UNIVERSITÉ SAAD DAHLEB DE BLIDA

Faculté des sciences

Département d'informatique

MÉMOIRE DE MASTER

En Informatique

Option : Ingénierie du Logiciel

THÈME :

Dérivation automatique des applications e-banking dans une ligne de produits logiciels guidée par les ontologies

Réalisé par

BETTAHAR SAMIR

Encadré par

Mme. Lahiani Nesrine

September 2022

Remerciements

Je tiens tout d'abord à remercier Dieu le Tout-Puissant, qui Il m'a donné la force et la patience d'accomplir ce travail dans les meilleures conditions.

Je voudrais exprimer ma gratitude à Madame Lahiani Nesrine , je la remercie de sa présence Sous la direction, l'orientation, l'assistance et les conseils pendant faire ce travail.

Je tiens également à exprimer une reconnaissance aux membres du jury pour avoir accepté d'examiner et de porter leur jugement sur mon travail.

Mes sincères remerciements à ma famille pour m'avoir aidé à surmonter tous les obstacles et les difficultés durant cette période de travail.

' Merci à Tous.'

Résumé

La réutilisation de logiciels est un problème fondamental et récurrent depuis les origines du génie logiciel. Son objectif principal est de réduire les coûts de développement et de la maintenance logiciel en facilitant la réutilisation d'éléments logiciels préexistants. Actuellement, l'approche des lignes de produits logiciels et de l'architecture logicielle permet de réaliser conjointement ces objectifs.

Les objectifs de notre projet de recherche final comprennent le développement d'une ligne de produits de l'e-banking et l'identification de composants réutilisables, cette ligne permet plus tard de générer diverses applications en fonction des besoins des utilisateurs.

la construction d'une ontologie de domaine qui est utilisé pour représenter le feature model de l'e-banking qui détermine les fonctionnalités (features) communes et les variabilités des logiciels de ce domaine. L'architecture est modélisée selon des principes orientés composants pour améliorer la réutilisation.

Mots clés : Ligne de produit logiciel, Composant, variabilité, e-Banking, feature model, Réutilisation.

Abstract

Software reuse is a fundamental and recurring problem since the origins of software engineering. Its main objective is to reduce the costs of software development and maintenance by facilitating the reuse of pre-existing software elements. Currently, the software product line and software architecture approach allows to jointly achieve these objectives.

The objectives of our final research project include the development of an e-banking product line and the identification of reusable components, which later allow the generation of various applications according to user needs.

The construction of a domain ontology that is used to represent the e-banking feature model that determines the common functionalities and variabilities of the software in this domain. The architecture is modeled according to component-oriented principles to improve reusability.

Keywords : Software product line, Component, variability, e-Banking, feature model, Reuse.

ملخص

إعادة استخدام البرمجيات هي مشكلة أساسية ومتكررة منذ أصول هندسة البرمجيات. ويتمثل هدفه الرئيسي في خفض تكاليف تطوير وصيانة البرامجيات عن طريق تيسير إعادة استخدام عناصر البرامجيات الموجودة من قبل. في الوقت الحالي، يسمح خط إنتاج البرمجيات ونهج بنية البرمجيات بتحقيق هذه الأهداف بشكل مشترك.

تشمل أهداف مشروعنا البحثي النهائي تطوير خط إنتاج للخدمات المصرفية الإلكترونية وتحديد المكونات القابلة لإعادة الاستخدام، والتي تسمح لاحقًا بتوليد تطبيقات مختلفة وفقًا لاحتياجات المستخدم.

إنشاء أنطولوجيا المجال التي تستخدم لتمثيل نموذج ميزة الخدمات المصرفية الإلكترونية الذي يحدد الوظائف والتغيرات المشتركة للبرمجيات في هذا المجال. تم تصميم البنية وفقًا للمبادئ الموجهة نحو المكونات لتحسين قابلية إعادة الاستخدام.

الكلمات المفتاحية

خط إنتاج البرمجيات ، المكون ، التباين ، الخدمات المصرفية الإلكترونية ، نموذج الميزة ، إعادة الاستخدام.

Table des matières

Table des figures

Liste des tableaux

Liste des acronymes et abréviations

CSS *Cascading Style Sheets*

e-banking *Electronic Banking*

FODA *Feature-Oriented Domain Analysis*

GAB *Guichet Automatique Bancaire*

HTML *HyperText MarkUp Language*

IBM *International Business Machines*

Jakarta EE *Jakarta Enterprise Edition*

Java EE *Java Enterprise Edition*

Java SE *Java Standard Edition*

LDP *Lignes de produits logiciels*

OWL *Web Ontology Language*

OWL DL *OWL Description Logics*

RDF *Ressource Description Framework*

RDFS *Ressource Description Framework Schema*

SEI *Software Engineering Institute*

SQL *Structured Query Language*

W3C *World Wide Web Consortium*

XML *Extensible Markup Language*

Introduction Générale

Contexte de travail

Pour les développeurs des systèmes informatiques d'aujourd'hui, la complexité est un enjeu important à gérer pendant tout le cycle de développement logiciel. Des approches telles que les lignes de produits logiciels apportent des solutions aux problèmes de gestion de la complexité, de réduction des coûts et d'amélioration de la qualité du logiciel final basé sur la réutilisation.

La réutilisation devient de plus en plus une méthode et une solution technique pour le développement de systèmes logiciels complexes. L'un des principaux défis auxquels sont confrontés la communauté de la recherche et l'industrie est de trouver les bons concepts, mécanismes et langages pour une meilleure réutilisation et configuration des systèmes logiciels à grande échelle.

Par ailleurs, le concept de ligne de produits suggère une vision du développement et de modélisation où l'objectif n'est pas d'obtenir un seul système, mais plusieurs systèmes possèdent des fonctionnalités communes.

Electronic Banking (e-banking) représente un nouveau mode d'opération crée par la transition des banques vers un approche qui permet de profiter des technologies de communications afin de faire face à l'accélération du rythme de changement de l'environnement commercial [1].

Notre travail consiste à mettre sur pied une *Lignes de produits logiciels* (LDP) pour l'e-banking. Dans le but de développée non seulement un logiciels pour satisfaire les besoins des banques et ces clients mais un ensemble des logiciels qui partagent des fonctionnalités communes, afin de simplifier le processus de développements, par la génération des applications différentes, cette génération est guidée par les ontologies.

Problématique

Les services bancaires connaissent une utilisation croissante des technologies d'information et de communication, dans le but de produire des services plus efficaces aux clients de la banque toute en gagnant en termes de temps et de coût. Dans notre travail de recherche, la problématique n'est plus de développer un seul logiciel à la fois mais plutôt de développer une LDP pour répondre plus rapidement aux demandes croissantes.

Aujourd'hui, la majorité des approches existantes utilisées dans la dérivation des LDP se sont concentrées sur le mappage de modèles de fonctionnalités (Feature model) à des modèles de modèles afin de réaliser la variabilité au niveau de l'implémentation. Ainsi, les variantes du système sont sélectionnées et configurées dans le produit final et décrites dans des modèles instanciés (par exemple, un modèle UML).

Cependant, la dérivation au niveau de l'architecture a été peu explorée, en tant qu'étape préalable à la dérivation de produits concrets. Par conséquent, les ingénieurs de la LDP font souvent manuellement la sélection des éléments de l'architecture de la ligne de produit (*Product Line Architecture* (PLA)) et des variantes qui feront partie d'un produit concret.

Une autre problématique est l'intégration des ontologies pour guidées les ingénieurs dans l'étape de sélection des fonctionnalités a inclure dans le produits finale.

Objectifs du travail

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 e-Banking.
- Développer les core asset du domaine (élément réutilisable) :
  - — Une ontologie de domaine
  - — Les diagrammes de modélisation du domaine.
  - — Le composant réutilisable
- Développer des applications e-Banking en se basant sur les core assets.
- Dérivation automatique des applications guidées par l'ontologie
- 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.

Organisation du mémoire

Pour mener à bien notre mémoire, nous avons organisé notre travail en cinq chapitres

● Chapitre 1 : Les lignes de produits logiciels

Est un chapitre de généralités, il représente une vue globale sur les lignes de produits logiciels, telles que l'historique, la définition des lignes de produits logiciels, et ses différentes étapes.

● Chapitre 2 : Les ontologies

Ce chapitre, consiste à concevoir l'objectif des ontologies, leurs différents types et composants, le processus de développement, les langages d'implimentations et de représentation, et leurs usages.

● Chapitre 3 : E-Banking

Ce chapitre sera consacré à la définition du e-banking et la présentation de l'évolution des technologies de l'information et de la communication dans le secteur bancaire.

● Chapitre 4 : Ingénierie du domaine

Dans ce chapitre, nous avons décrit le déroulement étape par étape de l'analyse et de la conception de notre application.

● Chapitre 5 : Ingénierie d'application

Dans ce dernier chapitre, nous avons décrit en particulier l'environnement dans lequel nous avons programmé notre application, les langages de programmation utilisés et nous terminerons par la description des principales interfaces de l'application.

Chapitre 1

Les lignes de produits logiciels

1.1 Introduction

Les LDP représentent peut-être le changement de paradigme le plus passionnant dans le développement logiciel depuis l'arrivée des langages de programmation de haut niveau.

L'évolution vers l'ingénierie des LDP est généralement fortement basée sur des considérations économiques. En raison de sa prise en charge de la réutilisation à grande échelle, une telle approche améliore principalement le côté processus du développement logiciel, c'est-à-dire qu'elle réduit les coûts, les délais de mise sur le marché et améliore les qualités des produits résultants [2].

Au cours des deux dernières décennies, les LDP sont devenues l'un des paradigmes de développement de logiciels les plus prometteurs pour augmenter considérablement la productivité des industries logicielles, l'idée clé est que la plupart des systèmes logiciels ne sont pas nouveaux ou isolés, au contraire, les systèmes logiciels dans un domaine d'application partagent plus de points communs que l'unicité.

Le but est de profiter de ces points communs pour ne pas devoir tout redévelopper, mais de créer une architecture de base à partir de laquelle de nouvelles applications ont construit facilement par l'assemblage des éléments réutilisables.

L'ingénierie de la LDP vise à supporter une large ligne de produits. Ces produits peuvent prendre en charge différents clients individuels ou peuvent s'adresser à des segments de marché entièrement différents [3].

1.2 Historique

Le rêve d'une réutilisation des logiciels à grande échelle est aussi vieux que le génie logiciel lui-même. Il y a eu de nombreuses tentatives ou initiatives de réutilisation de logiciels, mais généralement avec peu de succès [4].

L'idée des LDP vient de deux domaines de recherche : réutilisation et analyse de domaine et l'article de MC Ilroy (1968) [5] est considérée comme la source de l'idée du développement de logiciels basés sur la réutilisation, qui proposait à construire les logiciels à partir d'un ensemble des composants interchangeables, au lieu de programmer chaque ligne de code.

Plus tard dans les années 1970, Parnas a développé l'idée de familles de programmes qui peuvent être définies comme des ensembles de programmes dont les propriétés communes sont si étendues qu'il est avantageux d'étudier les propriétés communes des programmes avant d'analyser les membres individuels [6], qui est devenue la base d'ingénierie pour le développement de composants réutilisables et le développement d'applications basées sur la réutilisation.

Un autre prédécesseur des LDP peut être trouvé dans le travail de Neighbors publié en 1989 [7]. Neighbors a introduit les concepts de domaine et d'analyse de domaine, et a fourni la première approche systématique de l'ingénierie de domaine, connue sous le nom de Draco. Peu de temps après, d'importants efforts de recherche sur la réutilisation des logiciels et l'ingénierie de domaine ont suivi [8]-[10].

Le concept de LDP a été entièrement introduit au début des années 1990. L'une des premières contributions a été la description de la méthode *Feature-Oriented Domain Analysis* (FODA) [11]. Vers la même époque, un certain nombre d'entreprises ont commencé à aborder la question de manière plus systématique. Ces premières approches ont bénéficié d'investissements massifs en Europe dans le domaine de l'ingénierie des LDP logiciels. Les suivants sont parmi eux [12] :

- ARES (1995–1998) – Architectural Reasoning for Embedded Systems.
- Praise (1998–2000) – Product-line Realisation and Assessment in Industrial Settings.
- ESAPS (1999–2001) – Engineering Software Architectures, Processes and Platforms.
- CAFE (2001–2003) – from Concepts to Application in system-Family Engineering .
- FAMILIES (2003–2005) – FAct-based Maturity through Institutionalisation, Lessons-learned and Involved Exploration of System-family engineering .

Ces projets ont contribué à la construction systématique d'une communauté de recherche et de pratique en ingénierie de LDPs en Europe.

Dans le même temps, en particulier, le *Software Engineering Institute* (SEI) de l'Université Carnegie Mellon a soutenu le développement de l'ingénierie des LDP logiciels aux États-Unis, par le développement d'un Framework basé sur la réutilisation d'une série d'applications connexes, appelée ingénierie de LDP [13], notamment dans le contexte des organisations gouvernementales [4].

1.3 Définition

Une LDP est une famille de produits conçue pour prendre l'avantage de leurs aspects communs et de prévoir la variabilité [14].

Une LDP est un ensemble de systèmes d'information à forte intensité logicielle permettant de configurer des produits logiciels similaires, qui partagent des caractéristiques communes pour satisfaire les besoins d'un marché commercial particulier [15].

1.4 L'ingénierie des lignes de produits logiciels

L'ingénierie des LDP est le développement d'un ensemble de produits à partir d'un ensemble réutilisable de ressources suivant une architecture commune et un plan prédéfini [16].

La différence principale entre l'ingénierie logicielle traditionnelle et l'ingénierie de la LDP est le passage d'une vision temporaire du prochain produit à la vision stratégique du domaine d'activité.

Du point de vue de l'ingénierie logicielle traditionnelle, un processus de développement itératif est recommandé, composé généralement de quatre étapes consécutives :

- **Application Requirements** : La définition des besoins.
- **Application Design)** :(Conception) La spécification de la solution pour répondre à ces besoins.
- **Application Coding :** L'implémentation de la solution.
- **Application Testing :** tester et évaluer par rapport aux besoins.

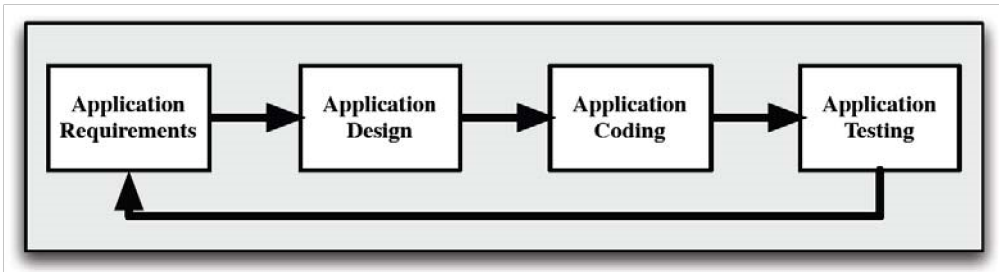


FIGURE 1.1 – Processus de développement logiciel classique [17]

Le processus de développement d'une LDP est divisé en deux processus distincts : Domain Engineering et Application Engineering. Figure 1.2.

