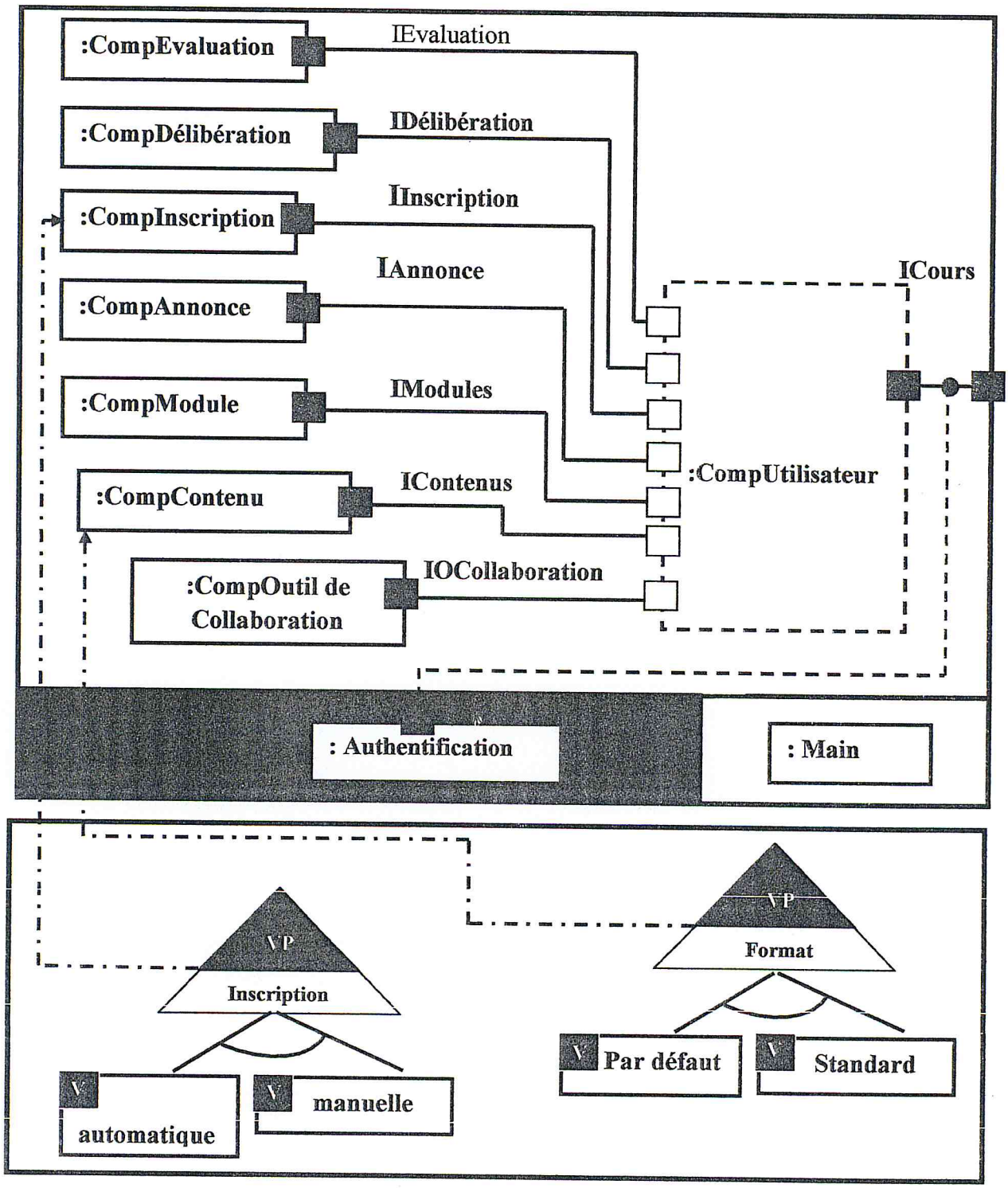


Figure [13]: Architecteur Composant Cours

Le composant Cours est un composant composite qui se charge de toutes les fonctionnalités d'e-formation tel que évaluation module délibération inscription contenu annonce.

3.2.2. Composant Gestion Profils

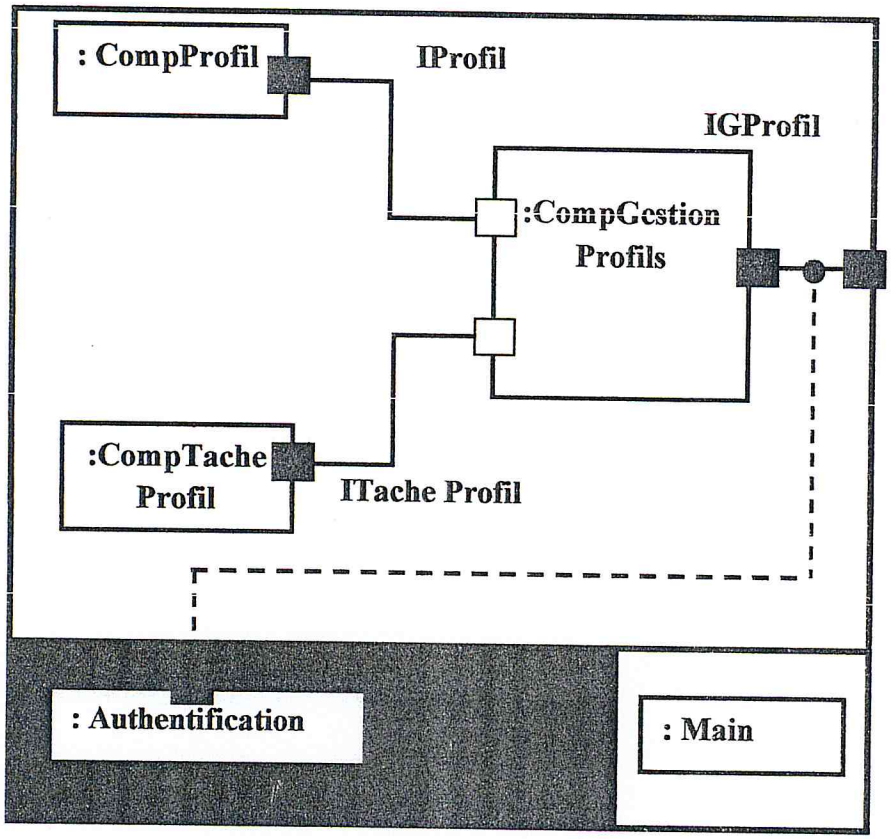


Figure [14]: Architecteur Composant Gestion Profils

Le composant gestion des profils se charge de :

1. Spécifier les tâches attribuées aux profils.
2. De créer de nouveaux profils grâce au composant CmpProfils.
3. La suppression et la modification des profils ainsi que les tâches attribuées

3.2.3. Composant Profil

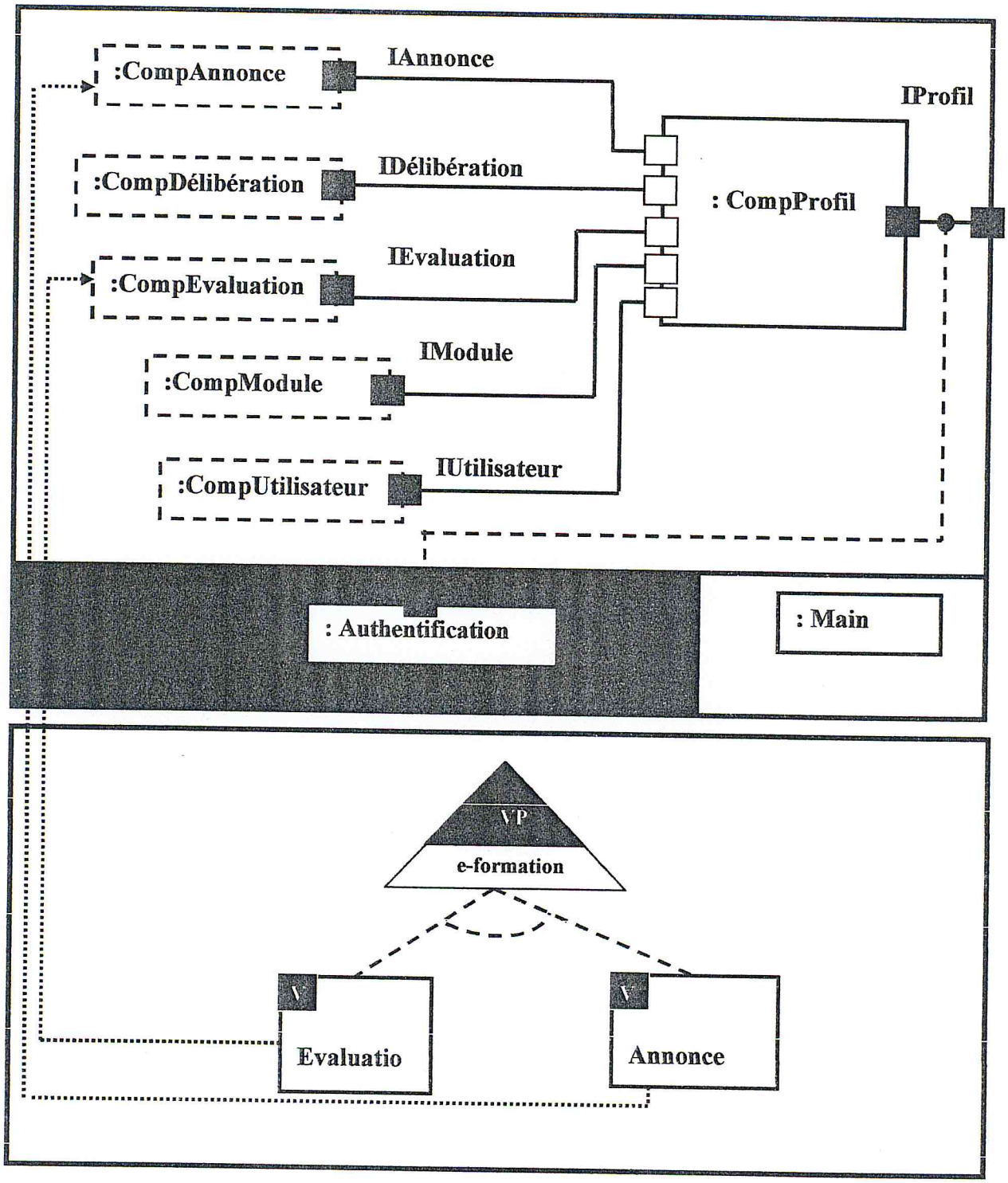


Figure [15]: Architecteur Composant Profils

Le composant Profils se charge de tout le parcours de l'utilisateur dans le system e-formation