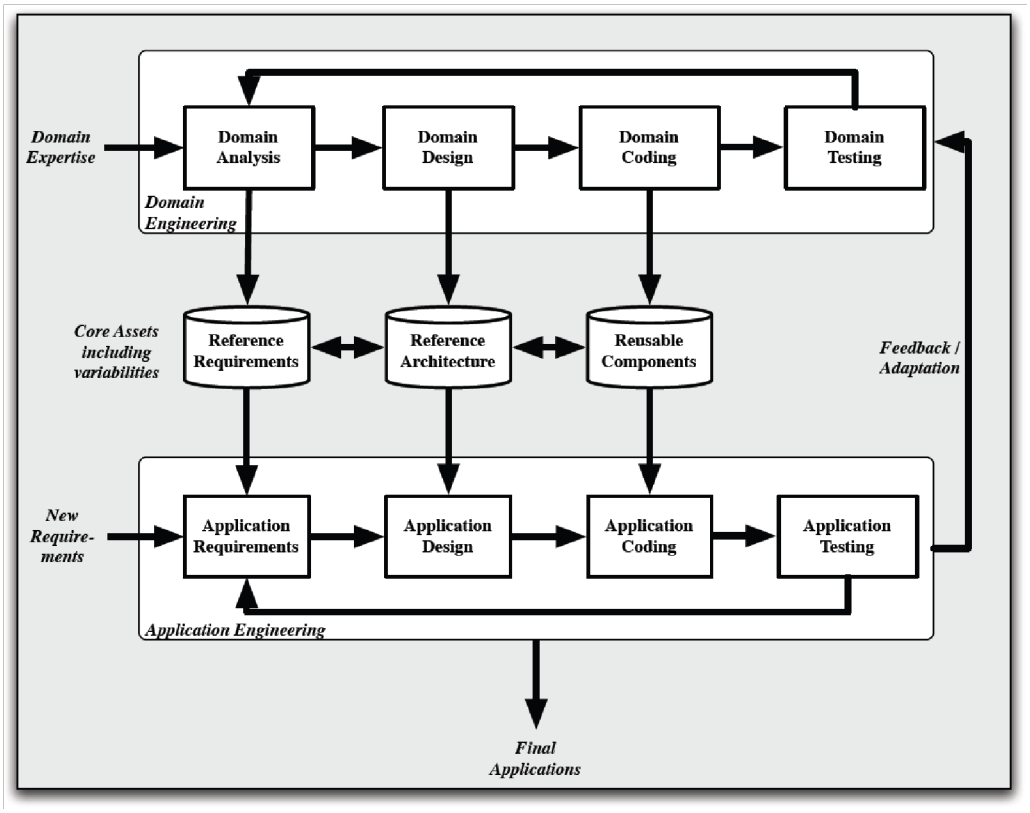


FIGURE 1.2 – Processus de développement d'une LDP [17]

1.4.1 Ingénierie de domaine

Le premier processus Ingénierie de domaine " Domain Engineering " a pour objectif de construire la plate-forme d'éléments réutilisables, à partir d'identification des membres de la ligne de produits, les similarités qui existe entre eux et les variabilités permettant de les différencier. Les principaux sortie de ce processus sont :

- **Scope :** La définition du scope de la ligne de produits est une énumération explicite des produits de la ligne et les produits qui pourront être ajouter aux future.
- **Exigences de référence (Reference Requirements) :** Il s'agît de définir les fonctionnalités (ou fonctionnalités qui sont des similarités et des variabilités) ainsi que les contraintes existantes entres elles.
- **Reusable Components :** l'ensemble de composants réutilisables implémentant les fonctionnalités.
- **Reference Architecture :** Une architecture de référence décrivant comment ces composants doivent être combinés afin de construire le produit final.
- **Production Plan :** Une documentation qui décrit comment utiliser l'architecture de référence et les composants réutilisables.

1.4.2 Ingénierie d'application

Le deuxième processus " Application Engineering " vise à dériver le logiciel final à partir d'éléments réutilisables. Ce processus s'intègre parfaitement au processus de développement classique figure 1.1. Cependant, chaque étape est facilitée en réutilisant la sortie de processus précédent.

- **Application Requirements :** Par contre au développement classique cette étape est moins compliqué a cause de l'existence des exigence de référence qui permet de se focaliser sur les besoins spécifiques du nouveau logiciel.
- **Application Design :** La conception de l'application consiste a configurer l'architecture de référence par rapport aux variabilités sélectionnés et l'adapter selon les besoins spécifiques.
- **Application Coding :** L'implémentation est principalement limité au développement de nouveaux composants ou à l'extension de composants de référence pour tenir en compte les besoins spécifiques.
- **Application Testing :** La réutilisation assure que les tests des composants ont été déjà spécifiés et exécutés, l'objectif de cette phase est de vérifié que l'interaction entre les composants réutilisable et le développement spécifique ne produit pas de comportement inattendu.

Ces deux processus doivent être parfaitement articulés puisqu'ils sont étroitement liés. En outre, une LDP doit toujours évoluer par des mise a jour fréquente sur les éléments réutilisables pour pouvoir répondre aux besoins du marché [17].

1.5 La variabilité

La variabilité de la ligne de produits logiciel est un regroupement des fonctionnalités qui différencient les produits de la même famille [18]. Elle est un concept clé qui couvre tout le cycle de vie d'une LDP, cela commence par les premières étapes de la définition de scope, couvrant jusqu'à la mise en œuvre (l'implémentation) et les tests et enfin l'évolution.

Une fonctionnalité combine les exigences fortement connexes auxquelles le produit logiciel final doit répondre, chaque fonctionnalité est simplifiée en sous-fonctionnalités pour obtenir des fonctionnalités terminales (variabilité) leur choix permet de distinguer les produits logiciels de la même famille. Ces fonctionnalités terminales sont associées à des éléments logiciels réutilisables (composant, service, etc.) implémentant les exigences déterminées par la fonctionnalité correspondante.

1.5.1 Types de variation

Une classification des différents types de variabilité dans les lignes de produits a été présentée dans [4] :

- **Similarité (Commonality) :** C'est une fonctionnalité qui est commune à tous les produits de la famille. Cette caractéristique est implémentée comme partie de la plate-forme commun de la famille.
- **Variabilité (Variability) :** C'est une fonctionnalité qui peut être commune à quelques produits, mais pas à tous les membres de la famille de produits.
- **Produit-Spécifique (Product-specific) :** une fonctionnalité peut faire partie d'un seul produit dans un avenir prévisible. De telles spécialités ne sont souvent pas requises par le marché en soi, mais sont des besoins des clients individuels. Bien que ces variabilités ne soient pas intégrées à la plate-forme, la plate-forme doit être en mesure de les prendre en charge.

Tandis que les points communs et les variabilités sont principalement traités dans l'ingénierie de domaine, les parties spécifiques au produit sont traitées exclusivement dans l'ingénierie d'application. Ceci est montré dans la Figure 1.3

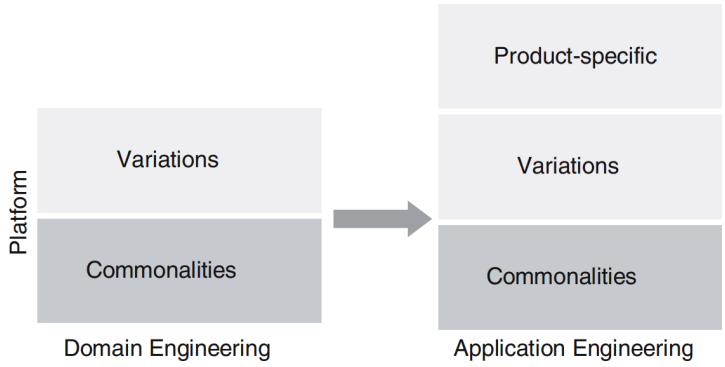


FIGURE 1.3 – La relation entre les différents types de variabilités [4]

1.6 La gestion de variabilité

Ces variabilités ont besoin d'être définies, représenter, exploité et implémenter, etc. d'une autre façon la gestion de la variabilité [4]. La gestion de variabilité permet d'identifier les fonctionnalités, de préciser les contraintes qui les relient et d'interpréter les alternatives proposées lors de la sélection (réutilisation) des fonctionnalités.

L'objectif de la gestion de la variabilité est de déterminer dans quel contexte, sous quelles conditions et comment une fonctionnalité peut être réutilisée de manière optimale. La gestion de la variabilité est décomposée en trois activités principales [17] :

- **L'identification de la variabilité :** qui détermine, les fonctionnalités terminales, les contraintes qui existent entre ces fonctionnalités et où et comment et pourquoi les variabilités peuvent apparaître.
- **L'implémentation de la variabilité :**qui détermine quels mécanismes (héritage, configuration, génération, 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 fonctionnalités lorsque de nouveaux points de variation ou de nouvelles variabilités apparaissent.

Lorsque le nombre de fonctionnalités augmente la gestion de la variabilité se complexifie très rapidement (exponentiellement) , et pour facilité la gestion de la variabilité durant tout le processus de développement, différents outils logiciels ont été développés, qui offrent principalement trois types de fonctionnalités [17] :

a) Un environnement de développement spécifique aux lignes de produits logiciels.

b) Des outils de configuration et de gestion du changement.

c) Des outils de tests et de vérification adaptés aux lignes de produits logiciels.

Cependant, même avec les meilleurs outils de gestion de la variabilité, il est clairement impossible de la gérer sans l'expliciter et la documenter clairement dès le départ.

1.7 La représentation de la variabilité

Pour représenter la variabilité, plusieurs approches différentes ont été conçues. Alors que la plupart des approches modernes utilisent des caractéristiques comme concepts de base pour la représentation de la variabilité tel que feature model [11], d'autres approches existent également comme orthogonal variability model [19]. L' objectifs est d'expliciter la variabilité dès les premières étapes du projet et réduisez la complexité de la gestion de la variabilité tout au long du processus de développement.

1.7.1 Feature Model

L'analyse de domaine est souvent traduite dans un document appelé feature model. En 1990, Kang a proposé FODA [11], afin d'identifier et représenter les fonctionnalités et les relations existant entre elles, cette méthode définit des notation graphiques et textuelles utilisé Feature Model. Cette méthode est minimal et simple a utilisé en comparaison avec d'autres méthodes de modélisation plus complexes tels que UML [20].

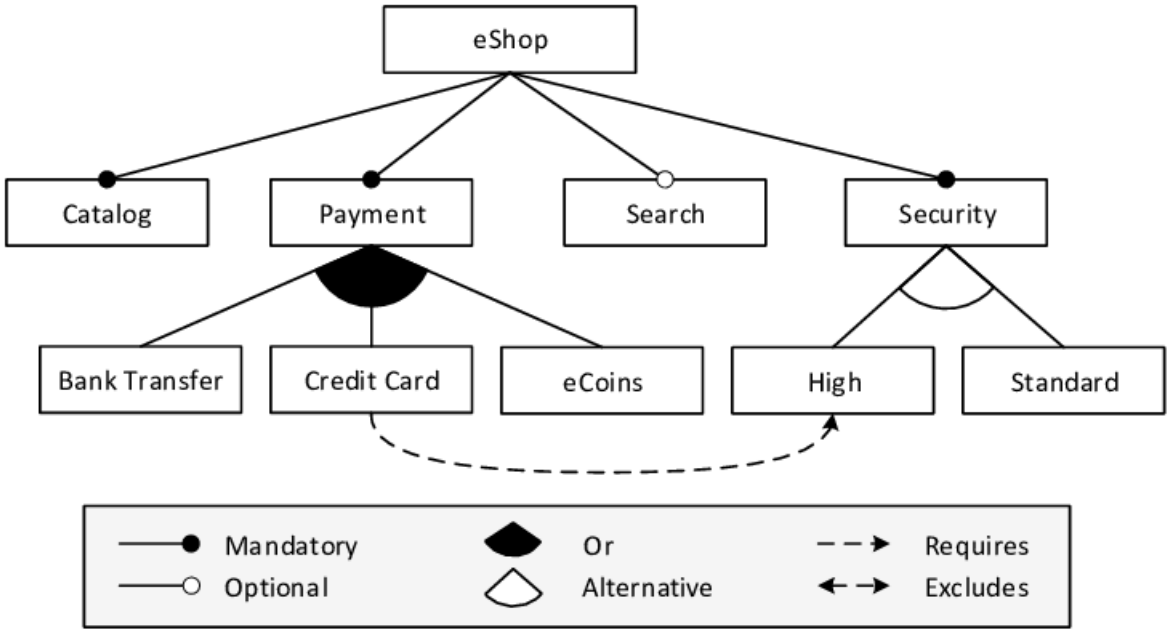


FIGURE 1.4 – Un feature model pour un système eShop [21]

Un feature model se présente généralement sous la forme d'un arbre. Chaque nœud correspond à une fonctionnalité et chaque arête correspond à une relation entre deux fonctionnalités. Différents types de fonctionnalités existent :

- **La racine (root) :** Un feature model est caractérisé et identifié par une fonctionnalité spécifique appelée la racine. Cette racine détermine le point d'entrée du diagramme, c'est le seul nœud sans nœud parent. Dans la Figure 1.4, la racine est la fonctionnalité " eShop " située au sommet de l'arbre et qui représente la ligne de produits.
- **Les fonctionnalités optionnelle (Optional) :** Si une fonctionnalité est facultative, alors si l'un de ses parents est sélectionné, elle ne l'est pas nécessairement. Du point de vue graphique, les fonctions optionnelles sont agrémentées de cercles creux. Dans la Figure 1.4, la fonctionnalité " Search " est considérée comme optionnelle.
- **Les fonctionnalités obligatoire (mandatory) :** Si une fonctionnalité est obligatoire alors si un de ses parents est sélectionné elle est aussi sélectionnée. La root est par définition toujours obligatoire. Du point de vue graphique, les fonctions obligatoires sont agrémentées de cercles pleines. Dans la Figure 1.4, la fonctionnalité " Payment " est considérée comme obligatoire.

Ces fonctionnalités sont reliées par différents types de relations :

- **Les décompositions :** Les fonctionnalités peuvent être décomposées en sous-fonctionnalités selon différents types de décomposition. Ces décompositions définissent des contraintes entre des entités qui partagent le même parent.

- *Xor-decomposition* : Cette décomposition signifie que si le parent est sélectionné alors un et un seul de ses enfants peut être sélectionné. Dans la Figure 1.4, la fonctionnalité " Security " est décomposée en deux sous-fonctionnalités : " High " et " Standard ". Cette décomposition est du type " xor-decomposition ", donc une seule sous-fonctionnalité sera sélectionné.
- *Or-decomposition* : Cette décomposition signifie que si le parent est sélectionné alors au moins un de ses enfants peut être sélectionné.la fonctionnalité " Payement " est décomposée en trois sous-fonctionnalités : " Bank Transfer ", " Credit card " et " eCoins ". Cette décomposition est de type " or-decomposition ".

- **Les contraintes :** Dans certains cas, les contraintes sont exprimés entre des fonctionnalités qui appartient au même Feature model, et qui ne partagent pas le même parent. Pour éviter le surchargement de modèle qui va réduire sa lisibilité les contraintes sont représentées de manière textuelle. Les deux contraintes les plus utilisées sont [17] :
  - *Requires* : La contrainte requires entre deux fonctionnalités (f1 requires f2) permet d'exprimer que si la fonctionnalité f1 est sélectionnée alors f2 doit nécessairement l'être, mais pas inversement. Dans la Figure 1.4, la sélection de la fonctionnalité " Credit card " impose la sélection de la fonctionnalité " High ".
  - *Excludes* : La contrainte excludes entre deux fonctionnalités (f1 excludes f2) permet d'exprimer que si une des deux fonctionnalités est sélectionnée alors l'autre fonctionnalité ne peut pas l'être en même temps pour le même produit.

1.8 Les perspectives et les défis

Malgré malgré leur avantages , les LDP ne sont pas une solution de tous les problèmes de réutilisation et de productivité. Dans cette section, nous faisons la distinction entre les différents perspectives et défis de cette approche.

1.8.1 Les perspectives

Les principaux avantages associés à une approche de lignes de produits logiciels portent principalement sur [17] :

- **L'amélioration la productivité et réduction des coûts de développement et de maintenance du produit final**. Les développeurs peuvent s'appuyer sur des exigences cohérentes et réutiliser des composants spécifiquement conçus pour optimiser la réutilisation. En outre, lorsque les produits sont mis en production, leur maintenance est nettement mieux contrôlée.

- **L'amélioration de la qualité du produit final**. Les développeurs peuvent réutiliser des composants de meilleure qualité en suivant des normes de programmation communes à toutes la ligne de produits. Cela évite la duplication de code et améliore le contrôle de la qualité du logiciel.

1.8.2 Les défis

Une LDP, ainsi que les avantages associés, ne semblent pas magiquement. Des efforts importants sont nécessaires et des défis importants sont relevés [17] :

- Même si le projet peut être exécuté de manière itérative, l'investissement initial reste considérable car il faut toujours avoir une vision globale de toute la ligne de produits, pas seulement des logiciels individuels, c'est-à-dire les logiciels passés, présents et futurs.

- Le retour sur investissement n'est pas immédiat et la rentabilité ne peut être atteinte qu'à moyen ou long terme. Il est nécessaire d'attendre un certain niveau de production de logiciels (Break even point) pour que de réels gains puissent être réalisés Figure 1.5.

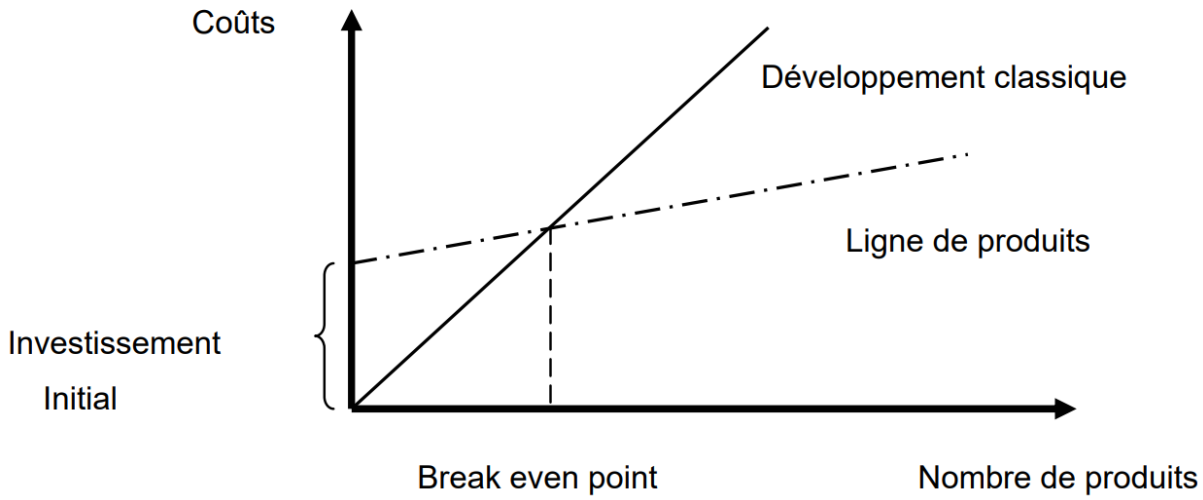


FIGURE 1.5 – Rentabilité des lignes de produits logiciels [17]

- Les coûts et les avantages sont pratiquement impossibles à estimer en détail avant la mise en œuvre d'une LDP. Cela laisse souvent les décideurs incapables d'évaluer le budget nécessaire et d'élaborer un plan raisonnable.

- La façon dont une LDP est conçue et développée implique une évolution radicale des mentalités et un changement organisationnel important. Chaque participant doit être convaincu des avantages potentiels pour lui-même et pour l'organisation dans son ensemble.

- Une attention particulière doit être portée à l'évolution et à la gestion des changements des lignes de produits logiciels. Toute modification apportée à un élément d'une ligne de produits doit être proprement contrôlée et tous les impacts potentiels sur les produits finaux doivent être pris en compte avant de mettre ces modifications en production. Par conséquent, afin d'augmenter l'efficacité, la flexibilité des lignes de produits logiciels est souvent réduite.

- La quantité d'information qu'il faut recueillir pour établir une nouvelle ligne de produits est souvent considérable. Le défi consiste à identifier les personnes et les documents pour recueillir assez d'informations pour comparer ces produits et pour identifier leurs similarités et leur variabilité.

1.9 Conclusion

L'approche de la LDP est très prometteuse pour la réduction significative sur les coûts de développement et de maintenances de logiciels et améliorer la qualité. Malheureusement, initialement un projet de LDP est très coûteux par rapport au développement classique, et le retour sur l'investissement n'est pas immédiat, mais après un certain nombre de produits, un seuil de rentabilité est atteint, ce qui entraîne des gains et des avantages. Plus récemment, l'approche de lignes de produits logiciels a été appliquée à un nombre de domaines tels que l'e-gouvernement, l'e-commerce et l'e-banking.

Chapitre 2

Les ontologies

2.1 Introduction

L'exploitation des connaissances informatiques vise à ne plus permettre aux machines de manipuler aveuglément l'information, mais à permettre une coopération entre le système et l'utilisateur. Il faut donc que le système ait accès non seulement aux termes employés par l'être humain, mais aussi à la sémantique qui s'y rattache, pour que la communication soit efficace.

Les ontologies visent à capturer les connaissances d'un domaine et à fournir une compréhension commune de ce domaine de manière générique, et aussi fournir une sémantique en définissant les connexions entre les différents concepts d'un domaine [22].

2.2 Définitions

Le mot "ontologie" est utilisé avec différentes significations dans différentes communautés. L'ontologie en tant que discipline philosophique est la branche de la philosophie qui traite de la nature et de la structure de la "réalité" [23].

Dans le contexte de l'informatique et des sciences de l'information, une ontologie définit un ensemble de primitives représentationnelles avec lesquelles modéliser un domaine de connaissance ou de discours. Les primitives de représentation sont généralement des classes (ou des ensembles), des attributs (ou des propriétés) et des relations (ou des relations entre les membres de la classe) [24].

De façon plus opérationnelle, l'ontologie cherche à décrire de façon formelle un domaine de connaissance, en identifiant les types , les propriétés et les relations des objets de ce domaine [25].

La définition la plus citée dans la littérature et par la communauté ontologique est celle de Gruber : "Une ontologie est une spécification formelle et explicite d'une conceptualisation partagée" [26].

- **Spécification explicite** : signifie que les éléments (les concepts, les propriétés, les relations, les fonctions, les restrictions et les axiomes) de l'ontologie doivent être clairement définis.
- **Formelle** : indique que la spécification doit être interprétable par machine.
- **Conceptualisation** : représente un modèle abstrait d'un domaine.
- **Partagé** : indique que elle est partagée par un groupe ou une communauté.

2.3 Composition d'une ontologie

Les ontologies fournissent un vocabulaire commun pour un domaine et définissent la signification des termes et les relations entre eux. Les connaissances en ontologie sont essentiellement formalisées selon cinq types de composantes [26] à savoir : concepts (ou classes), relations (ou propriétés), fonctions, axiomes (ou règles) et instances (ou individus).

1. **Concept :** Un concept est une abstraction qui rassemble un certain nombre d'entités du monde réel, et qui représente un ensemble d'objets et leurs propriétés communes. Les concepts d'ontologie sont choisis selon l'objet de l'ontologie et l'application prévue.
2. **Relation :** Les relations d'une ontologie désignent différentes interactions et associations existant entre les concepts du domaine. Une relation est définie comme une notion de lien entre des entités, exprimée souvent par un terme, les liens sont classés en deux catégories : des liens hiérarchiques et des liens sémantiques.

   Ces relations incluent les associations suivantes :

   a) Sous classe de (généralisation-spécialisation) est une relation binaire entre un concept général et un concept plus spécifique.

   b) Partie de (agrégation ou composition) est une relation qui permet de construire des concepts complexes à partir d'autres concepts appelés composants.

   c) Associe à.

   d) Instance de, etc.

   Ces relations permettent de voir la structure et les interrelations entre les concepts.
3. **Fonction :** Une fonction est un cas particulier d'une relation où un élément (le nième) de la relation est défini en fonction des N-1 éléments précédents.
4. **Axiome :** Les axiomes sont des expressions considérés toujours comme vrais, ils sont utilisés pour décrire les assertions de l'ontologie, et ils sont destinés à définir la description d'un concept et les relations qui permettent d'exprimer sa sémantique dans un langage logique.

L'inclusion des axiomes dans une ontologie peut avoir plusieurs objectifs :

- Définir la signification des composants.
- Définir des restrictions sur la valeur des attributs.
- Définir les arguments d'une relation.

5. **Instance :** C'est la définition extensionnelle de l'ontologie, une extension est un ensemble d'objets véhiculent les connaissances du domaine, elles sont utilisées pour représenter des éléments spécifiques dans un domaine donné [27].

2.4 Les types des ontologies

2.4.1 Type selon le l'objet de conceptualisation

Les ontologies sont regroupées dans le livre "Ontology Evaluation" [28] selon les objets modélisés par l'ontologie Fig.2.1.

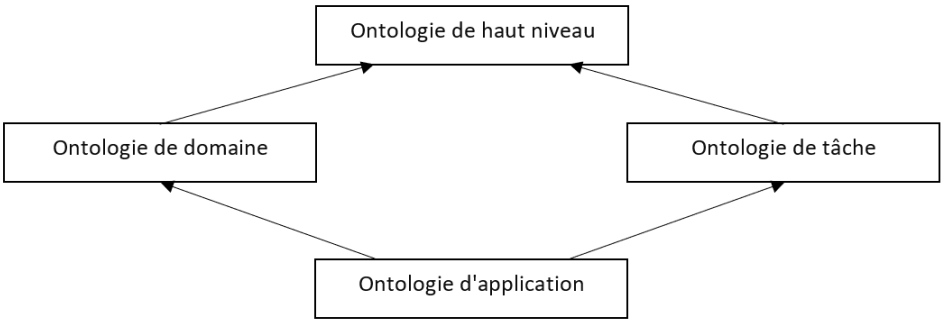


FIGURE 2.1 – Classification des ontologies selon N. Guarino [29]

1. **Ontologies de haut niveau :** également appelées méta-ontologies, ou core d'ontologies [30]. Elle exprime des concepts très généraux valables dans différents domaines. Ces concepts ne sont pas liés à un problème ou à un domaine particulier et, au moins en théorie, doit faire l'objet d'un accord de la part d'une vaste communauté d'utilisateurs.
2. **Ontologies de représentation de connaissances :** Ce type d'ontologie est un cas spécial d'ontologies de haut niveau qui rassemble des concepts déjà utilisés pour formaliser la connaissance. Indépendant des domaines , car elles décrivent des primitives cognitives communes [31]. Un exemple sur les ontologies de représentation est Frame-Ontology qui définit les concepts utilisés dans les langages fondés sur les frames : classes, sous classes, attributs, valeurs, relations et axiomes [26].
3. **Ontologies du domaine :** Ce type d'ontologies fournit un vocabulaire des concepts, les relations entre ces concepts, les activités, les théories et les principes d'un domaine particulier, donc elles sont réutilisable seulement dans un domaine donné.

4. **Ontologie de Taches :** Ce type d'ontologie est utilise pour résoudre des problèmes lies a des tâches spécifiques dans les systèmes, ces tâches peuvent ou non appartenir au même domaine. Ces ontologies fournissent un ensemble de termes par lesquels on peut décrire à un niveau générique comment résoudre une classe de problèmes [32].
5. **Ontologies d'application :**

   Cette ontologie est la plus spécifique, elle contient suffisamment de connaissances requises pour une application particulière [30].Correspondant à la réalisation d'une tâche particulière, leur domaine d'application est limité. Ce sont généralement des spécialisations d'ontologie de domaine et d'ontologie de tâche.

2.4.2 Selon le niveau de détail de l'ontologie

La classification suivante est fondée sur le degré de granularité [33].

1. **Granularité fine** correspondant à des ontologies très détaillées, possédant ainsi un vocabulaire plus riche capable d'assurer une description détaillée des concepts pertinents d'un domaine ou d'une tâche.
2. **Granularité large** correspondant à un vocabulaire moins détaillé. Les ontologies de haut niveau ont une granularité large, du fait que les notions sur lesquelles elles portent peuvent être raffinées par des notions plus spécifiques.

2.4.3 Selon le niveau du formalisme

M. Uschold et M. Grüninger ont défini quatre types d'ontologies : les ontologies informelles, les ontologies semi formelles et les ontologies rigoureusement formelles [34].

1. **Informelles :** Ce type d'ontologie est exprimé en langage naturel.
2. **Semi informelles :** Exprimé dans un langage naturel restreint et structuré.
3. **Formelles** exprimé en langage artificiel, et définis des termes avec une sémantique formelle, des théorèmes et des preuves de propriétés.
4. **Semi formelles :** exprimé dans un langage artificiel formellement défini.

2.5 Processus de développement d'une ontologie

Le processus de développement de l'ontologie fait référence aux activités à réaliser lors de la construction d'une ontologies. Cependant, le processus de développement de l'ontologie n'implique pas un ordre d'exécution de ces activités. Son objectif est de d'identifier la liste des activités à réaliser. Le processus de développement de l'ontologie d'après [35] se compose des éléments suivants :

1. **Planification :** Avant de construire une ontologie, il faut planifier les principales tâches à accomplir, comment elles seront organisées, le temps nécessaire pour les réaliser et avec quelles ressources (personnes, logiciels et matériel).
2. **Spécification :** Dans cette étape, l'objectif de l'ontologie (scénarios d'usage), le domaine de connaissance et les utilisateurs sont spécifiés. Ceci permet de choisir le degré de formalisme de l'ontologie ainsi que sa granularité.
3. **Conceptualisation :** Cette étape comprend l'identification des connaissances acquises dans le domaine dans des modèles conceptuels qui décrivent le problème et sa solution.
4. **Formalisation :** Pour transformer le modèle conceptuel en un modèle formel ou semi-formel, il faut le formaliser à l'aide de systèmes de représentation ou la logique de description.
5. **Intégration :** Les ontologies sont construites pour être réutilisées. Par conséquent, il faut intégrer autant que possible les ontologies existantes dans la construction d'une nouvelle ontologie.
6. **Implementation :** Pour rendre une ontologie computable, il faut l'implémenter dans un langage formel.
7. **Documentation :** L'absence d'une bonne documentation est un obstacle important lorsque la réutilisation et le partage des ontologies déjà construites, donc la documentation est très importante même pour la maintenance de l'ontologie.
8. **Évaluation :** L'utilisation des ontologies erronés à l'étape d'intégration résulte une ontologie qui est aussi erroné, donc il est indispensable de l'évaluer avant la partagé. L'évaluation est réalisé par des tests correspondant à l'objectif opérationnel de l'ontologie.
9. **Maintenance :** A tout moment, quelqu'un peut demander l'inclusion ou la modification d'une définition dans l'ontologie. Maintenir l'ontologie est une activité importante à faire avec soin. Des consignes pour maintenir l'ontologie sont également nécessaires.
10. **Évolution :** Le cycle de vie évolutifs est le plus approprié pour la construction d'une ontologie, puisque elle se développe en fonction des besoins. L'ontologie peut évoluer si l'évaluation a échoué, si le domaine est élargit ou si le contexte d'usage est modifié.

Ainsi, le processus de développement de l'ontologie transforme le produit initial (le besoin de construire l'ontologie) en un produit final (l'ontologie évaluée, documentée, codifiée dans un langage formel).

2.6 Cycle de vie des ontologies

Les activités nécessitent un ordre et qu'elles doivent être divisées et réalisées étape par étape de manière planifiée, afin de réussir. Le cycle de vie a pour objectif d'indiquer l'ordre et la profondeur dans lesquels les activités de développements doivent être réalisées.

De toute évidence, la planification est première et la maintenance est la dernière, mais il n'est pas clair si l'acquisition de connaissances est totalement ou partiellement simultanée avec les activités de spécification et de conceptualisation, et si la conceptualisation précède l'intégration et celle-ci va avant l'implémentation, si l'évaluation et la documentation sont séquentielles à l'implémentation ou si elles doivent être faites au fur et à mesure que les activités avancent, et si une activité doit être entièrement ou partiellement achevée avant de commencer l'activité suivante.

Fernández [35] a proposé le cycle de vie suivant (Figure 2.2) pour répondre à ses questions :

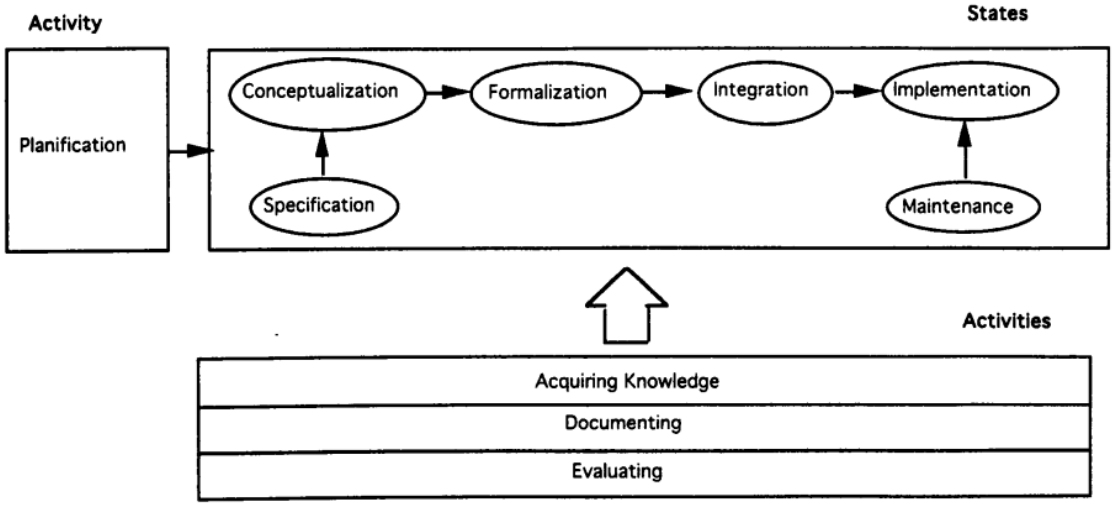


FIGURE 2.2 – Cycle de vie d'une ontologie [35]

Comme le montre la figure 2.2 L'acquisition des connaissances, l'évaluation des ontologies et la documentation sont des tâches qui sont effectuées tout au long de la vie de l'ontologie. En fait, à moins que l'ontologue ne soit un expert du domaine d'application, la plupart de l'acquisition se fait en même temps que la phase de spécification des exigences et diminue au fur et à mesure que le processus de développement de l'ontologie avance.