3.2.4. Composant Evaluation

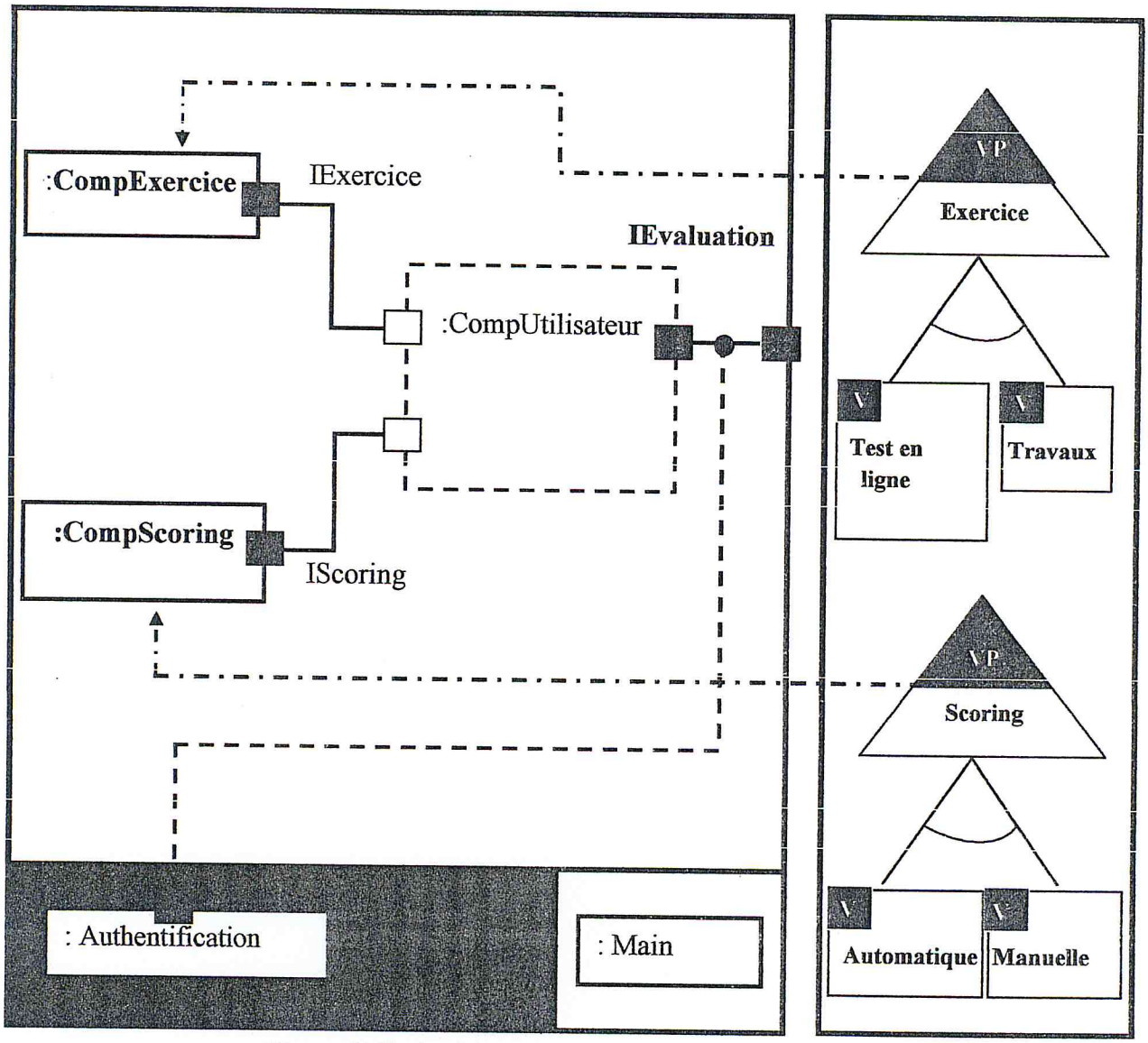


Figure [16]: Architecteur Composant Evaluation

Le composant Evaluation se charge de :

- Passer des exercices.
- Corriger des exercices

Ces tâches seront effectuées à l'aide du composant externe compUtilisateur pour le processus de passage des exercices ainsi que la sauvegarde du résultat des exercices.

3.2.5. Composant Contenu

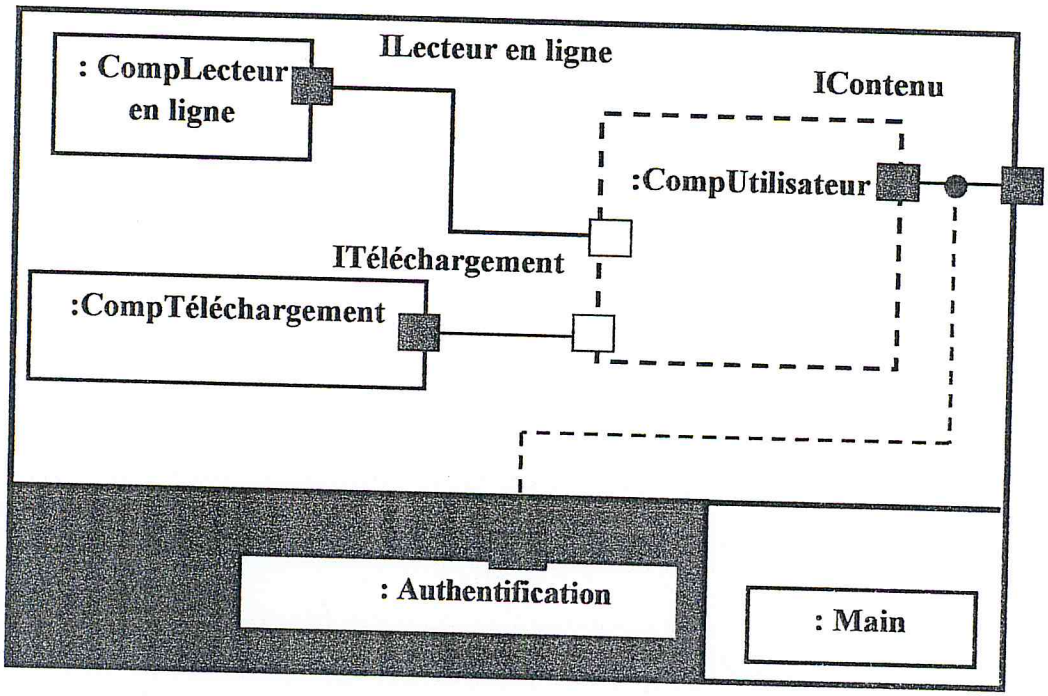


Figure [17]: Architecteur Composant Contenu

Le composant Contenu se charge des documents du cours. Ce composant permet le téléchargement et/ou la lecture en ligne, des documents.

3.2.6. Composant Module

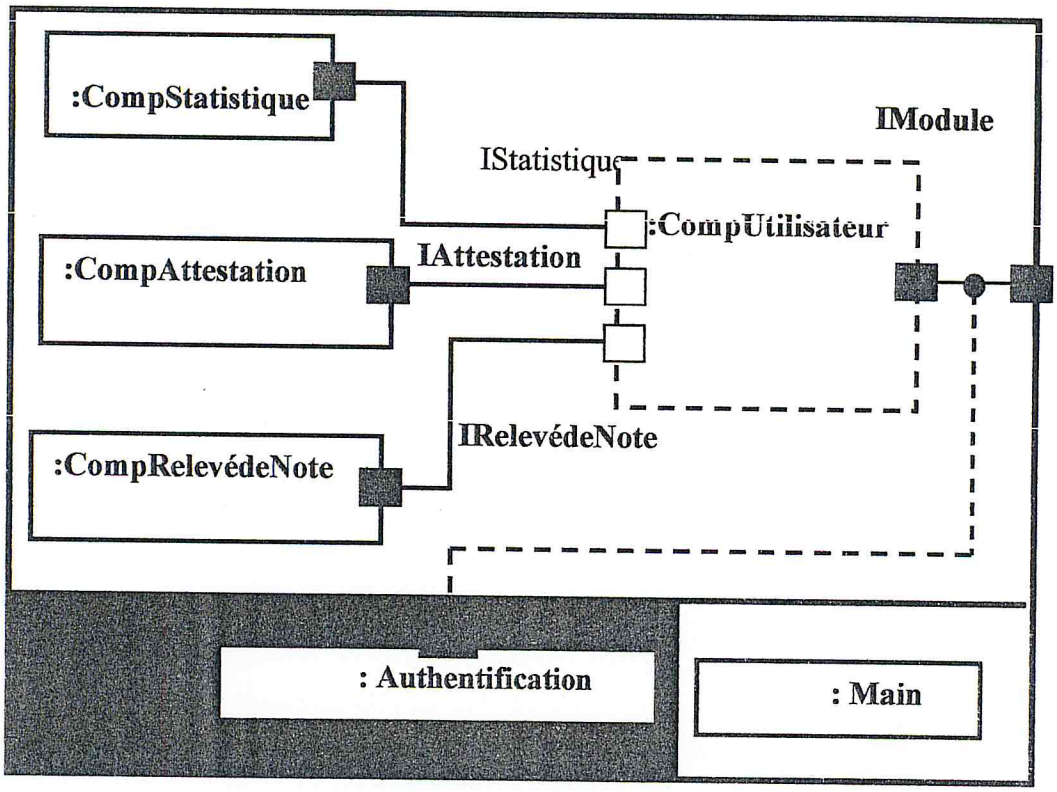


Figure [18]: Architecteur Composant Module

4. Réalisation de domaine

La réalisation du domaine consiste à implémenter les composants réutilisables. Pour notre ligne de produit d'e-formation les composants réutilisables pour tous les membres la LdP sont :