Afin d'éviter la propagation des erreurs, la majorité de l'évaluation devrait être réalisée dans les premières étapes du processus de développement de l'ontologie.

2.7 Les langages des ontologies

2.7.1 RDFS :

Ressource Description Framework Schema (RDFS) est un vocabulaire pour décrire les propriétés et les classes des ressources RDF, avec une sémantique pour les hiérarchies de ces propriétés (rdfs :subPropertyOf) et classes (rdfs :subClassOf) [36].

Le système de classes et de propriétés RDFS est similaire aux systèmes de types des langages de programmation orientés objet, mais il diffère par le fait qu'au lieu de définir une classe en termes de propriétés que ses instances peuvent avoir, RDFS décrit les propriétés en fonction de classes de ressources auxquelles elles s'appliquent.

Les propriétés en RDFS sont définis par le domaine de définition (rdfs :domain) et le domaine image (codomaine) (rdfs :range) et le nom de propriété.

Par exemple, la propriété "auteur" peut avoir un domaine de définition "Document" et un domaine image de "Personne", alors qu'un système classique orienté objet généralement définit une classe Livre avec un attribut appelé auteur de type Personne, l'avantage de l'approche RDFS est que il est possible d'utiliser "Document" et "Personne" pour exprimer d'autre propriétés [37].

```
<rdf:Property rdf:ID="author">
    <rdfs:domain rdf:resource="#Document"/>
    <rdfs:range rdf:resource="#Person"/>
</rdf:Property>
```

FIGURE 2.3 – Exemple RDFS en XML

2.7.1.1 RDF :

Ressource Description Framework (RDF) est un modèle de données pour les objets ("ressources") et les relations entre eux, fournit une sémantique simple pour ce modèle de données, et ces modèles peuvent être représentés dans une syntaxe *Extensible Markup Language* (XML) [36]. Une ressource peut être un document, une personne, des objets physiques et des concepts abstraits. Une déclaration RDF exprime une relation entre deux ressources, la première ressource est appelé sujet et la deuxième est appelé objet, la relation entre eux appelé propriété, donc RDF est représenté par des triplés (sujet, propriété, objet).

Dans l'exemple ci-dessus, Bob est le sujet de quatre triplés, et la Mona Lisa est le sujet d'un triplé et l'objet de deux autres. La possibilité d'avoir la même ressource en position de sujet d'un triplé et en position d'objet d'un autre, permet de trouver des connexions entre les triplés, ce qui est une partie importante de la puissance de RDF.

```
<Bob> <is a> <person>.
<Bob> <is a friend of> <Alice>.
<Bob> <is born on> <the 4th of July 1990>.
<Bob> <is interested in> <the Mona Lisa>.
<the Mona Lisa> <was created by> <Leonardo da Vinci>.
<the video 'La Joconde à Washington'> <is about> <the Mona Lisa>
```

FIGURE 2.4 – Exemple de triplés RDF (exprimé de façon informelle en pseudo-code) [38]

On peut visualisé les triples sous forme d'un graphe où les sujets est les objets sont les nœuds et les propriété sont les arcs.

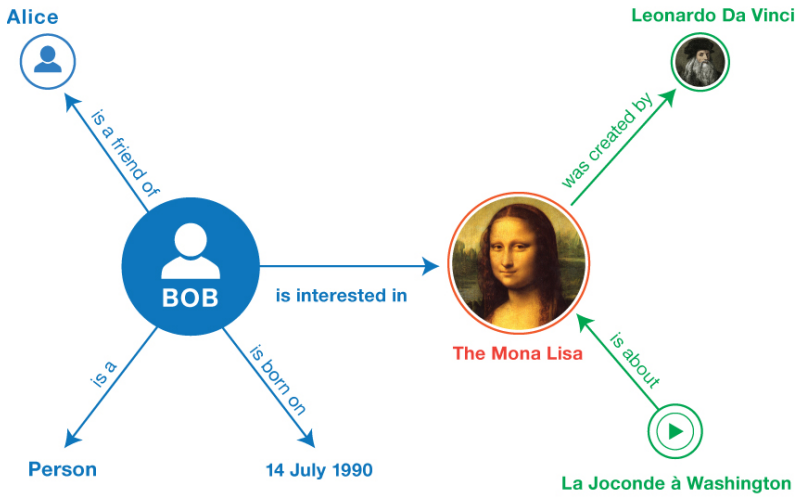


FIGURE 2.5 – Exemple de graphe RDF [38]

2.7.2 OWL :

Web Ontology Language (OWL) a été recommandé par le *World Wide Web Consortium* (W3C) pour enrichir RDFS par la définition d'un vocabulaire plus complet pour décrire des ontologies complexes. Il est enrichi par rapport à RDFS en lui ajoutant de nouvelles notions telles que : l'équivalence des classes, l'équivalence des relations, la symétrie et la transitivité des relations, la cardinalité, etc.

2.7.2.1 Sous Langages de OWL

OWL offre trois sous langages destinés à des communautés de développeurs et d'utilisateurs spécifiques [36] :

1. Le langage **OWL Lite** concerne les utilisateurs ayant principalement besoin d'une hiérarchie de classifications et de mécanismes de contraintes simples. Par exemple, quoique OWL Lite gère des contraintes de cardinalité, il ne permet que des valeurs de cardinalité de 0 ou 1.

2. Le langage **OWL DL** s'adresse aux utilisateurs désirant un maximum d'expressivité. tout en maintenant la complétude de calcul (toutes les conclusions sont garanties d'être calculables) et la décidabilité (tous les calculs se termineront dans un temps fini).

   OWL Description Logics (OWL DL) comprend toutes les expressions du langage OWL , mais elles ne peuvent être utilisées qu'avec certaines restrictions.Il est basé sur la logique de description d'où son nom OWL Description Logics.

3. Le langage **OWL Full** est destiné aux utilisateurs qui veulent une expressivité maximale et la liberté syntaxique de RDF sans garantie de calcul (raisonnement). La syntaxe ne subit pas de modification par rapport à OWL DL mais OWL Full permet une utilisation sans restrictions.

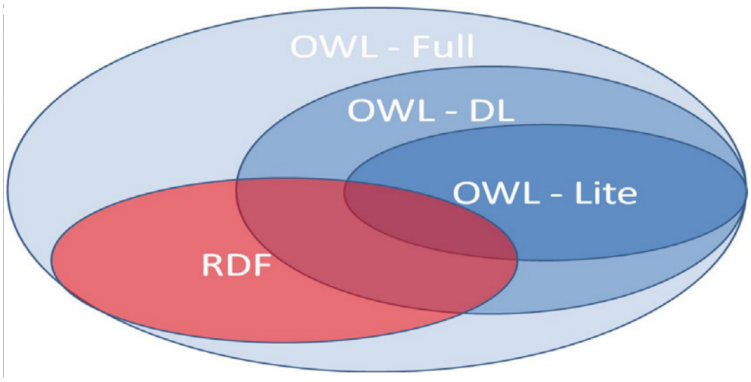


FIGURE 2.6 – Hiérarchie de langage OWL [39]

2.7.2.2 Propriétés

OWL distingue deux catégories principales de propriétés qu'un constructeur d'ontologie peut vouloir définir [36] :

— Les propriétés d'objet relient les individus aux individus. Une propriété d'objet est définie comme une instance de la classe intégrée OWL owl :ObjectProperty.

— Les propriétés de type de données relient des individus à des valeurs de données. Une propriété de type de données est définie comme une instance de la classe intégrée OWL owl :DatatypeProperty.

Les deux classes owl :ObjectProperty et owl :DatatypeProperty sont des sous-classes de la classe RDF rdf :Property.

Un axiome de propriété définit les caractéristiques d'une propriété. Dans sa forme la plus simple, un axiome de propriété définit simplement l'existence d'une propriété.

Par exemple : **<owl :ObjectProperty rdf :ID="isParentOf"/>**, Ceci définit une propriété avec la restriction que ses valeurs doivent être des individus.

Souvent, les axiomes de propriété définissent des caractéristiques supplémentaires des propriétés. OWL supporte les concepts suivants pour les axiomes de propriété [36] :

— concepts de RDFS : rdfs :subPropertyOf, rdfs :domain et rdfs :range.

— Relations avec d'autres propriétés : owl :equivalentProperty et owl :inverseOf.

— Les caractéristiques logiques des propriétés : owl :SymmetricProperty et owl :Transitive-Property.

2.8 Utilisations des ontologies

Uschold et Gruninger [34] divisent l'espace d'utilisation de l'ontologie dans les catégories suivantes :

— Communication.

— Inter-opérabilité.

— Ingénierie des systèmes : Spécification, Fiabilité et Réutilisabilité.

2.8.1 Communication

Les ontologies réduisent la confusion conceptuelle et terminologique en offrant un cadre unifié dans une organisation.

De cette manière, les ontologies permettent une compréhension et une communication communes entre des personnes ayant des besoins et des perspectives différentes en fonction de leur contexte particulier.

Il existe plusieurs aspects de l'utilisation des ontologies pour faciliter la communication entre les personnes au sein d'une organisation.

1. **Modèles normatifs :** Grâce à l'ontologie, il est possible de construire un modèle normatif qui permet d'avoir une compréhension commune du système et de ses objectifs.
2. **Réseaux de relations :** Les ontologies peuvent servir à créer un réseau de relations pour suivre ce qui est connecté, explorer et naviguer sur ce réseau.
3. **Cohérence et réduction d'ambiguïté :** En adoptant une ontologie partagée, tous les participants utilisent une terminologie standardisée pour tous les objets et relations de leurs domaines.
4. **Intégration des points de vue des utilisateurs :** En utilisant une ontologie pour fournir un modèle normatif du système, on peut intégré les différents perspectives des utilisateurs.

2.8.2 Inter-opérabilité

L'interopérabilité est traitée dans de nombreuses applications des ontologies, afin de créer un environnement d'intégration pour différents outils logiciels, utilisés par différents utilisateurs.

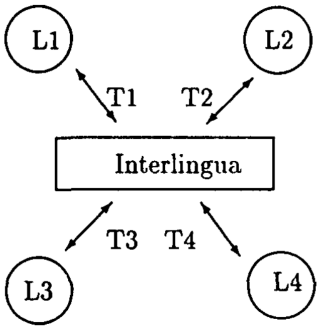


FIGURE 2.7 – L'ontologie en tant qu'outil interlinguistique [34]

— **Les ontologies en tant qu'inter-lingua :** Pour favoriser l'interopérabilité, les ontologies peuvent être utilisées pour prendre en charge la traduction entre différents langages et représentations. En utilisant les ontologies comme support de traduction inter-lingua,on peut traduire d'une ontologie native vers une ontologie d'échange (voir la figure 2.7).

2.8.3 Ingénierie des systèmes

Les applications des ontologie ne se limitent pas au rôle que jouent les ontologies dans le fonctionnement des systèmes logiciels, mais les ontologies peuvent également soutenir la conception et le développement de systèmes logiciels [34].

1. **Spécification :** Une compréhension commune du problème et de la tâche à accomplir peut faciliter la spécification des systèmes logiciels.

   Les ontologies facilitent le processus d'identification des exigences du système et la compréhension des relations entre les composants du système, et elles fournissent une spécification déclarative d'un système logiciel, permettant de raisonner sur l'objectif de système, au lieu de la prise en charge de ses fonctionnalités.

2. **Fiabilité :** Les ontologies peuvent améliorer la fiabilité des systèmes logiciels en servant de base à la vérification manuelle (ontologie informelles) ou semi-automatique (ontologie formelles) de la conception par rapport à la spécification.

   De plus, les ontologies formelles peuvent être utilisées pour clarifier diverses hypothèses faites par différents composants d'un système logiciel, facilitant ainsi leur intégration.

3. **Réutilisabilité :** Pour être efficaces, les ontologies doivent également permettre la réutilisation, de sorte qu'il soit possible d'importer et exporter des modules entre différents systèmes logiciels.

Les ontologies fournissent une bibliothèque "facile à réutiliser" d'objets de classe pour modéliser des problèmes et des domaines dans le but de construire une bibliothèque d'ontologies qui peut être réutilisée et adaptée à différentes catégories de problèmes et de contextes.

Pour être utiles, ces ontologies doivent être adaptables et extensibles, permettant d'intégrer de nouvelles classes de contraintes et de spécialiser les concepts et les contraintes pour un problème particulier.

2.9 Travaux connexes

- **Czarnecki et Kim**

  Dans [40], Czarnecki et Kim décrivent la relation entre le Feature model et les ontologies, ils discutent de l'essence des Feature model, en particulier de leur structure hiérarchique et de leur variabilité. Ils concluent que les ontologies sont plus expressives que les Feature model. La contribution de leur travail est de proposer que le modèle de fonctionnalité peut être interprété comme des vues sur les ontologies.

  Pour justifier cette hypothèse, Czarnecki et Kim présentent des exemples de Feature model et leur représentation équivalente à l'aide d'ontologies. Cependant, la correspondance directe entre les deux notations n'est pas exploitée. Les associations syntaxiques et sémantiques sont claires, mais une mise en correspondance formelle n'a pas été explorée.

- **Bu-qing, Bing et Qi-ming**

  Bu-qing, Bing et Qi-ming [41] proposent une approche de modélisation de la variabilité des LDP basée sur les processus et l'ontologie. Le travail est limité au paradigme orienté service qui est la base du développement de la ligne de produit. Le principal inconvénient de ces travaux est que le point de départ de la modélisation du domaine LDP est une ontologie.

  En fait, ce nouveau modèle n'est pas nécessaire, puisque les auteurs proposent de remplacer le Feature model par le modèle ontologique. Étant donné qu'il est largement utilisé par la communauté LDP, le fait de retirer le Feature model du paradigme de développement LDP entraînerait un effort supplémentaire pour changer la notation de modélisation de la variabilité.

- **Zaid, Kleinermann et De Troyer**

  Zaid, Kleinermann et De Troyer [42] ont développé une ontologie pour formaliser la spécification des Feature model. Ces travaux se concentrent sur la fourniture d'un cadre ontologique permettant de vérifier la cohérence des modèles et de détecter les conflits à l'aide de règles prédéfinies.

- **Falbo, Guizzardi et Duarte ; Peng, Zhao, Xue et Wu**

  Dans [43] et [44], les auteurs présentent des approches ontologiques pour l'ingénierie de domaine en LDP. Ces approches sont essentiellement basées sur la définition d'ontologies afin de représenter le domaine. Comme dans les autres travaux mentionnés, ils considèrent également l'ontologie comme le point de départ de la LDP.

Malgré les efforts pertinents des travaux connexes présentés dans cette section, il existe encore des lacunes en ce qui concerne l'expressivité des notations utilisées dans les LDP, et le manque d'outils qui permet de profiter de l'ontologie développée.

Par conséquent, le principal avantage de notre proposition est de réutiliser le Feature model existant et d'ajouter des descriptions sémantiques et le développement d'un outil qui permet d'utiliser l'ontologie pour simplifier l'étape de sélection des fonctionnalités.

Dans le tableau 2.1, nous comparons les travaux connexes présentés et notre approche proposée. Dans un premier temps, nous vérifions si l'approche réutilise les informations déjà décrites dans le diagramme de fonctionnalités, en vérifiant si ces informations sont traduites dans l'ontologie par un mapping, et si oui, si ce mapping est manuel ou automatique. Enfin, nous vérifions si le travail propose un outil pour sélectionner les fonctionnalités a inclure dans le produit finale de la LDP.

	Feature model	Mapping	Outil
Czarnecki	Oui	Non	Non
Bu-qing	Non	Non	Non
Peng	Non	Non	Non
Zaid	Oui	Manuel	Non
Falbo	Non	Non	Non
Approche proposé	Oui	Manuel	Oui

TABLE 2.1 – Comparaison des travaux ontologiques connexes.