Evaluation, contenu, gestion de profile, cours, exercice, inscription, outils de collaboration, connexion BD, IHM.

5. Conclusion

Nous nous sommes intéressés dans ce chapitre, à l'application des notions de la réutilisation et de composant pour la réalisation d'une architecture de référence pour notre LdP d'e-formation. La phase d'ingénierie du domaine peut également fournir

un environnement de spécification d'architecture de produits individuel utilisable durant la phase suivante (ingénierie d'application).

Chapitre 5 : Ingénierie d'application

1. Introduction

L'ingénierie d'application consiste à utiliser les résultats de l'ingénierie de domaine pour la construction appelée aussi dérivation d'un produit particulier. Il s'agit d'un développement par la réutilisation. Les résultats de l'ingénierie de domaine (feature diagram, l'architecture des composants, et les composants) contiennent de la variabilité, la dérivation d'un produit particulier a donc besoin de décisions (ou des choix) associées à ces points de variation, pour cela nous utilisant la sélection à partir de feature model pour fixer les décisions selon un cas particulier d'application d'e-formation.

Pour montrer un processus de développement d'un membre de la LdP nous avons choisi le cas d'une application d'e-formation pour l'université parce que nous avons déjà travaillé avec ce système dans l'université.

2. Analyse d'application

Comme on dit précédemment dans cette phase, il faut sélectionner les exigences de l'application parmi les exigences du domaine...etc. Pour le cas de l'université on a proposé les exigences suivantes :

Chaque faculté contient des départements, chaque département est divisé en plusieurs branches et aussi chaque branche a plusieurs modules.

Les figures suivantes présentent le diagramme de features de notre application dérivé du feature diagram de la ligne de produit e-formation:

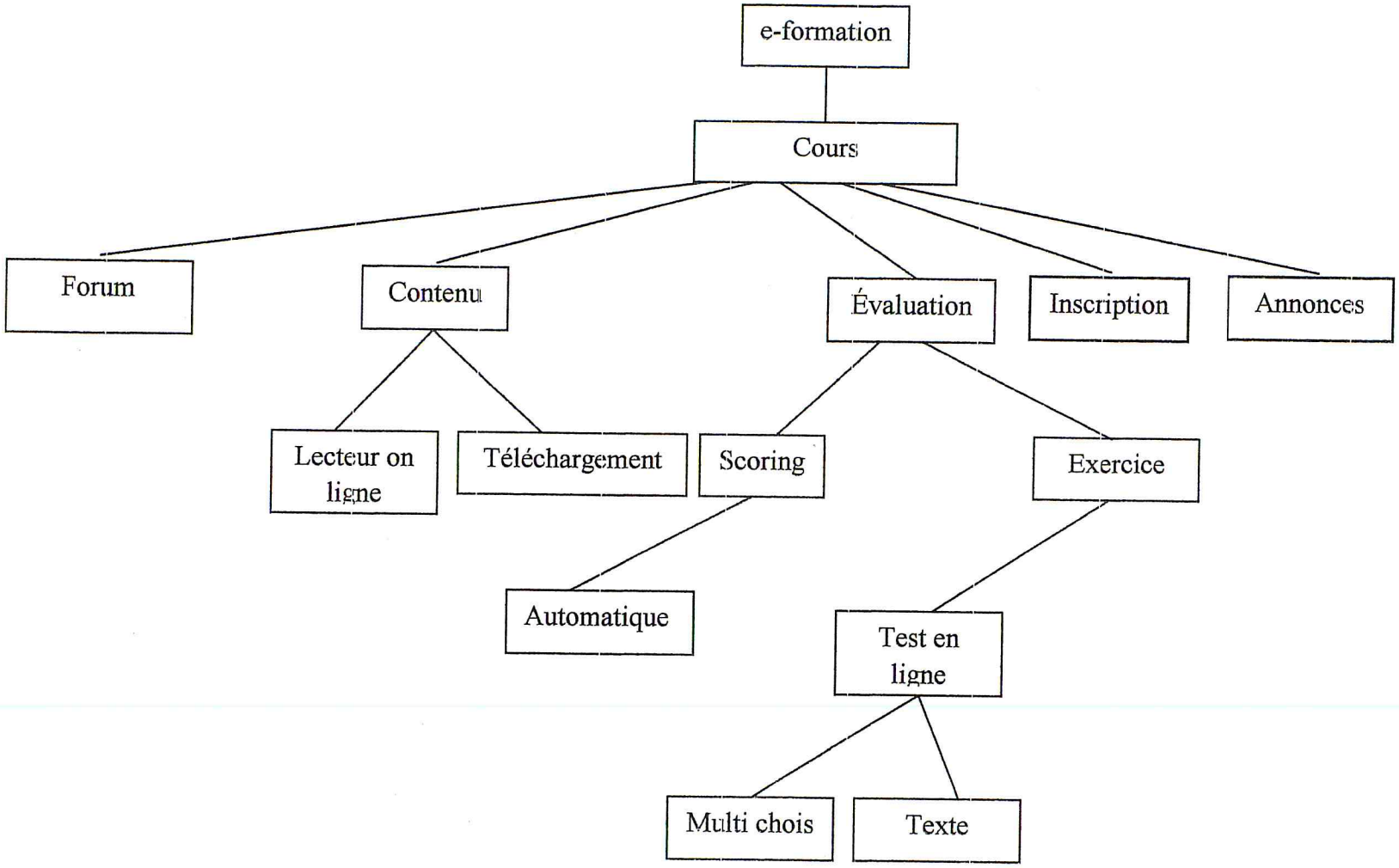


Figure [19]: feature model Diagramme pour e-formation-1

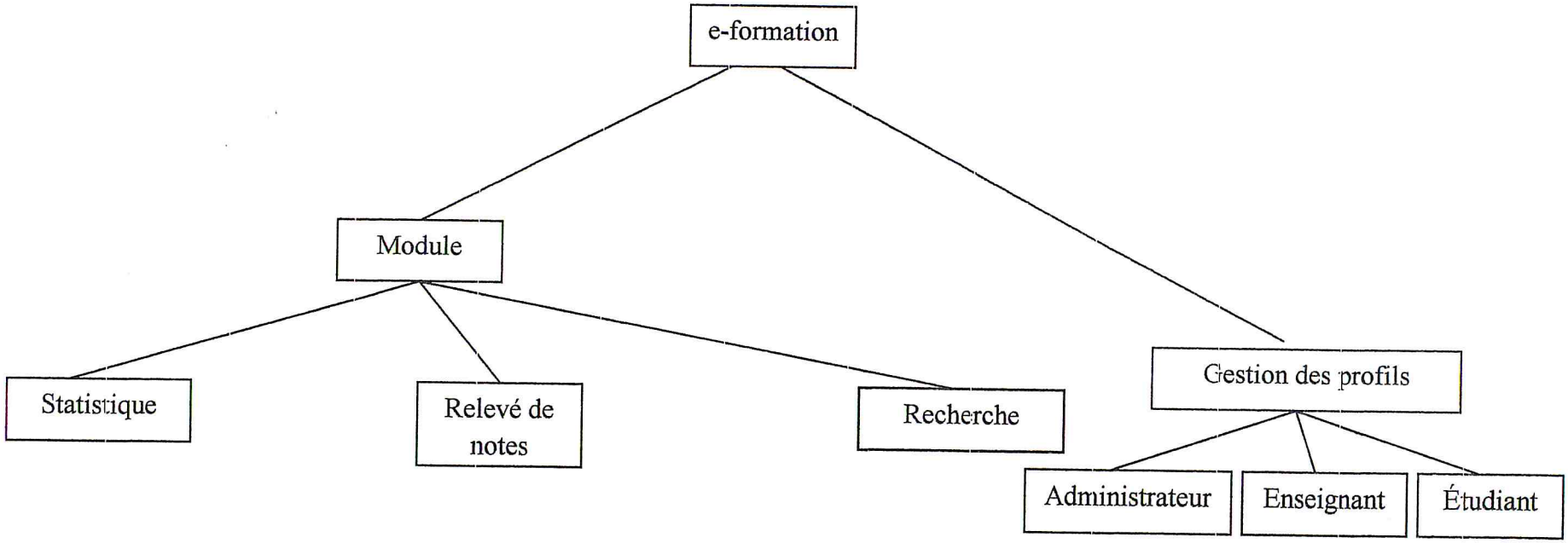
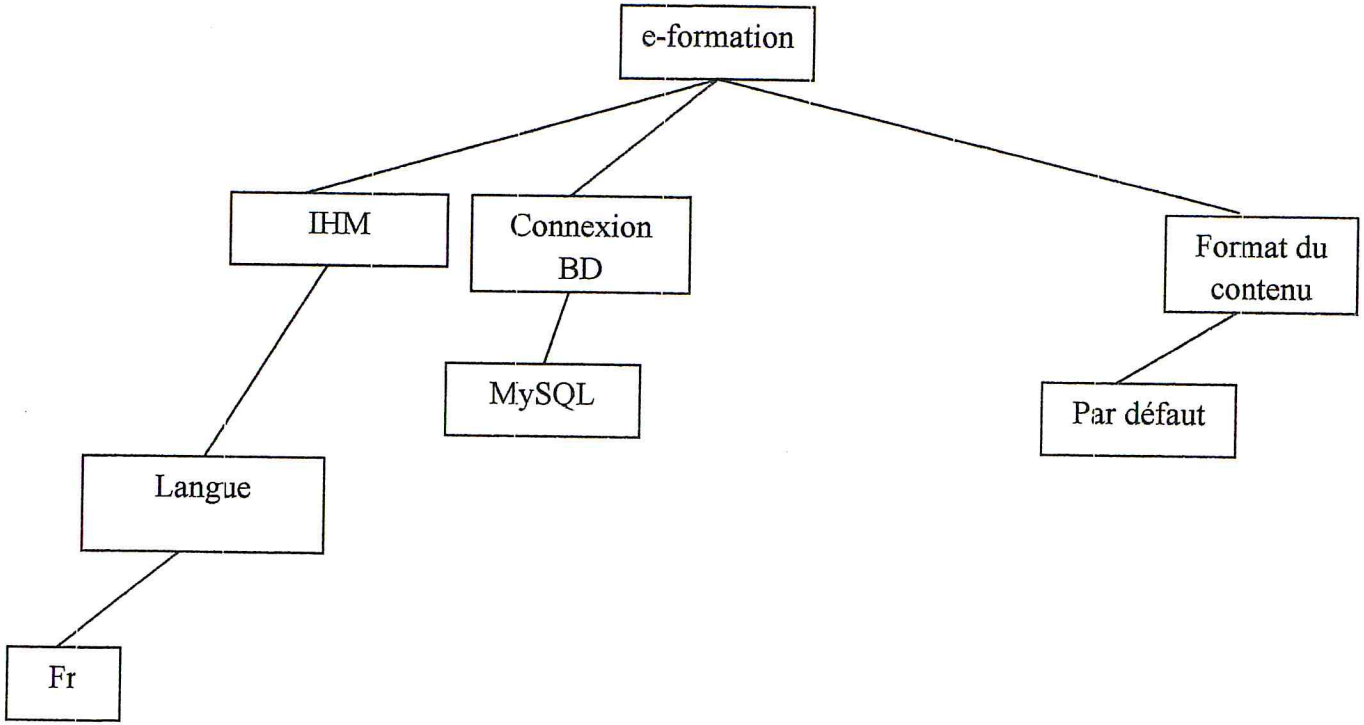


Figure [20]: feature model Diagramme pour e-formation-2



Scoring require exercice

Relevé des notes require scoring

Figure [21]: feature model Diagramme pour e-formation-3

3. Conception d'application

Dans cette partie on présente l'architecture de notre application (dérivée à partir de l'architecture de référence) :