2.10 Conclusion

Dans ce chapitre, nous avons rappelé la notion des ontologies, leurs différents types et composants, le processus de développement, les langages d'implémentations et de représentation, et leurs usages, et enfin nous avons présenté quelques travaux connexes.

Chapitre 3

E-Banking

3.1 Introduction

Le secteur bancaire constitue le moteur de toute économie. C'est une industrie ayant des impacts énormes sur le développement du pays, via sa contribution non négligeable au financement de l'économie.

La relation entre les banquiers et les clients connaît des changements majeurs avec l'augmentation d'utilisation des produits électroniques. Les différentes révolutions technologiques ont apporté de nouveaux modes de mise en relation électronique et permis d'imaginer de nouveaux moyens de paiement par l'e-banking. Sous la pression d'une modernisation croissante, de l'intrusion des nouvelles technologies et de l'émergence de nouveaux acteurs sur le marché des services financiers, les banques cherchent à adapter les solutions non seulement pour établir leur propre spécificité, mais aussi leur propre identité.

On parle ainsi de la révolution technologique en matière d'information et de communication qui a permis de nouvelles formes de distribution de services bancaires [45].

3.2 Définition

L'e-banking ou la banque électronique a contribué de manière significative à la mise en place et à l'automatisation des services financiers et à la croissance des volumes de transactions en ligne. Il s'agit d'accéder à des services bancaires au moyen d'une interface interactive comme un navigateur Web. Sa disponibilité donne accès aux comptes, aux opérations et aux renseignements financiers récents. Il est accessible via un guichet automatique, un téléphone ou un ordinateur, généralement connecté via un accès sécurisé [45].

3.3 L'évolution d'E-banking

Depuis la fin des années 1990, les services bancaires en ligne sont passés d'insignifiants à des dizaines de millions d'utilisateurs dans le monde [46]. Cependant, la banque électronique est le produit de différentes générations de transactions électroniques, et c'est la dernière de plusieurs générations de systèmes : *Guichet Automatique Bancaire* (GAB) , banque par téléphone, PC banking ou banque à domicile.

Les GAB ont été les premières machines connues à fournir un accès électronique aux clients. Dans le cas des services bancaires par téléphone, les utilisateurs appellent le système informatique de leur banque à partir de leur téléphone habituel et utilisent le clavier du téléphone pour effectuer des transactions bancaires. Les services bancaires par ordinateur ont remplacé les services bancaires par téléphone et ont permis aux utilisateurs d'interagir avec leur banque via un ordinateur connecté au réseau téléphonique via un modem commuté. Les services bancaires par téléphone et PC impliquent des coûts de maintenance associés à la mise à jour de divers modems et à l'évitement de procédures de configuration trop compliquées [47].

Après ces générations, Deutsche Bank a lancé son premier projet de services bancaires par Internet en Amérique latine en 1996, et Citibank a développé une "e-toolkit" spécial dans toutes ses succursales dans le monde [48].

De nombreuses banques ont commencé à considérer les services bancaires en ligne comme un impératif stratégique. L'attrait des banques pour les services bancaires en ligne est clair : des coûts de transaction réduits, une intégration de service plus facile, des capacités de marketing interactif et d'autres avantages qui augmentent les listes de clients et les marges bénéficiaires. De plus, les services bancaires en ligne permettent aux institutions de regrouper plusieurs services dans un package simple pour attirer les clients et minimiser les frais généraux.

Selon les recherches de Forrester [49], à la fin de 2002, les utilisateurs de services bancaires en ligne représentaient 37% des utilisateurs d'Internet et de services bancaires en ligne, et le nombre de transactions en ligne augmentera considérablement par rapport au nombre d'opérations de succursales traditionnelles.

L'e-banking utilise le navigateur web pour l'interface utilisateur et l'internet pour le transfert de données et le téléchargement de logiciels, et présente donc un potentiel de réduction des coûts de maintenance [49]. Pour les utilisateurs, e-banking offre des informations actualisées et un accès 24h/24 aux services bancaires. Les principaux services fournis par les banques électroniques sont le transfert d'argent entre ses propres comptes, le paiement de factures et la vérification du solde des comptes, les prêts, le courtage, l'échange d'actions, le regroupement de services et une foule d'autres services financiers s'ajoutent à ces services primaires [50].

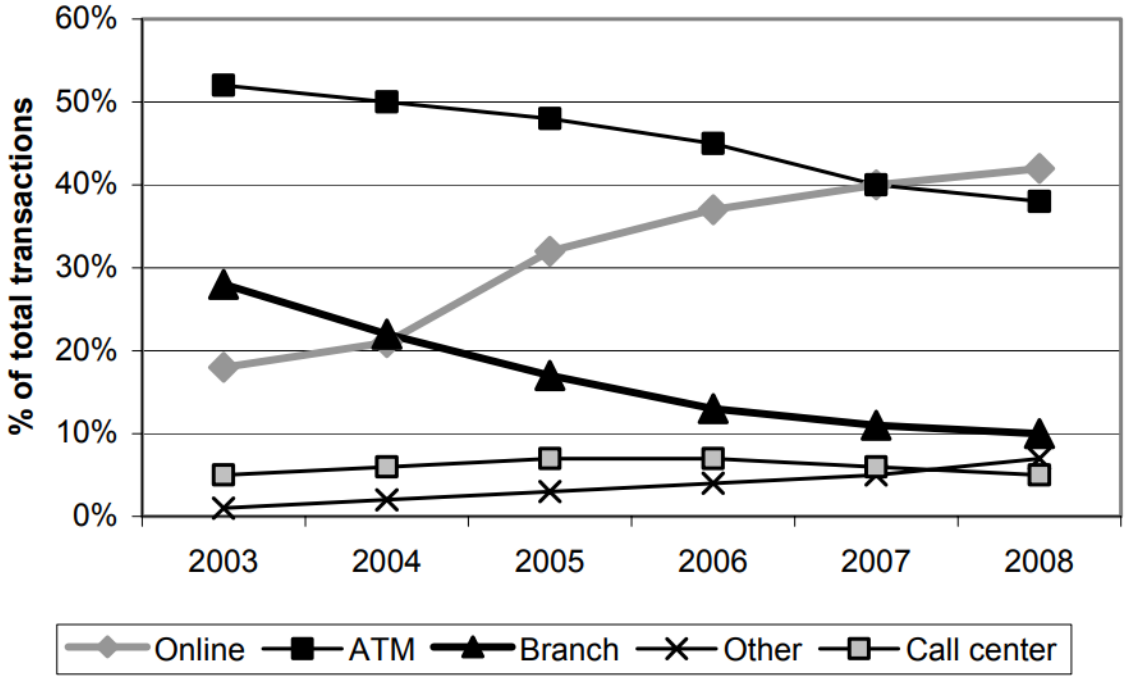


FIGURE 3.1 – Le volume des transactions bancaires dans les banques européennes entre (2003-2008) [49]

3.4 Les différentes formes d'E-Banking :

L'e-banking représentent des transactions électroniques entre les banques et les clients et fait référence à plusieurs types de services au moyen desquels les clients peuvent demander des informations ou effectuer des transactions, selon les besoins des clients et du dispositif utilisé [51].

— **Téléphone Banking :**

Dans les années 1970, les consommateurs pouvaient utiliser leurs anciens téléphones à domicile pour vérifier leur solde, transférer des fonds et payer des factures. Les services bancaires par téléphone permettent aux clients d'effectuer des opérations bancaires au téléphone à tout moment, le seul inconvénient, cependant, est que la vision du client n'est pas prise en compte lors de la transaction. Les services bancaires par téléphone peuvent être divisés à deux types [52] :

— Automatique : Ces fonctions doivent être effectuées et résolues par le système automatique sans qu'il soit nécessaire de faire appel à des opérateurs humains.

— Opérateur assisté : Ces fonctions conviennent aux clients qui ont des problèmes et des exigences très complexes qui ne peuvent pas être satisfaits par des services automatisés, ou qui ne sont pas à l'aise avec les services automatisés, ou qui exigent des services à haute sécurité.

— **Mobile banking :**

Barnes et Corbitt [53] définissent le m-Banking comme la combinaison d'Internet et des téléphones portables, un service qui permet aux clients d'utiliser la connexion Internet de leur téléphone portable pour vérifier les soldes de leurs comptes bancaires, transférer des actifs entre comptes, contrôler les limites de crédit, payer des factures etc. La banque fournit des services bancaires mobiles via des téléphones mobiles numériques et certains appareils sans fil.

— **PC banking :**

Il s'agit d'une forme d'e-banking qui permet aux clients d'accéder à des services bancaires à partir d'un ordinateur et de les utiliser à l'aide d'un logiciel d'application bancaire particulier [54]. Les banques ont commencé à offrir ce service au milieu des années 1980, où les clients pouvaient utiliser un ordinateur personnel avec un abonnement à l'intranet de la banque et accéder à leurs comptes avec un mot de passe [55]. Le service est moins populaire et n'est pas largement utilisé ou utilisé par de nombreuses banques car il nécessite des systèmes propriétaires et d'énormes investissements dans la technologie que seules quelques banques peuvent gérer.

— **Internet banking :**

Il s'agit d'une forme de banque électronique qui permet aux clients d'accéder à leurs comptes ou d'effectuer des transactions sur leurs comptes via un site Web bancaire, en utilisant Internet comme canal de distribution [51]. La banque en ligne diffère de PC banking en ce sens qu'il n'est pas nécessaire d'installer un logiciel spécifique pour accéder aux services bancaires [54].

— **GAB**

Les guichets automatiques fonctionnent 24 heures sur 24 et sont situés à divers endroits, en particulier dans les zones les plus peuplées, ils sont donc faciles à trouver. L'utilisation d'un guichet automatique nécessite une carte de guichet automatique et un code PIN personnel, permettant aux clients d'accéder à des services bancaires tels que les dépôts, les retraits, les virements, les demandes de solde de compte, les demandes de chéquier, les relevés de compte, etc.

Les types de services bancaires électroniques se multiplient certainement, mais ne semblent pas répondre aux mêmes besoins, de ce fait, chaque forme de banque offre différentes opportunités en termes de création de valeur pour le client.

3.5 Avantages de l'e-banking

- **Commodité :**

  Les banques qui proposent des services bancaires en ligne permettent aux clients d'effectuer des opérations bancaires n'importe où tant qu'ils disposent d'une connexion Internet. Toutes les opérations bancaires peuvent être effectuées dans le confort du domicile ou du bureau du client. De plus, ces services sont disponibles 24 heures sur 24, 7 jours sur 7 [56].

- **Facilité d'utilisation :**

  La facilité d'utilisation est définie comme la possibilité de vérifier plusieurs comptes à travers une seule interface [56]. Elle désigne également les outils de navigation disponibles, la fonction de recherche, le niveau d'interactivité et la conception de l'interface [57].

- **Coût réduit :**

  L'e-banking permet de fournir des produits et services aux clients à des coûts inférieurs à ceux des banques traditionnelles [57]. L'e-banking pur élimine les frais généraux des succursales physiques et du personnel. Il s'agit de fournir aux clients l'accès et l'utilisation des services bancaires au moindre coût [56].

- **Le facteur temps :**

  Le gain de temps est un facteur clé lié à l'utilisation de l'e-banking, car le client n'a pas besoin de se déplacer à l'agence bancaire pour demander des informations ou effectuer une transaction bancaire [57].

- **Livraison rapide :**

  Un avantage essentiel de l'e-banking est la livraison rapide de l'information sur la valeur de l'argent en tout lieu et à tout moment par un simple clic [57].

- **Paiement de factures en ligne :**

  Le paiement de factures en ligne est l'une des transactions électroniques les plus populaires. Au lieu d'avoir à faire des chèques pour payer les factures, avec l'e-banking le client a seulement besoin d'un simple clic.

3.6 Les défis de l'e-banking

Malgré les nombreux avantages présentés ci-dessus, l'utilisation de l'e-banking présente certains inconvénients.

- **Problèmes d'accessibilité :**

  Le client de la banque ne peut pas avoir accès à l'e-banking si la connexion internet n'est pas disponible [58].

- **Difficulté d'utilisation :**

  Les clients qui sont nouveaux dans la technologie et qui ont des difficultés à utiliser un téléphones ou Internet seront confrontés à certains obstacles [58].

- **Coût de démarrage élevé :**

  La banque électronique nécessite un coût de démarrage élevé. Il comprend l'installation d'Internet, le matériel et les logiciels avancés, ainsi que les coûts informatiques et de maintenance [58].

- **Durée du processus d'authentification :**

  L'un des principaux défis liés à l'utilisation de l'e-banking est la sécurité des systèmes et des transactions. Pour remédier à ce problème, beaucoup de banques essaient de mettre en place des systèmes de sécurité améliorés. Le client doit donc passer beaucoup de temps à effectuer le processus d'authentification [58].

- **Problèmes de disponibilité :**

  L'utilisation des services de banque en ligne est liée à la disponibilité du serveur de la banque. Par conséquent, si celui-ci est en panne, l'accès aux comptes sera impossible [58].

- **Problèmes de sécurité :**

  La sécurité des transactions est l'un des principaux problèmes liés à l'utilisation des services bancaires en ligne. Les clients des banques craignent que leurs comptes ne soient piratés ou consultés par des personnes non autorisées. Ils craignent également que les fonds qu'ils transfèrent n'atteignent pas leurs bénéficiaires prévus [58].

3.7 Conclusion

L'e-banking facilite les comparaisons entre les services et les produits bancaires, peut accroître la concurrence entre les banques et permettre aux banques de pénétrer de nouveaux marchés, élargissant ainsi leur portée. Certains y voient même un moyen de faire des économies dans les pays aux systèmes financiers sous-développés, où les clients peuvent accéder plus facilement aux services bancaires situés à l'étranger, car les systèmes de communication sans fil évoluent plus rapidement que les réseaux traditionnels [59].

Par ailleurs, l'adoption de l'e-banking est devenue incontournable pour les banques qui souhaitent conserver leurs parts de marché tout en fidélisant leurs clients.

Chapitre 4

Ingénierie du domaine

4.1 Introduction

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

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

4.2 Analyse du domaine

Le but de l'analyse du domaine (Domain Analysis) Figure 1.2 est d'étudier le domaine de la LDP et d'identifier les similarités et les variabilités (Exigences de références) et le feature model est la principale technique pour représenter la variabilité du domaine dans les LDP. Cependant, les diagrammes de fonctionnalités ne suffisent pas à modéliser certaines spécificités de domaine.

Par exemple, un ingénieur logiciel du domaine peut vouloir regrouper des fonctionnalités dans des catégories, ou même associer des fonctionnalités à différentes choses, comme la personne qui en est responsable ou le code source qui l'implémente. Cette inadéquation du feature model est une conséquence de son expressivité limitée, en particulier par rapport aux ontologies sémantiquement plus riches [40].

De plus, ces diagrammes n'ont pas été conçus pour faciliter la recherche d'informations, l'interopérabilité et l'inférence. En revanche, les ontologies semblent être la meilleure solution pour répondre à ces exigences.

4.2.1 Diagramme de cas d'utilisations

Avant de modéliser le Feature model et construire l'ontologie, on doit identifié les acteurs et spécifier les exigences minimal de chaque produit dans notre ligne qui est possible après l'étude du domaine e-banking faite au chapitre 3.

4.2.1.1 Identifications des acteurs

— **Visiteur :** C'est toute personne pouvant consulter le site dans le but d'avoir des renseignements.

— **Client :** Personne inscrite auprès de la banque et qui a le droit d'accéder à l'espace client pour consultation ou pour effectuer des opérations après authentification.

— **Admin :** est la personne qui a le droit d'accéder à l'espace administrateur, son rôle est la gestion de la banque.

4.2.1.2 Spécification des besoins

Acteurs	Besoins
Visiteur	Consulter le site .
	S'inscrire.
Client	Ouvrir un compte.
	Demander une carte de crédit.
	Consulter et modifier ses informations personnelles.
	Consulter ses comptes.
	Consulter et gérer ses cartes de crédits
	Geler (Bloquer) sa carte.
	Consulter ses transactions.
	Consulter ses statistiques
	Effectuer des virements entre compte.
	Transferts inter-compte (virement interne).

	payer ses factures.
Administrateur	Gérer les clients.
	Gérer les comptes.
	Gérer les cartes de crédits.
	Consulter des statistiques.