3.1. Conception du System E-formation

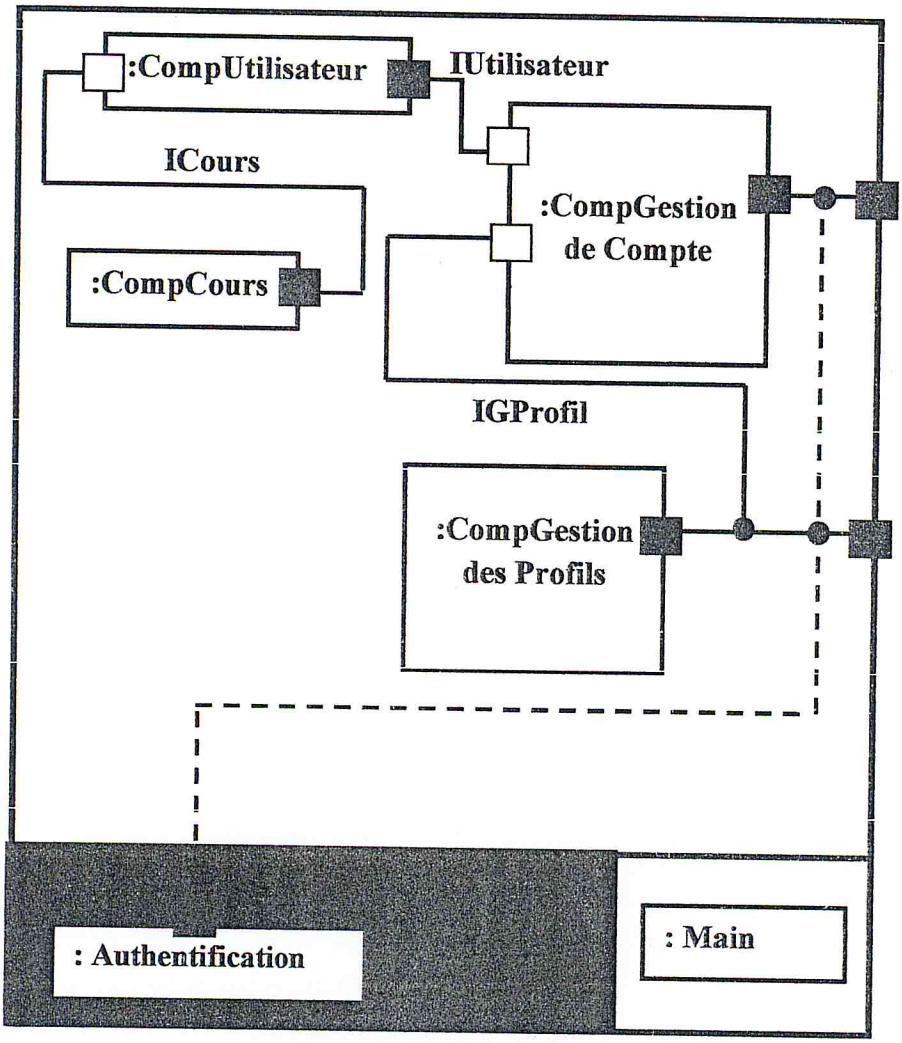


Figure [22]: Conception du System E-formation

3.2. Raffinement des composants d'E-Formation

3.2.1. Composant Cours

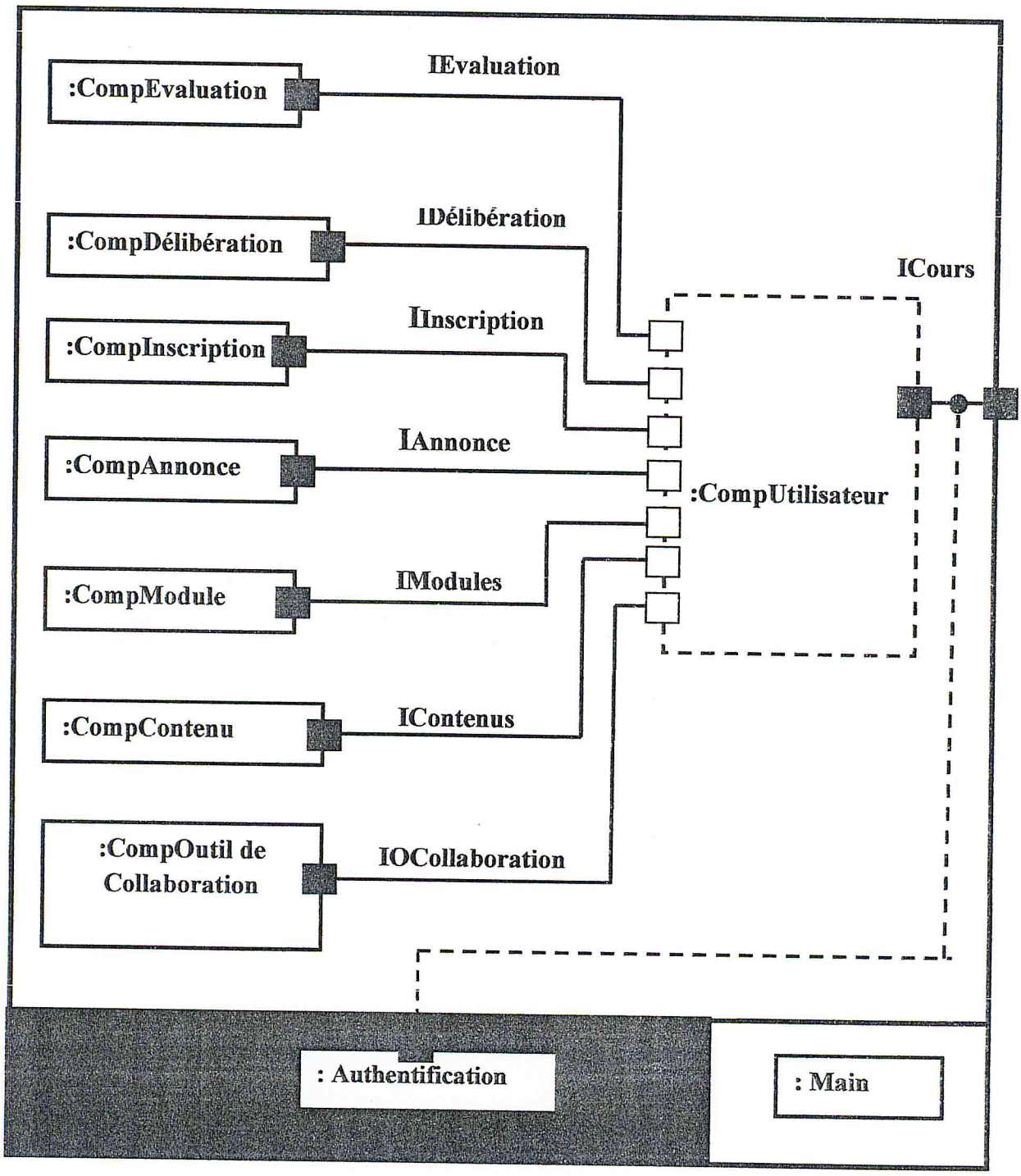


Figure [23]: Composant Cours

3.2.2. Composant GestiondeProfil

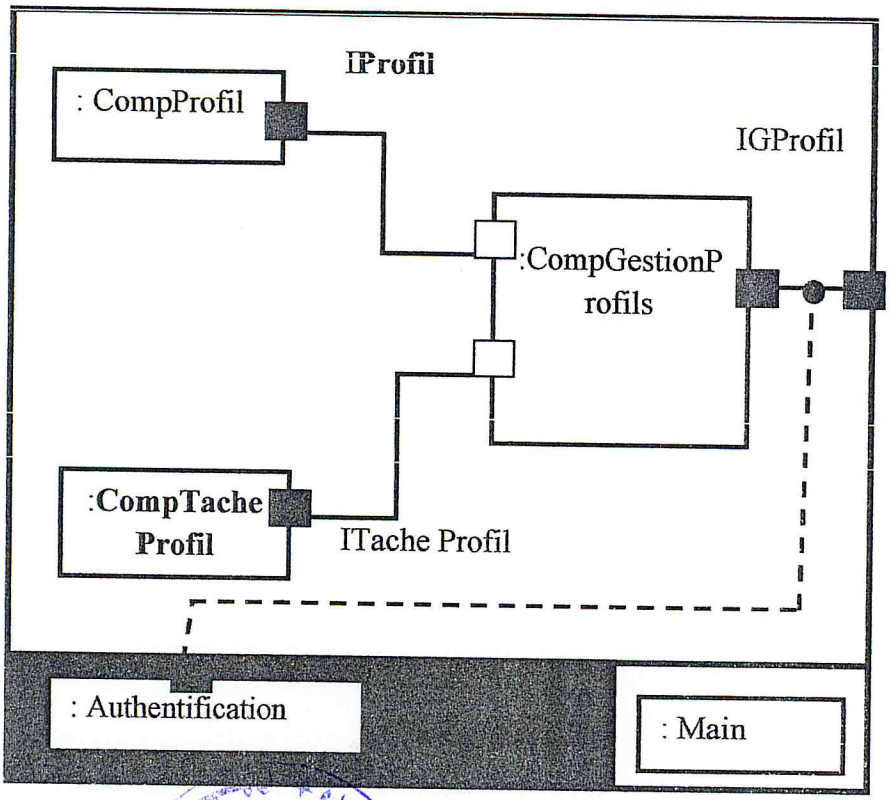


Figure [24]: Composant GestiondeProfil

3.2.3. Composant Profil

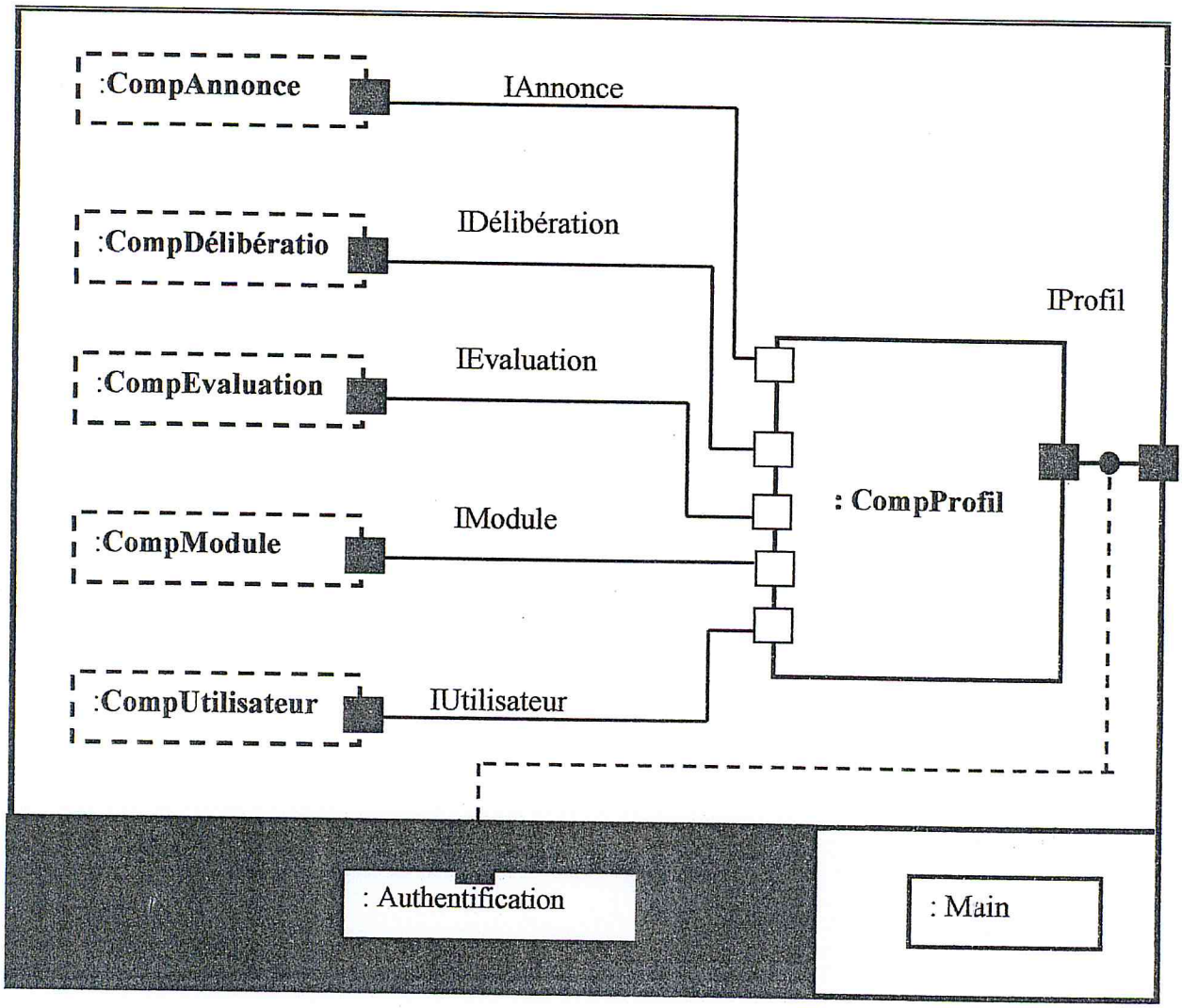


Figure [25]: Composant Profil

3.2.4. Composant Evaluation

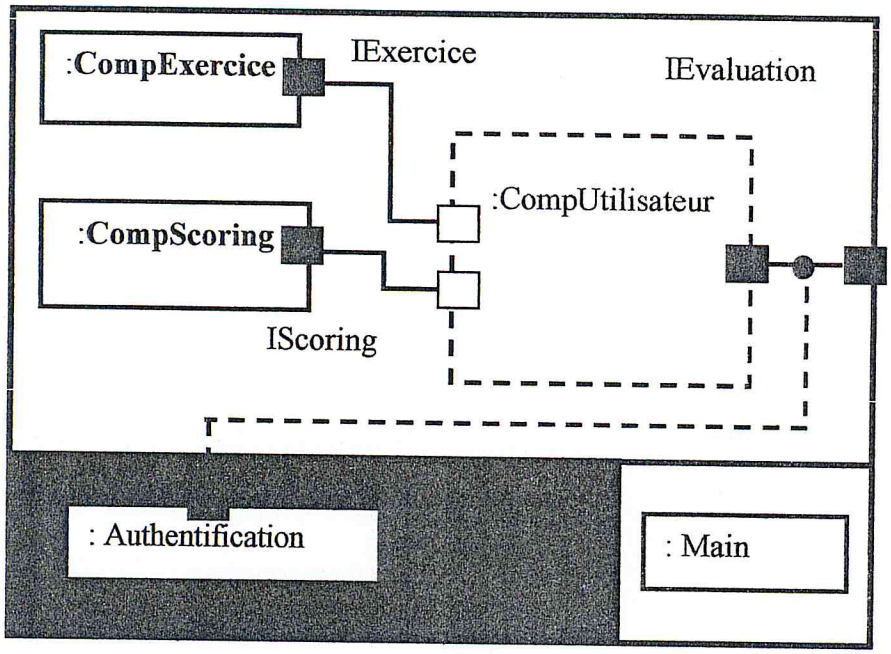


Figure [26]: Composant Evaluation

3.2.5. Composant Contenu

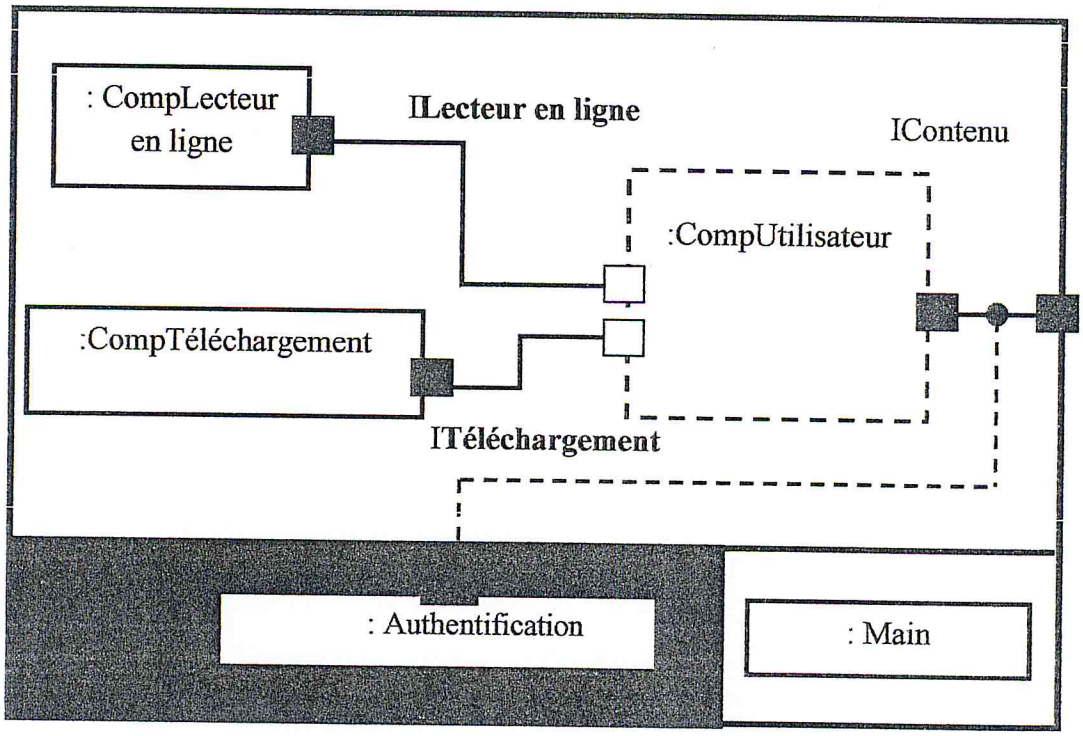


Figure [27]: Composant Contenu

3.2.6. Composant Module

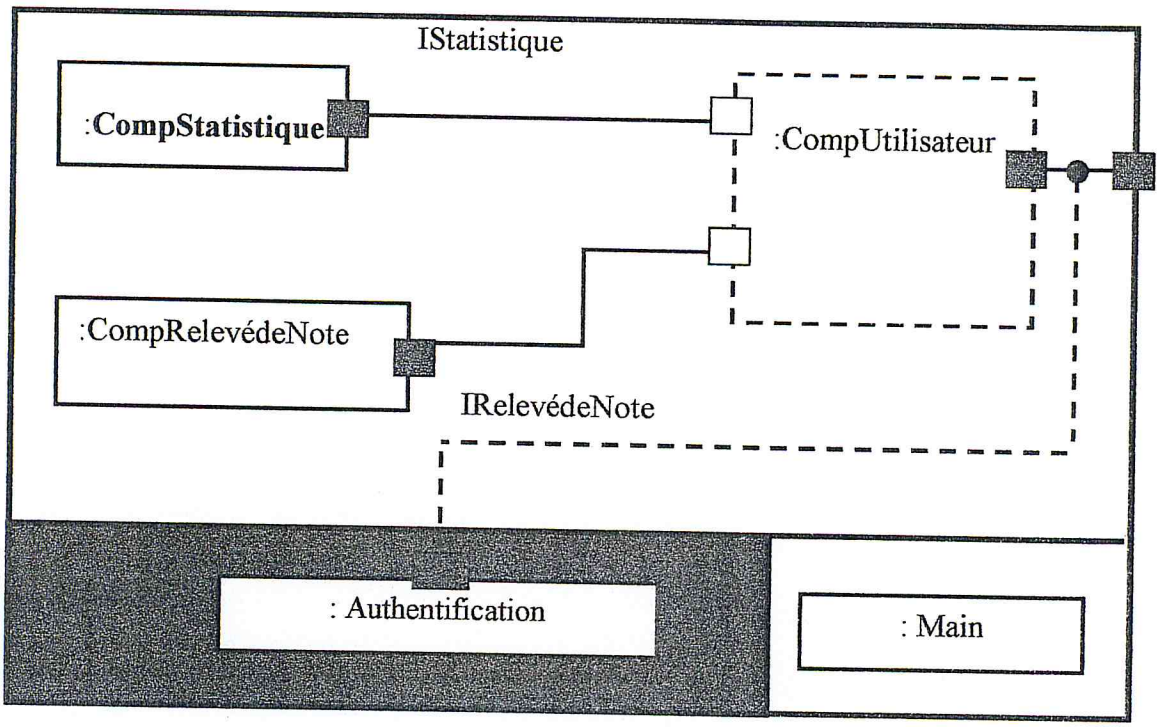


Figure [28]: Composant Module

4. Réalisation d'application

Maintenant, nous abordons la phase de la réalisation on va présenter les outils du développement et les langages de programmation utilisés. Et enfin nous présenterons quelques interfaces du travail réalisé.

4.1. Présentation des outils du développement et Les langages de programmation

Eclipse JEE est un IDE (environnement de développement intégré) puissant pour Java. Il permet au développeur de gérer ses projets avec une interface riche, il est plus rapide, plus efficace et plus fiable que d'autre Java IDE.

Nous avons travaillé avec deux langages : java pour le développement des composants et Java Server Pages (JSP) pour les interfaces (IHM). Les JSP et en générale des pages HTML contenant du code Java.

4.2. Les interfaces de l'application

4.2.1. L'interface principale

La figure suivante présente l'interface principale pour l'application e-formation Celle-ci contient plusieurs boutons d'accès comme :

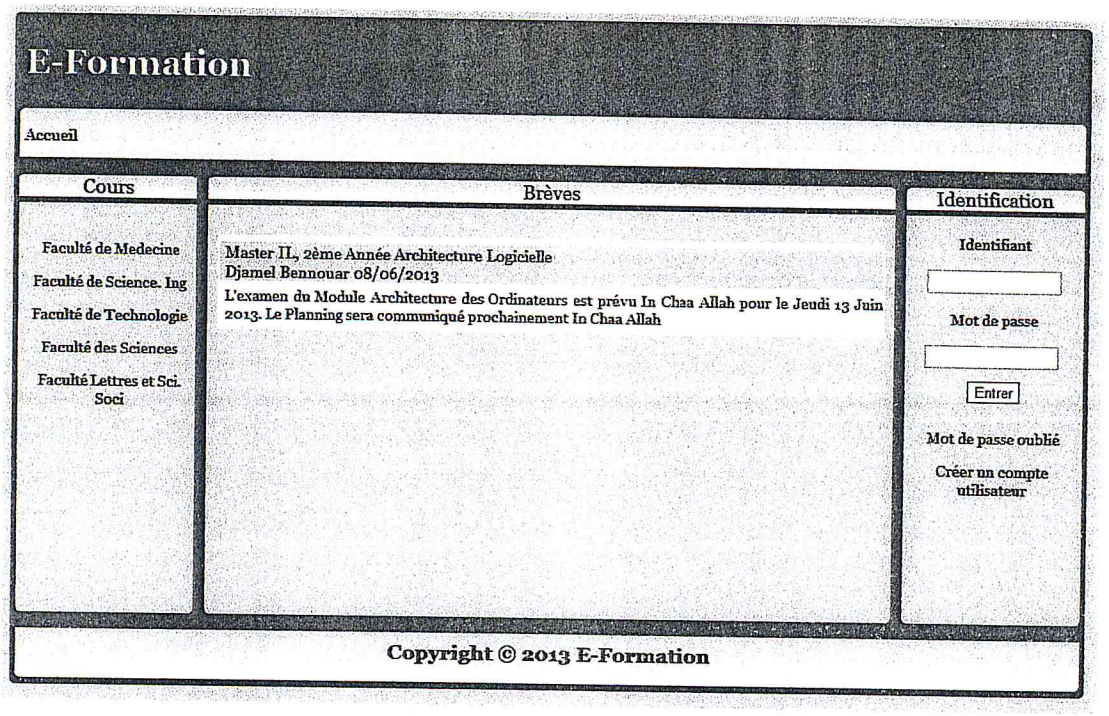


Figure [29]: L'interface principale

4.2.2. L'interface bureau d'utilisateur

Après l'authentification, l'utilisateur peut consulter son bureau présenté par une interface graphique qui contient cinq parties :

Compte utilisateur : dans cette partie on affiche les informations de l'utilisateur et la possibilité de déconnecter ou modifier ces paramètres.

Liste de mes cours : cette partie contient les cours pour lesquels l'utilisateur est inscrit.

Dernières annonces : cette partie concerne les annonces postées par d'autre utilisateur.

Statistique : c'est pour le nombre d'accès à la plateforme.

Et enfin, une partie pour la recherche.