TABLE 4.1 – Cas d'utilisations pour chaque acteur

4.2.1.3 Diagramme des cas d'utilisation Client et visiteur

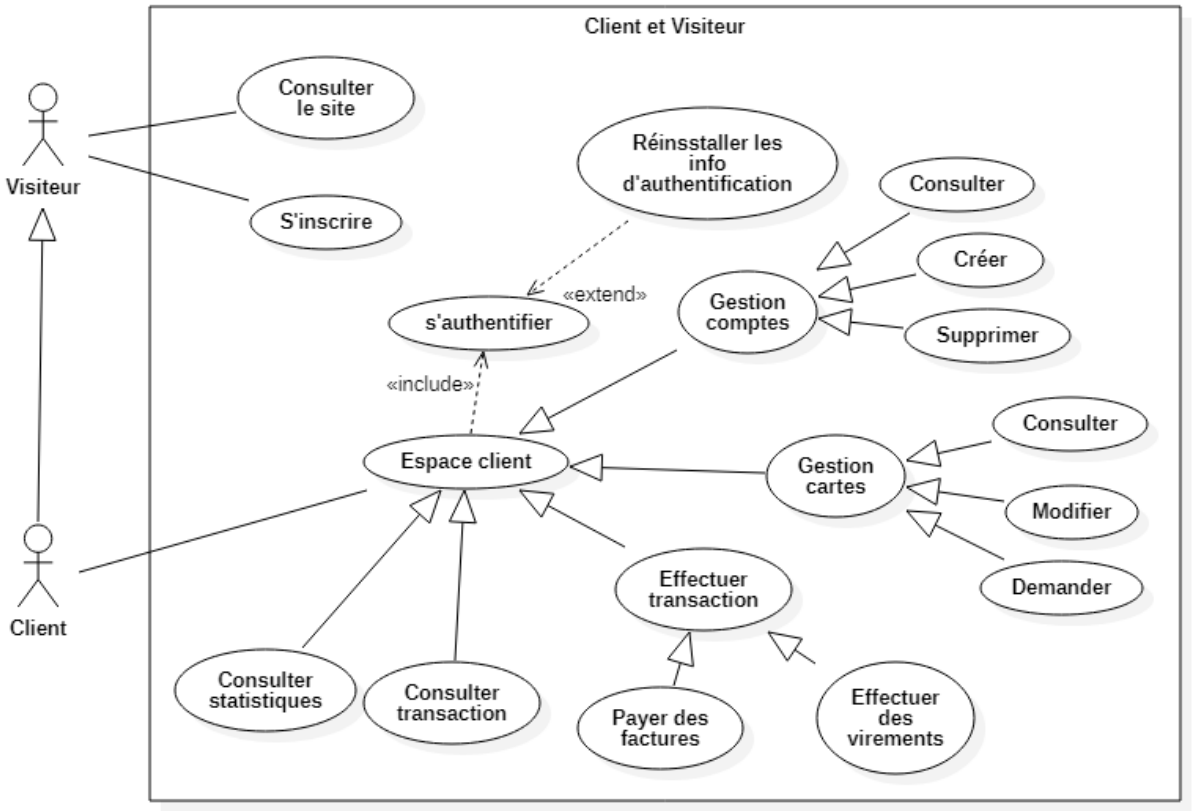


FIGURE 4.1 – Diagramme des cas d'utilisation Client et visiteur

4.2.1.4 Diagramme des cas d'utilisation Admin

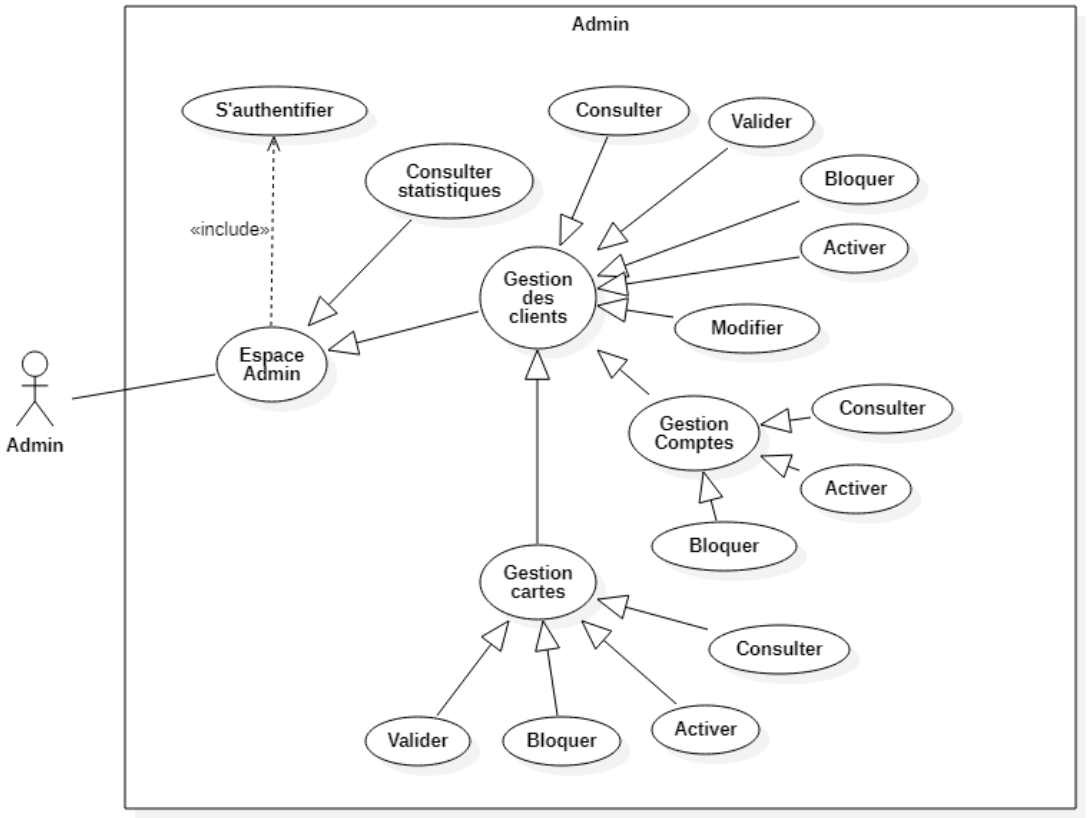


FIGURE 4.2 – Diagramme des cas d'utilisation Admin

4.3 Modélisation du domaine

Après l'analyse du domaine on arrive à la deuxième étape de l'ingénierie de domaine la modélisation du domaine (Domain design) 1.2 , la création de Feature model et l'ontologie et quelques diagrammes UML [60].

4.3.1 Feature Model

Le Feature model établi est constitué de deux modèles, qui se distinguent selon les types de fonctionnalités qu'il contient :

Le premier est le Feature model métiers, qui regroupe les fonctionnalités liées au cœur de métier de la ligne de produits, c'est-à-dire les fonctions pour lesquelles la ligne est crée. Ce diagramme comporte les fonctionnalités **"Espace Client"** et **"Espace Admin"** sont des fonctionnalités obligatoire.

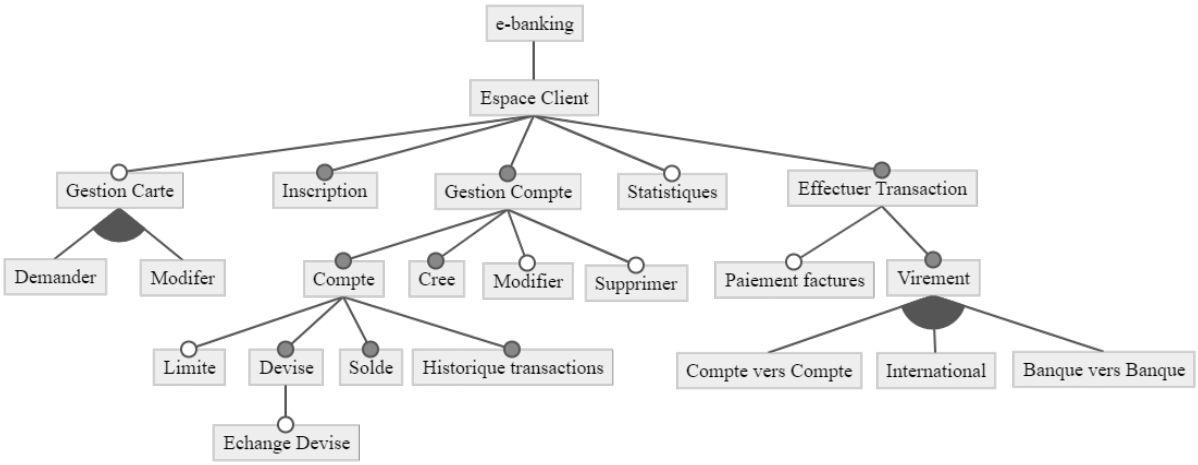


FIGURE 4.3 – Feature model métier - Espace Client

Les contraintes :

Espace Client **requires** Authentification.

Créer Compte **excludes** Demander Compte.

Demander Compte **excludes** Créer Compte.

Carte Bancaire **requires** Compte.

Effectuer Transaction **requires** Compte.

Inscription **requires** Valider Inscription.

Gestion document **requires** Authentification.

Historique transactions **requires** Effectuer Transaction.

- **Espace Client**
  implique plusieurs sous fonctionnalités tel que :
  - **"Inscription"** (permet au visiteur de demander de crée un compte)
  - **"Historique des transactions"**(permet au client de consulter l'historique de ses transactions) qui sont de fonctionnalités obligatoires.
  - **"Statistiques"**(permet au client de voir des statistiques sur ses comptes) qui sont des fonctionnalités optionnel.
  - La fonctionnalité obligatoire **"Gestion Compte"** implique trois autre fonctionnalités : **"Demander un nouveau compte"** (permet pour un client de demander un autre compte mais nécessite une validation par l'Admin),**"Créer compte"**, (permet pour un client de crée un compte directement sans validation), **"Modifier"** (Modifier un compte) et **"Supprimer"** (permet au client de supprimer son compte).
  - La fonctionnalité **"Gestion Cartes bancaire"** est optionnel et elle implique deux autres fonctionnalités : **"Modifier"** (permet de modifier le code PIN de la carte) et **"Demander une nouvelle carte"**.
  - La fonctionnalité obligatoire **"Effectuer Transaction"** implique deux autre fonctionnalités : **Paiement factures**, et **Virement** le client peut transférer des fonds soit d'un compte à un autre (Virement **"Compte vers Compte"**), soit à un compte dans une autre banque (Virement **"Banque vers Banque"**), ou il peut effectuer un virement international vers une banque à l'étrange (Virement **"International"**).

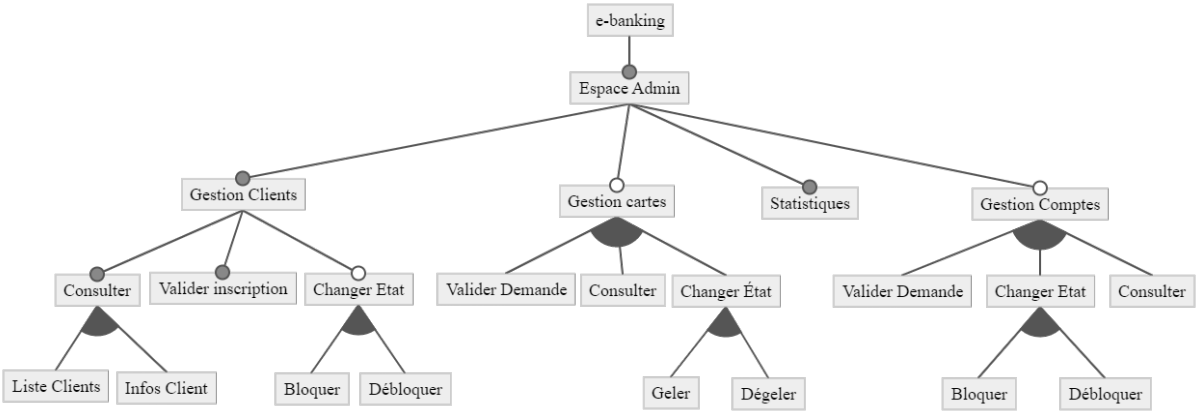


FIGURE 4.4 – Feature model métier - Espace Admin

FIGURE 4.5 – Notation de feature model

Les contraintes :

Espace Admin **requires** Authentification.
Gestion Cartes **requires** Carte Bancaire.
Gestion Comptes **requires** Compte.

- **Espace Admin**

  implique quatre sous fonctionnalités qui sont :

  — **"Statistiques"** est une fonctionnalité obligatoire (permet a l'admin de voir des statistiques générale sur toute la banque).

  — La fonctionnalité **"Gestion Client"** est obligatoire et implique trois autre fonctionnalités : **Changer État** (Bloquer un client et ces comptes, et Débloquer un client bloqué) , **Valider Inscription** (Valider une inscription d'un nouveau client), **Consulter**(Consulter les informations d'un client et afficher la listes des clients).

  — **"Gestion Comptes"** est une fonctionnalité optionnel qui implique trois autre fonctionnalités : **"Valider Demande"** (permet l'admin de valider une demande de compte d'un client), **Changer État** (Bloquer un compte, ou Débloquer un compte bloqué), **"Consulter"** (Consulter les informations d'un compte d'un client).

  — La fonctionnalité **"Gestion Cartes bancaire"** est optionnel et elle implique trois autres fonctionnalités : **"Consulter"** , **"Changer État"** (Geler une carte ou Activer une carte gelé) et **"Valider Demande"**(Validation d'une de demande de carte d'un client par l'admin).

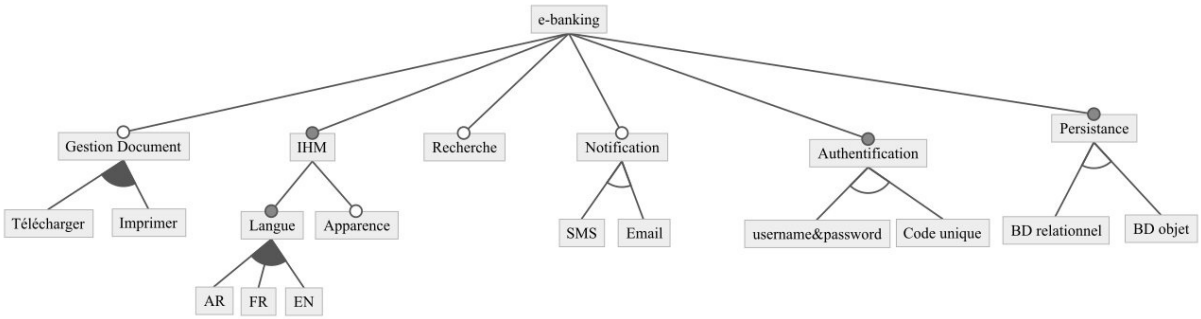


FIGURE 4.6 – Feature model technique

- **Feature model technique**

  Le deuxième est le Feature model technique, qui contient Les fonctionnalités techniques :

  — Le fonctionnalité **"Gestion Document"** implique deux fonctionnalités : **Télécharger** (permet télécharger les différents documents fournis par la banque), **Imprimer** (permet d'imprimer les différents document fournis par la banque).

  — **"Persistance"** est une fonctionnalité obligatoire et implique soit l'utilisation d'une **BD relationnel** ou bien une **BD objet**.

  — **"IHM"** est une fonctionnalité obligatoire qui implique la fonctionnalité optionnel **Apparence**(Mobile ou Desktop) et la fonctionnalité obligatoire **Langue** (permet de changé la langue d'affichage).

  — Pour la fonctionnalité **"Notification"** est une fonctionnalité optionnel et implique soit l'utilisation des **Email** ou bien des **SMS** pour notifier l'utilisateur de système.

  — la fonctionnalité obligatoire **Authentification** (pour authentifier les utilisateurs de systèmes) permet au utilisateurs de s' authentifier soit par l'utilisation d'un nom d'utilisateur et un mot de passe (**username password**) ou bien l'utilisation d'un **Code unique**.

4.3.2 Ontologie des lignes de produits logiciels

Notre contribution dans ce travail consiste a adopté les ontologies afin d'améliorer le travail de Filho [61] qui a proposé des moyens d'améliorer la sémantique des lignes de produits logiciels en utilisant des Feature model comme points de départ, en les mappant sur des ontologies, Protège [62] est utilisé pour construire notre ontologie.

Les concepts et les relations de l'ontologie sont définis afin de formaliser les fonctionnalités et leurs relations possibles. Ensuite, les fonctionnalités extraites du Feature model sont mises en correspondance avec les individus des concepts de l'ontologie.

4.3.2.1 Les concepts de l'ontologie :

Les concepts de l'ontologie représentent la notion de fonctionnalité et ses types selon la variabilité. Mandatory, optional, Or et Alternative sont des types de fonctionnalités chargés de définir l'aspect variabilité de la LDP. Chacun de ces types a sa propre sémantique, par exemple, si F est une MandatoryFeature alors F doit être sélectionnée.

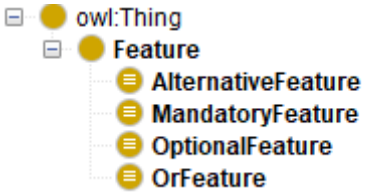


FIGURE 4.7 – Les classes de l'ontologie proposée

4.3.2.2 Les relations de l'ontologie :

Dans l'ontologie, la relation **isParentOf** est définie afin d'exprimer qu'une fonctionnalité est le parent d'une autre. Cette relation a pour domaine et image l'ensemble des fonctionnalités. De plus, elle a la propriété inverse **isChildOf**. La relation **hasOrFeature** indique qu'une fonctionnalité particulière est le parent d'une OrFeature. De la même manière, la relation **hasAlternativeFeature** exprime qu'une fonctionnalité est le parent d'une AlternativeFeature.

Comme mentionné précédemment, il existe des relations entre les fonctionnalités qui sont complémentaires à la variabilité, mais qui ne peuvent pas être exprimées dans le Feature model, comme les contraintes qui sont exprimées séparément du Feature model dans la majorité des travaux.

Dans notre ontologie on a ajouté deux relations, la première est la relation **requires**, qui exprime la dépendance entre deux fonctionnalités, symétriquement ou non. La seconde est la relation **excludes**, qui signifie l'exclusion d'une deuxième caractéristique en sélectionnant la première.

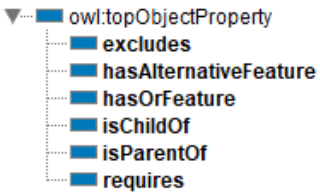


FIGURE 4.8 – Les propriétés de l'ontologie proposée

4.3.2.3 Représentation des fonctionnalités dans l'ontologie

Les fonctionnalités de notre LDP sont représenté par des instances des concepts de l'ontologie, et la relation **isParentOf** est utilisé pour construire la hiérarchie de Feature Model Figure 4.9.

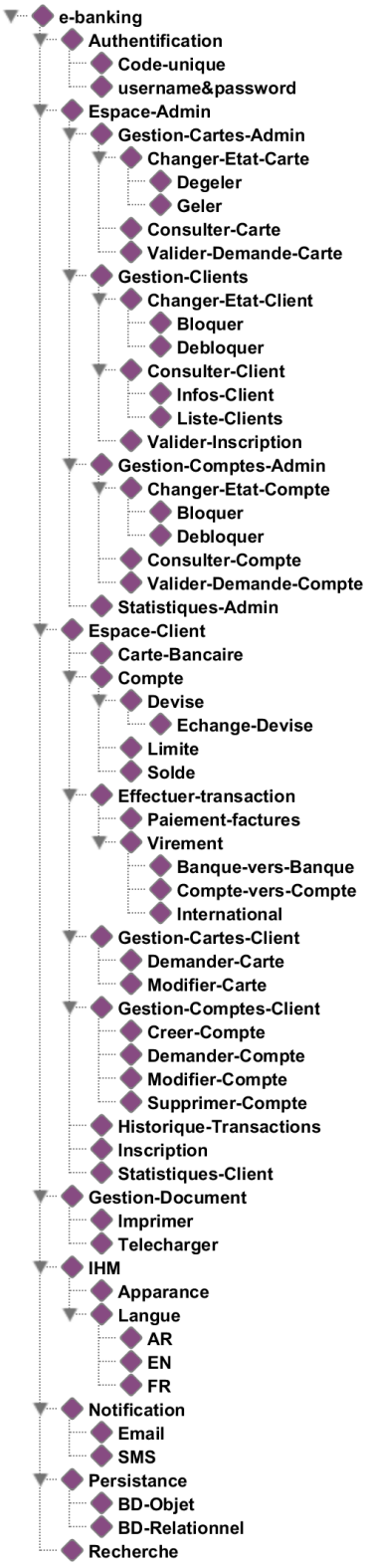
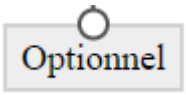
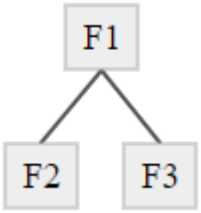
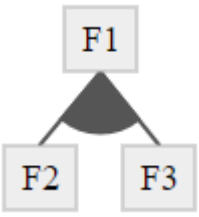
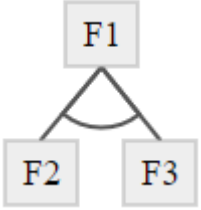


FIGURE 4.9 – La hiérarchie des fonctionnalités

4.3.2.4 Les règles de passage

Les principales transformations sont définies et illustrées dans le tableau 4.2. La première colonne indique le symbole dans le Feature Model, la deuxième contient la signification du symbole, et la troisième est l'axiome correspondant dans la description logiques.

Feature model	Signification	Ontologie (Notation DL)
	Root Feature F	F : Feature
	Mandatory Feature (Fonctionnalité Obligatoire)	F : MandatoryFeature
	Optional Feature (Fonctionnalité Optionnelle)	F : OptionalFeature
	F1 et le parent de F2 et de F3	isParentOf(F1, F2) isParentOf(F1, F3)
	Or décomposition	F2 :OrFeature F3 :OrFeature hasOrFeature(F1,F2) hasOrFeature(F1, F3)
	Xor décomposition	F2 :AlternativeFeature F3 :AlternativeFeature hasAlternativeFeature(F1, F2) hasAlternativeFeature(F1, F3)

Passage du Feature model a l'ontologie.

4.3.3 Diagramme de séquence

Un diagramme de séquence représente l'échange de messages entre objets. Il peut représenter un processus de manière simplifiée en mettant l'accent sur les échanges entre acteurs ou entre acteurs et systèmes.

Parmi les objets utilisés, nous avons retenu l'objet Vue, qui représente l'interface entre acteur et systèmes tels que des pages Web ou des écrans de saisie.

Dans ce qui suit nous allons traduire quelques scénarios en diagrammes de séquences :

- S'inscrire
- Authentification
- Virement

4.3.3.1 Diagramme de séquence de cas d'utilisation S'inscrire

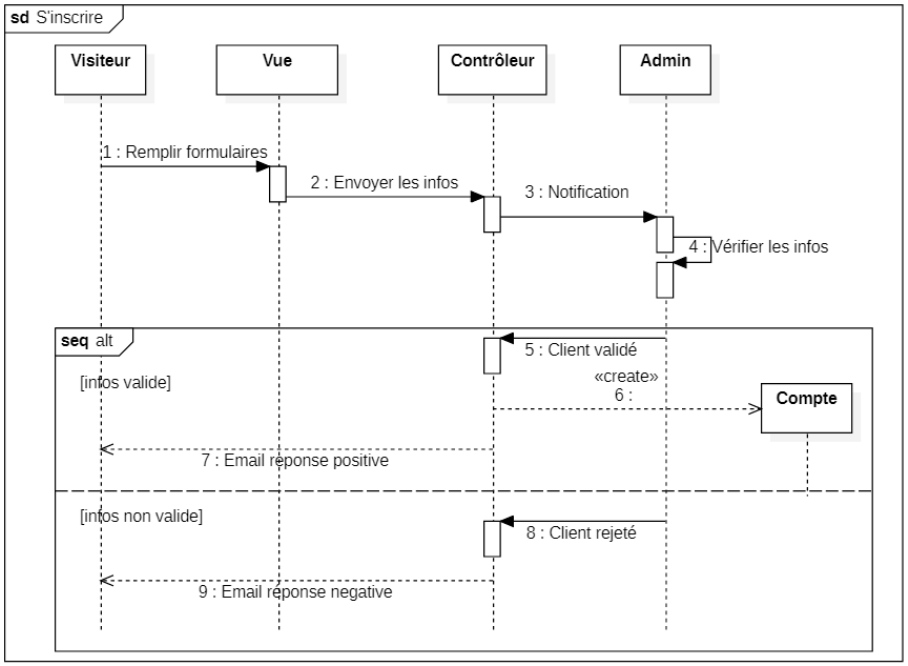


FIGURE 4.10 – Diagramme de séquence de cas d'utilisation S'inscrire

Scénario nominal de S'inscrire :

- Le visiteur remplit le formulaire d'inscription.
- L'admin accède à « valider client », la demande d'inscription s'affiche, il vérifie les informations saisies.
- Si les informations sont correctes, l'admin valide la demande, sinon, la rejette.
- Un message de confirmation sera affiché et un mail informatif de la réponse de l'admin sera envoyé au client.

4.3.3.2 Diagramme de séquence de cas d'utilisation S'authentifier

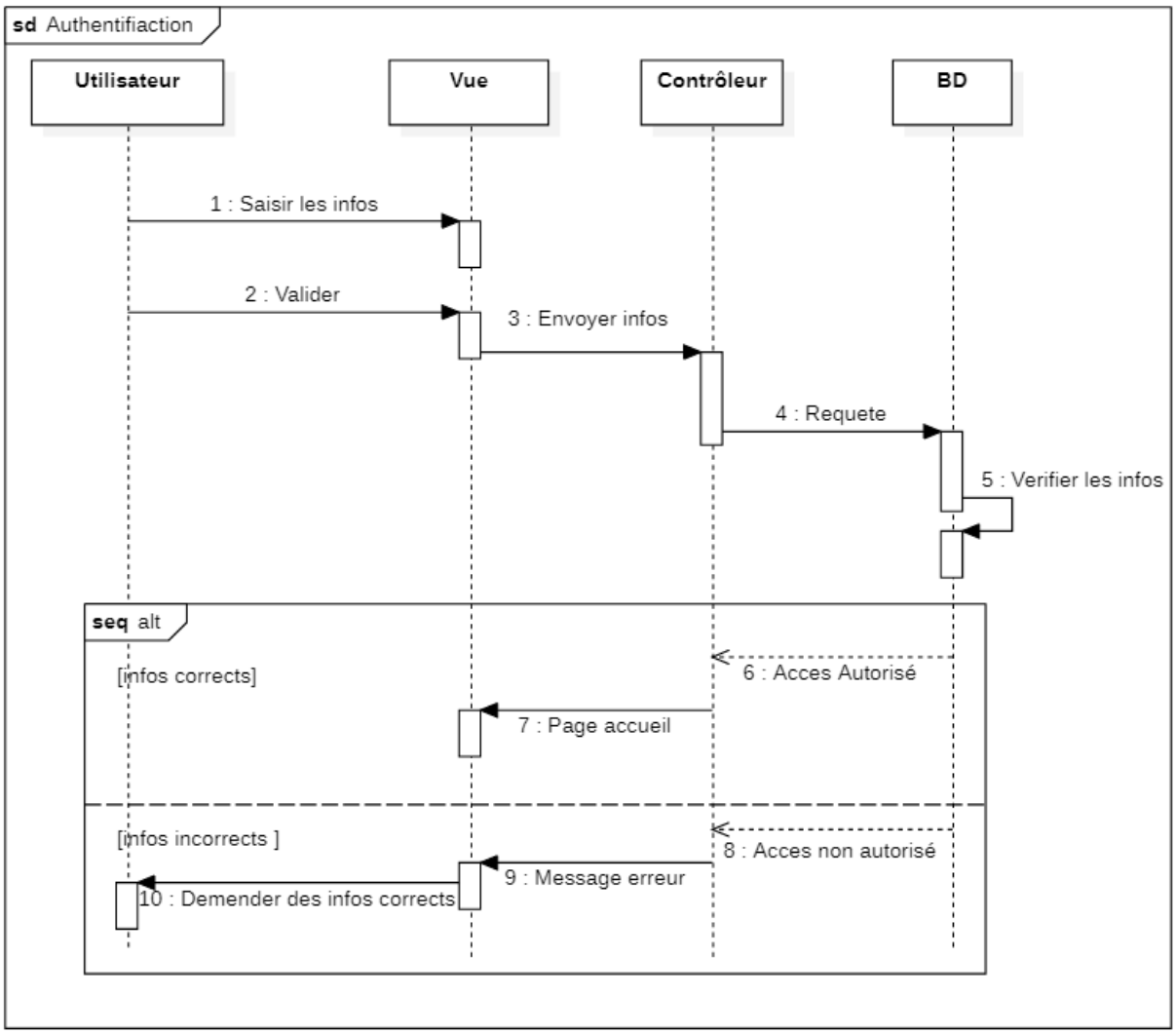


FIGURE 4.11 – Diagramme de séquence de cas d'utilisation S'authentifier

Scénario nominal de S'authentifier :

- L'utilisateur saisie et valide les information d'authentification.
- Le contrôleur envoie une requête à la BD pour vérifier l'utilisateur.
- Si les informations sont correctes, l'utilisateur est authentifié, sinon, message d'erreur est affiché.

4.3.3.3 Diagramme de séquence de cas d'utilisation Virement

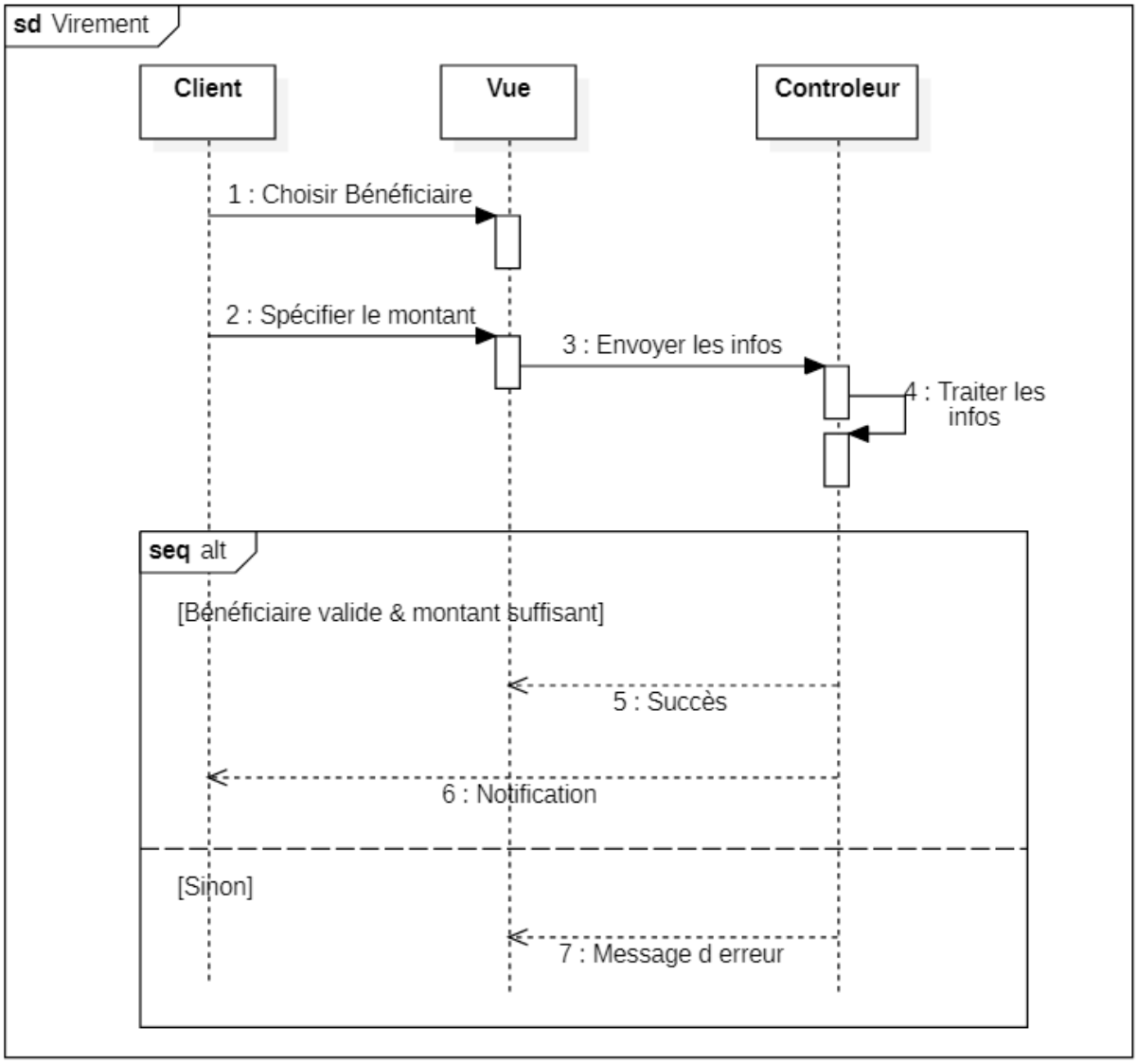


FIGURE 4.12 – Diagramme de séquence de cas d'utilisation Virement

Scénario nominal de Virement :

- Le client choisi le Bénéficiaire et spécifier le montant dans l'interface de virement.
- Le contrôleur traite les informations.
- Si le Bénéficiaire est valise et le montant est suffisant, le virement sera effectué est une notification est envoyer au client, sinon, un message d'erreur est afficher.