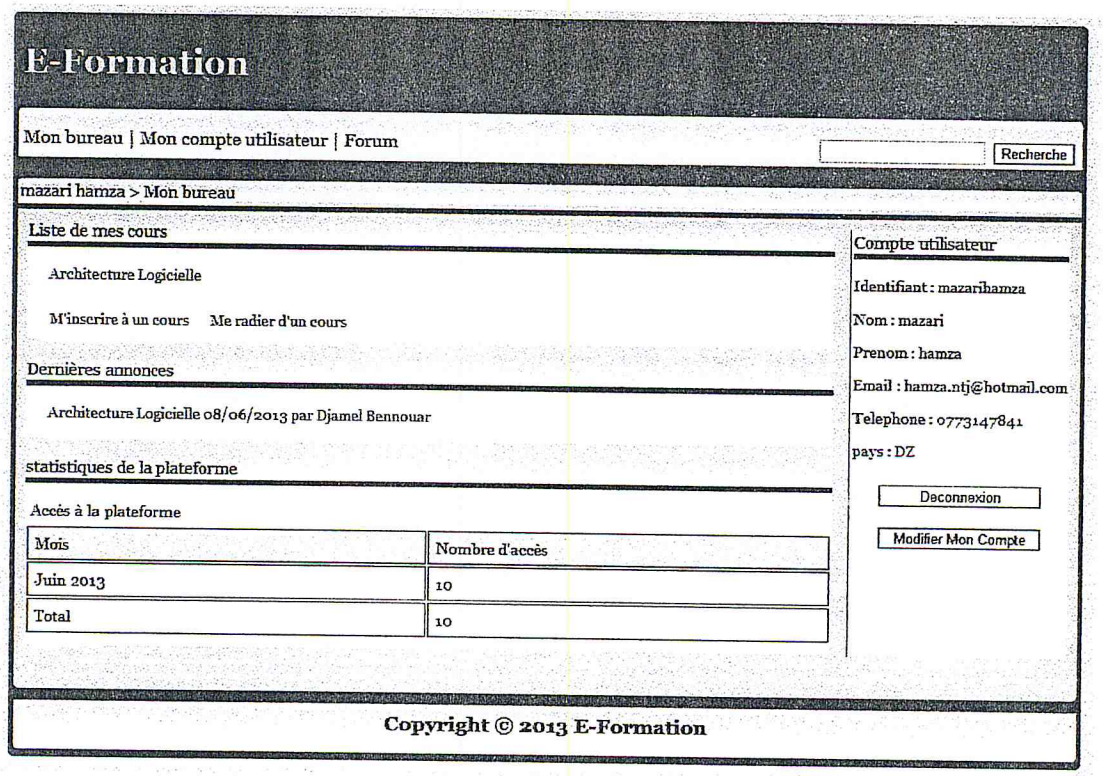


Figure [30]: L'interface bureau d'utilisateur

4.2.3. L'interface de création du compte

Cette interface a pour but de créer un nouveau compte utilisateur.

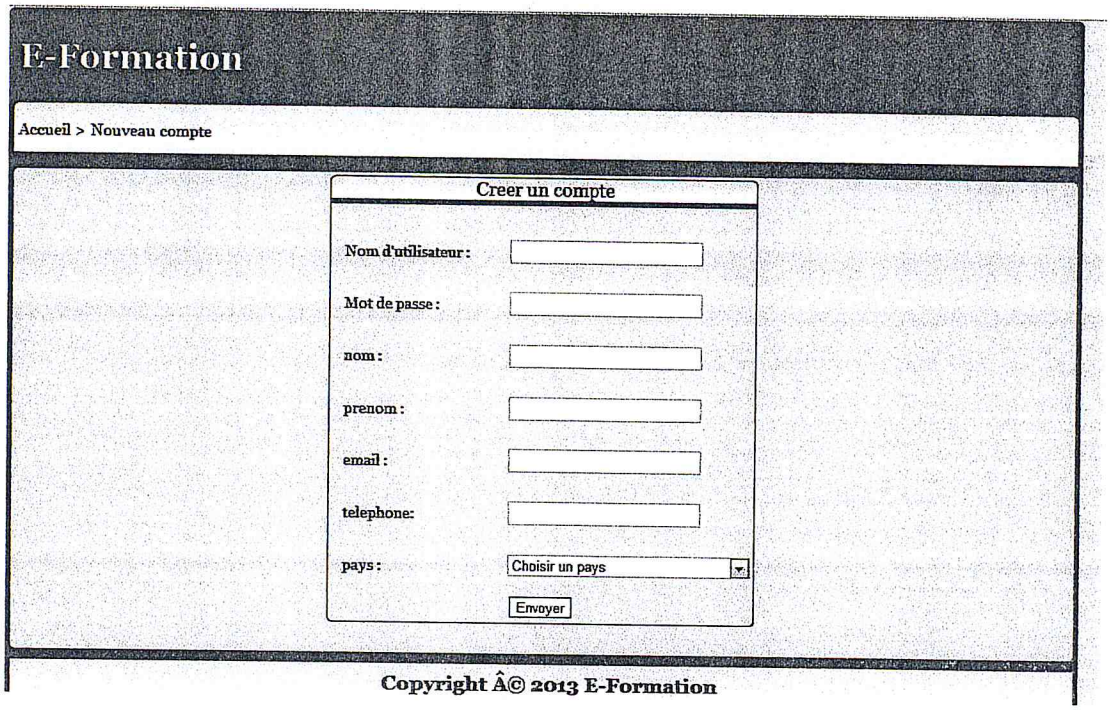


Figure [31]: L'interface de création du compte

4.2.4. L'interface de liste des cours

Cette interface affiche les cours disponible sur la plateforme et donne la possibilité à l'utilisateur de s'inscrire dans le cours qu'il a choisi.

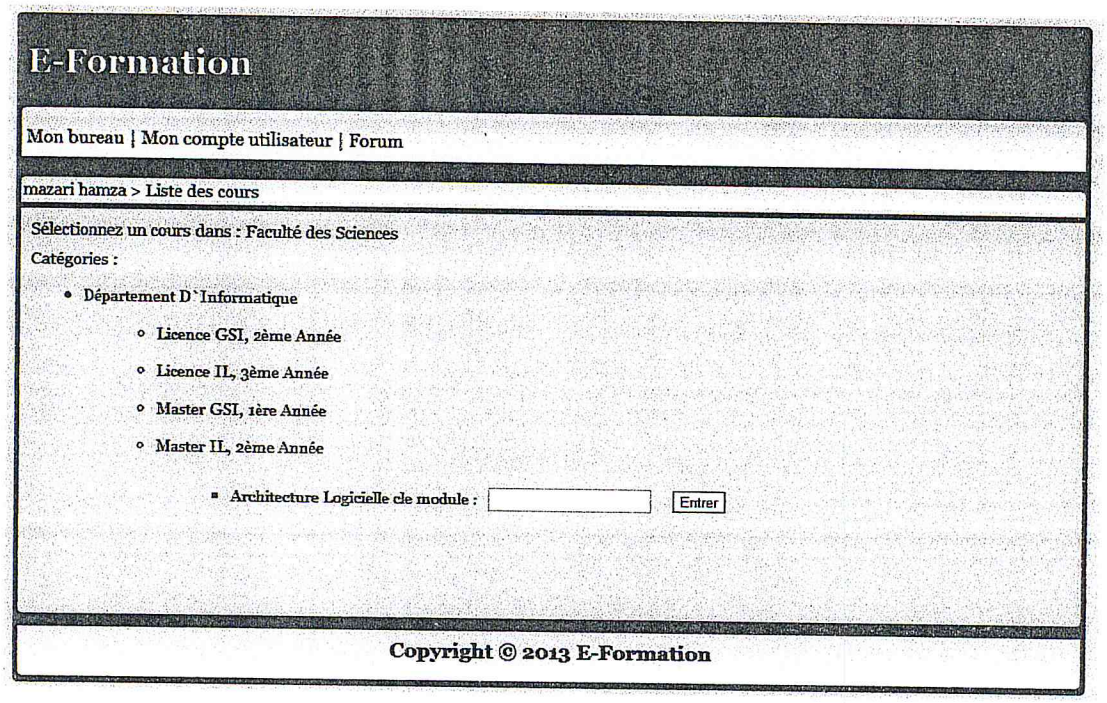


Figure [32]: L'interface de liste des cours

4.2.5. L'interface cours

Cette interface affiche tout ce qui concerne un cours spécifique : les annonces, statistique, document disponible et examen disponible.

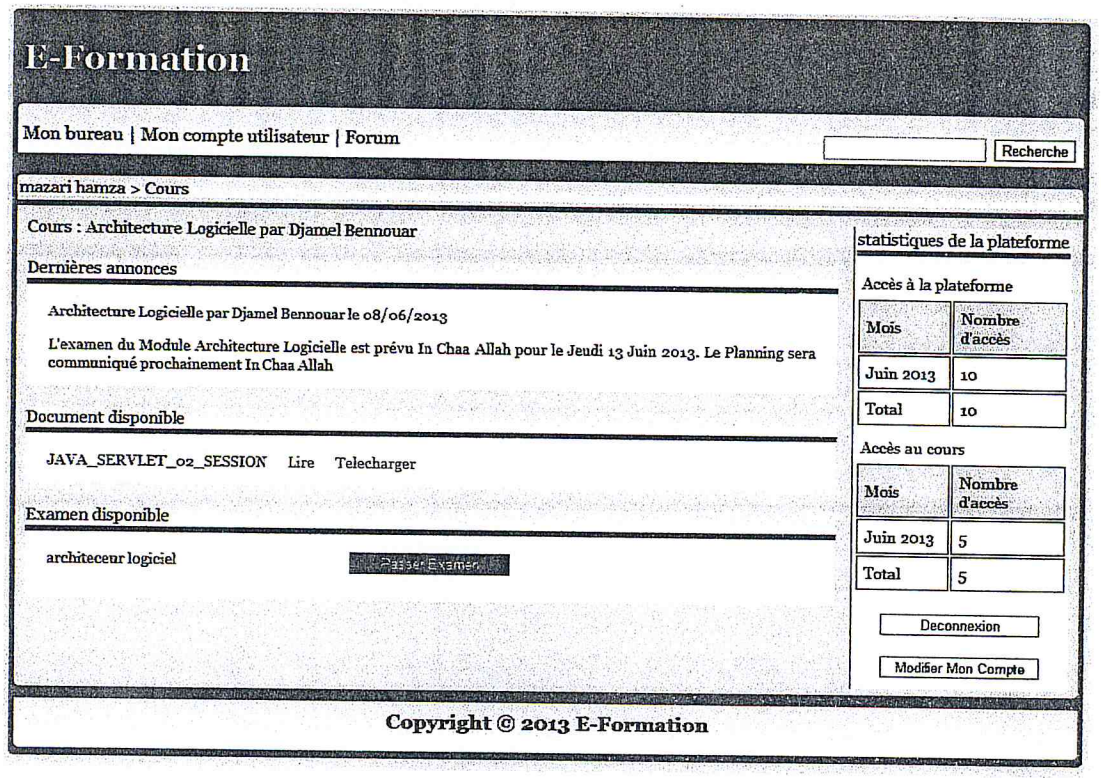


Figure [33]: L'interface cours

5. conclusion :

Dans ce chapitre, nous avons vu le deuxième processus développement d'une LdP, ingénierie d'application, ce processus se répète pour chaque nouvelle application (membre de la LdP). Dans notre projet de fin d'étude nous avons choisi le cas d'une application d'e-formation pour l'université.

Conclusion Générale

Le travail présenté dans ce mémoire est le produit d'environ quatre mois de recherche et de travail. Pendant ce temps, nous avons procédé dans deux directions : l'approche lignes de produits logiciel et l'approche orientée composant. Nous avons commencé par une étude bibliographique des deux approches dans le but de maitriser les concepts de base.

Notons que le domaine des lignes de projet est un domaine de recherche d'actualité ce qui nous a présenté un premier défi. Les travaux dans ce domaine sont rares et même lorsqu'ils existent on les trouve dans des articles de conférences qui ne dépassent pas quelques pages i.e., ils ne contiennent pas suffisamment de détail pour nous aider. Le principe des lignes de produit est lui-même un autre défi au lieu de penser à une application du domaine d'étude (e-formation) il fallait penser à tout ce qu'on peut avoir comme membre de la LdP. Et aussi le manque des outils dédiés au développement des LdPs logiciel nous a causé un vrai défi.

Afin de renforcer la réutilisation de nos éléments logiciels nous avons choisi de combiner l'approche LdP avec une approche orientée composant. Une étude bibliographique a été faite aussi pour cette approche, où nous avons choisi IASA pour la modélisation de notre architecture de référence.

Après l'étude bibliographique, nous avons entamé le processus de développement de notre ligne de produit pour le domaine d'application « e-formation ». Ce processus est divisé en deux processus : ingénierie de domaine et ingénierie d'application. Le deuxième processus a été appliqué pour le cas d'une application d'e-formation pour l'université ce processus a été simplifié par l'utilisation des résultats du premier processus.

Ce travail nous a permet d'une part, d'entamer un nouveau domaine de recherche qu'est les lignes de produit logiciel. Et d'autre part, d'approfondir et d'appliquer nos connaissances dans le développement des applications orientées composants.

Cependant, des perspectives sont envisagées notamment :

- Test des composants réutilisables on dérivants d'autres membres de la LdP.
- Implémentation des variantes qui peuvent être utilisées par d'autres membres de notre LdP.
- Le développement d'une méthode de dérivation des membres de la LdP qui pourrait être automatique ou semi- automatique.

BIBLIOGRAPHIE :

[1] C. W. Krueger, « Software reuse », ACM Computing Surveys, vol. 24, no. 2, p. 131–183, juin 1992.

[2] P. Clements et L. Northrop, Software Product Lines: Practices and Patterns, 1er éd. Addison Wesley, 2001.

[3] http://www.softwareproductlines.com/introduction/introduction.html

[4] David M. Weiss, Chi Tau Robert Lai, Software Product-Line Engineering: A Family-Based Software Development Process, Addison Wesley, 1999

[5] J. Bosch, Design & use of Software Architectures: Adopting and Evolving a Product-Line Approach, Addison-Wesley, 2000

[6] L.M. Northrop, "SEI's Software Product line Tenets", IEEE Software, Vol.19, pp.32-40, Issue 4, 2002

[7] http://www.sei.cmu.edu/productlines/

[8] IEEE recommended practice for architectural description of software intensive systems (IEEE-std-1471-2000), 2000.

[9] D.M. Weiss, "Next Generation Software Product Line Engineering", Software Product Lines, 9th International Conference, SPLC 2005, pp. 1, Rennes, France, September 26-29, 2005

[10] J. Andersson, J. Bosch, "Development and use of dynamic product-line architectures", IEEE Software, Vol.152, Issue 1, pp.15-28, 2005

[11] Ziadi, Tewfik et Jézéquel, Jean-Marc. Manipulation de Lignes de Produits Logiciels une approche dirigée par les modèles.

[12] Linden, Frank, Schmid, Klauss et Rommes, Eelco. Software Product Lines in Action. Berlin :Springer-Verlag, 2007.

[13] Millymaki, Tommi. Variability Management in Software Product Lines. Tampere, Finland : Tampere University of Technology, 2001.

[14] Jean-Christophe. Les lignes de produits logiciels. 2009.

[15] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S., Nov. 1990.Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. Rep. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University.

[16] Groebbens Adelbert, Oktober 2007,Productiviteit in Software Engineering.

[17] Pohl, K., Bockle, G., van der Linden, F., July 2005. Software Product Line Engineering: Foundations, Principles and Techniques. Springer.

[18] Ziadi, Tewfik. «Manipulation de Lignes de Produits en UML», Thèse de doctorat, université de Rennes 1,2004.

[19] JIANQI YU. «LIGNE DE PRODUITS DYNAMIQUE POUR LES APPLICATIONS A SERVICES», Thèse de doctorat, université de JOSEPH FOURIER, 2010.

[20] Klaus Pohl, Günter Bockle, Frank van der Linden, Software Product Line Engineering Foundations Principles and Techniques, 2005.

[21] L . Parnas. On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12) :1053–1058, 1972.

[22] Szypersky,D. Gruntz, and S.Murer. Component Software. Beyong Object-Oriented Programming. ACM Press, 2002.

[23] Bruno Bouyssounouse and Joseph Sifakis, editors. The ARTIST Roadmap for Research and Development, volume Vol. 3436. Lecture Notes in Computer Science – Springer, 2005.

[24] Dalila GUESSOUM, « SPECIFICATION HAUTEMENT FLEXIBLE D'ARCHITECTURE LOGICIELLE », MEMOIRE DE MAGISTER, UNIVERSITE SAAD DAHLEB DE BLIDA, 2011.

[25] M. Shaw et P. Clements. The Golden Age of Software Architecture. IEEE Software, 23(2):31–39. IEEE Computer Society Press, 2006.

[26] Adel Smeda, Mourad Oussalah, and Tahar Khammaci. A multi-paradigm approach to describe complex software system, WSEAS Transactions on Computers, 3(4) :936–941, October 2003.

[27] Reiko Heckel, Alexey Cherchago, and Marc Lohmann, "A formal approach to service specification and matching based on graph transformation", Electronic Notes in Theoretical Computer Science, 105 :37–49, 2004.

[28] Cyril Carrez, « Contrats Comportementaux pour Composants », Phd thesis, l'école Nationale Supérieure des Télécommunications, Spécialité : Informatique et Réseaux, 2003.

[29] Virginia C. Carneiro De Paula, George R. Ribeiro-Justo, and P.R.F.Cunha, "Specifying and verifying reconfigurable software architectures", In PDSE, pages 21–31, 2000.

[30] Sylvain Chardigny, « Extraction d'une architecture logicielle à base de composants depuis un système orienté objet Une approche par exploration », Thèse de doctorat, université de Nantes, 2010.

[31] Barais, O. Construire et Maîtriser l'Evolution d'une Architecture Logicielle à base de Composants, thèse de doctorat à l'université des Sciences et Technologies de Lille, 2005.

[32] T. Kalibera and P. Tuma. Distributed component system based on architecture description : The sofa experience. In On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002, pages 981–994, London, UK, octobre 2002. Springer-Verlag. ISBN : 3-540-00106-9.

[33] Abdelhakim BAOUYA et Mohamed MAHDJOUB, «Conception Orientée Aspect et par composants d'une application d'E-Gouvernement», MEMOIRE DE MASTER, UNIVERSITE SAAD DAHLEB DE BLIDA, 2011.

[34] D.Bennouar, A.Henni, A.SaadiTHE DESIGN OF A COMPLEX SOFTWARE SYSTEM USING A SOFTWARE ARCHITECTURE APPROACH,2009.

[35] FRANCIS TESSIER, ASSURANCE QUALITÉ ET DÉVELOPPEMENT DE LOGICIELS ORIENTÉS ASPECT: DÉTECTION DE CONFLITS ENTRE LES ASPECTS BASÉE SUR LES MODÈLES, L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES, 2005