4.3.4 Diagramme de classes

Le diagramme de classes montre la structure interne du système.

Il fournit une représentation abstraite des objets système qui interagiront pour réaliser des cas d'utilisation. Le but d'un diagramme de classes est de modéliser les entités d'un système d'information, qui peuvent représenter toutes les informations finales gérées par le domaine. Ces informations sont structurées (regroupées en classes).

Après l'analyse précédente, nous avons obtenu le diagramme de classes, comme le montre la Figure suivante :

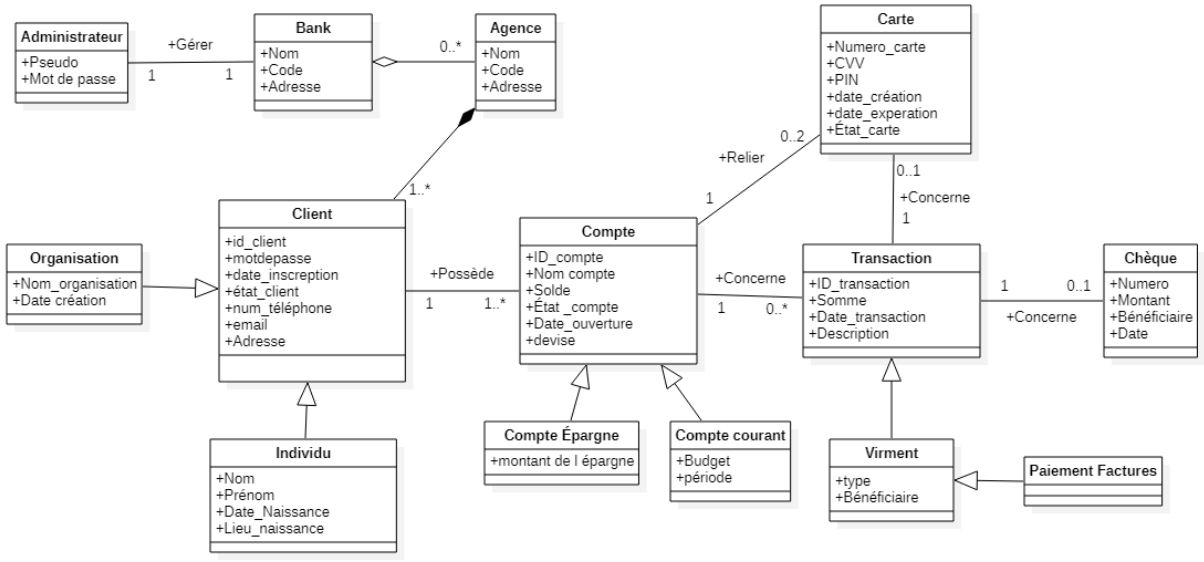


FIGURE 4.13 – Diagramme de classes

4.3.5 Architecture globale de la ligne de produit e-banking

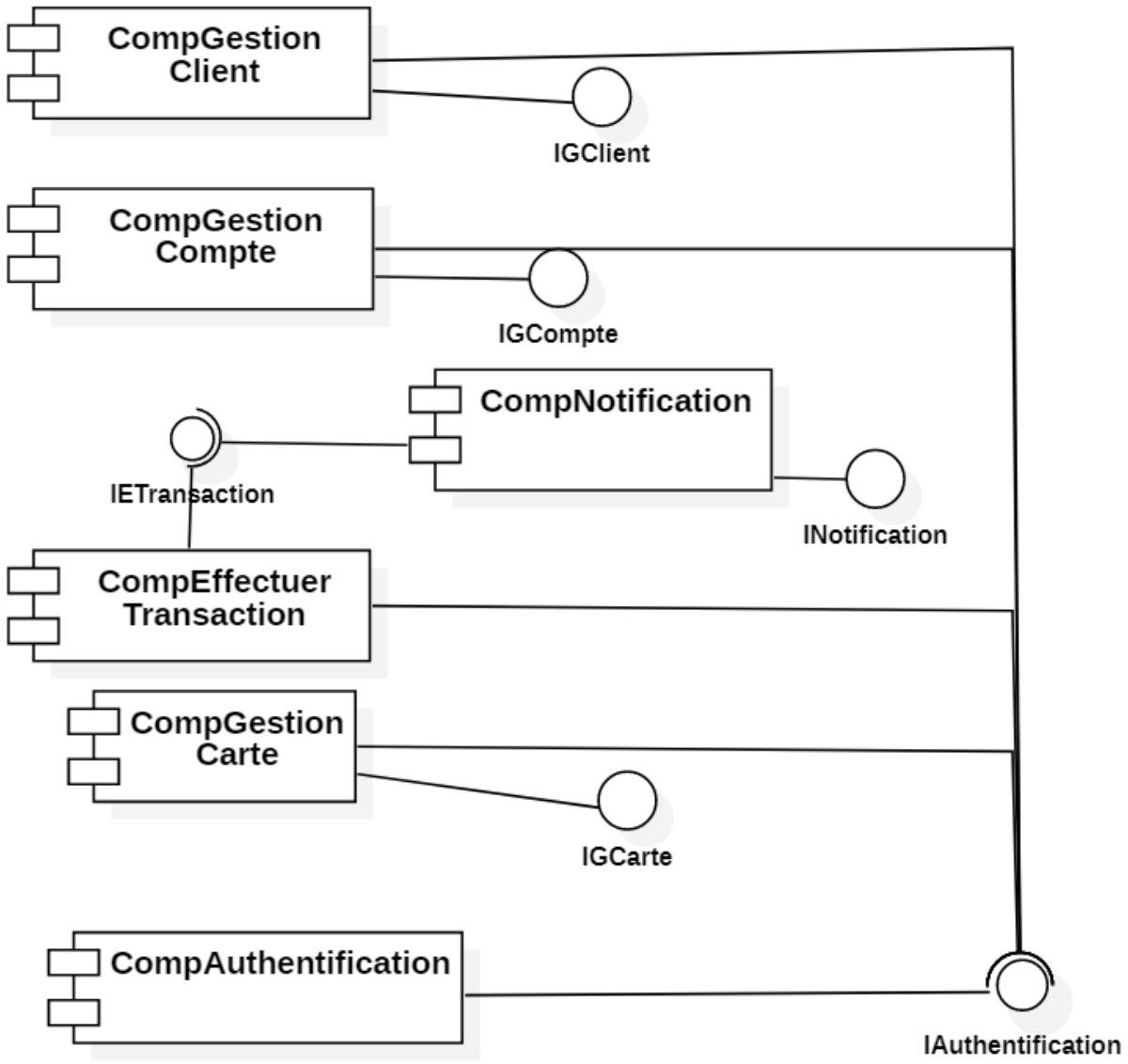


FIGURE 4.14 – Diagramme des composants globale

4.3.6 Raffinement des composants d'e-banking :

Après avoir défini l'architecture globale de l'e-banking, nous allons maintenant décrire une vue interne de certains de ses sous-composants composites pour donner un aperçu de leur structure.

4.3.6.1 Raffinement du Composant Gestion Client

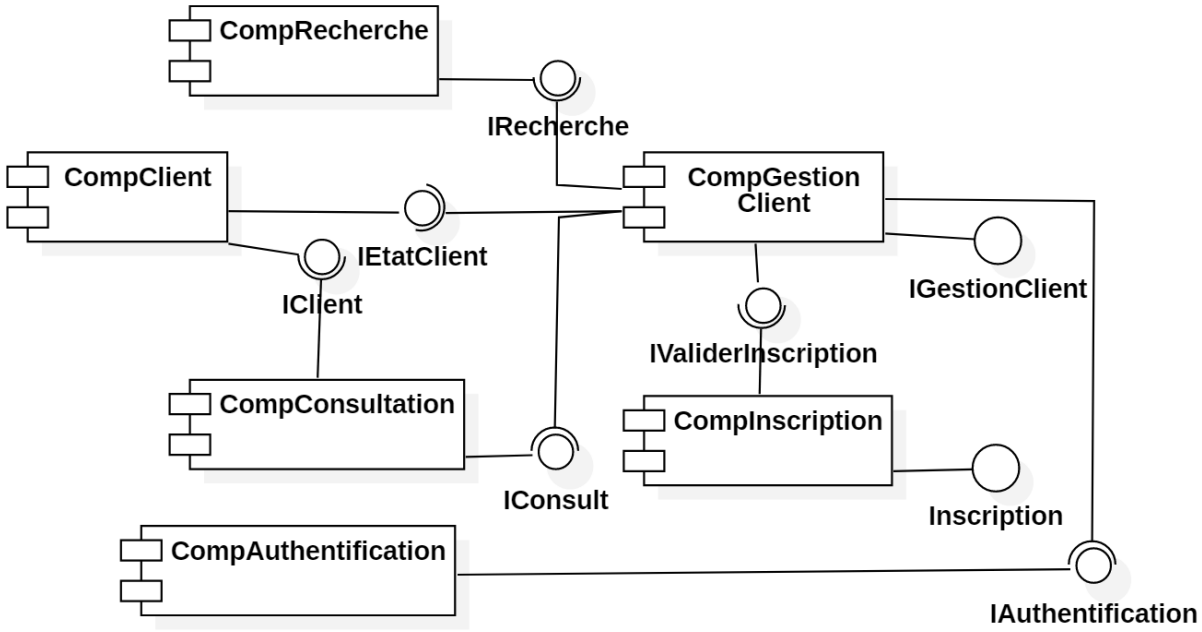


FIGURE 4.15 – Composant Gestion Client

Le composant Gestion de client est un composant composite qui se charge de :

- Recevoir les demandes d'inscription.
- Valider, Rejeter une demande d'inscription.
- Consultation des Clients (informations et la liste des clients).
- Rechercher un client.
- Bloquer ou débloquer un client.

Ces fonctions sont exécutées à l'aide des composants CompClient, CompConsultation, CompInscription et CompRecherche.

4.3.6.2 Raffinement du Composant Gestion Compte

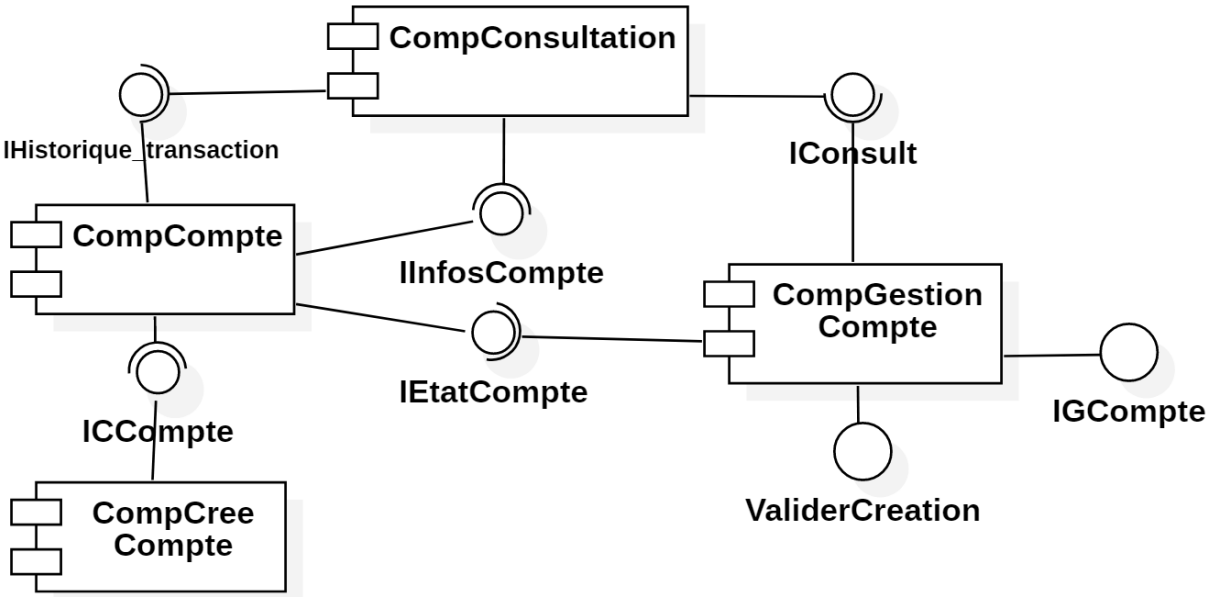


FIGURE 4.16 – Composant Gestion Compte

Le composant Gestion de Compte est un composant composite qui se charge de :

— Recevoir les demandes des création de compte.

— Valider, Rejeter une demande de création de compte.

— Consultation de Compte (informations sur le compte comme le solde le type...etc, et l'historique des transactions).

— Bloquer ou débloquer un compte.

Ces fonctions sont exécutées à l'aide des composants CompCompte, CompConsultation et CompCree Compte.

4.3.6.3 Raffinement du Composant Gestion Carte

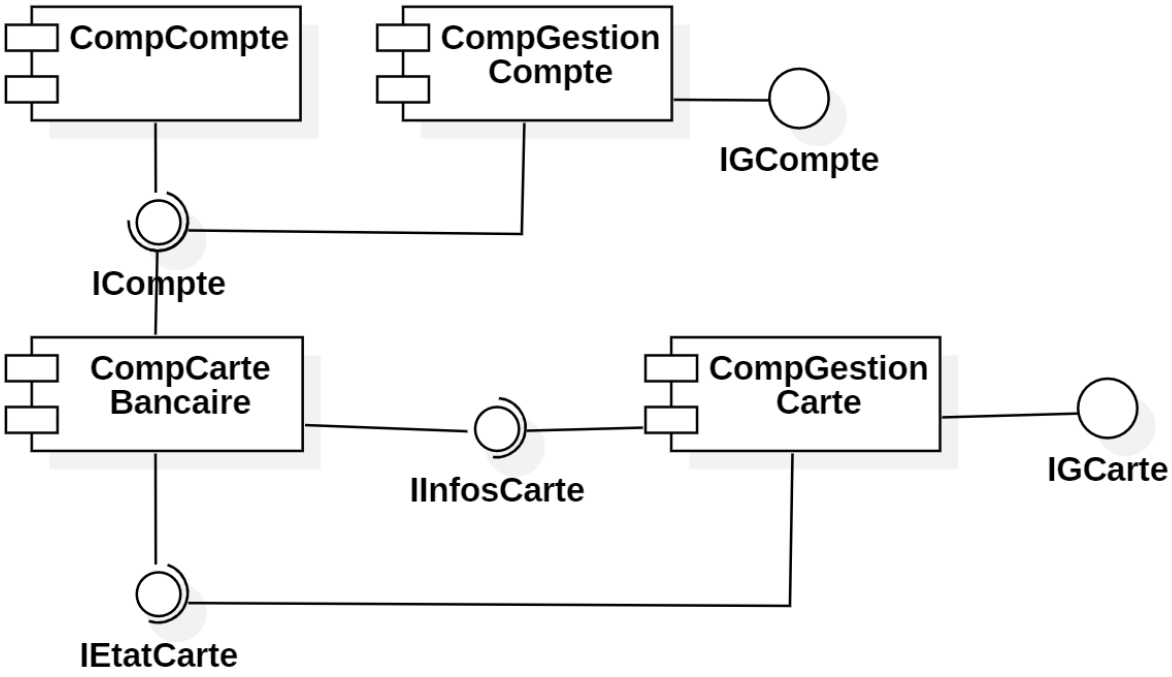


FIGURE 4.17 – Composant Gestion Carte

Le composant Gestion Carte est un composant composite qui se charge de :

— Recevoir les demandes des cartes bancaires.

— Valider, Rejeter une demande d'une carte bancaire.

— Modification de code PIN.

— Geler ou Activer une carte.

Ces fonctions sont exécutées à l'aide des composants CompCarte Bancaire , ainsi deux composants externes CompGestion Compte et CompCompte.

4.3.6.4 Raffinement du Composant Effectuer Transaction

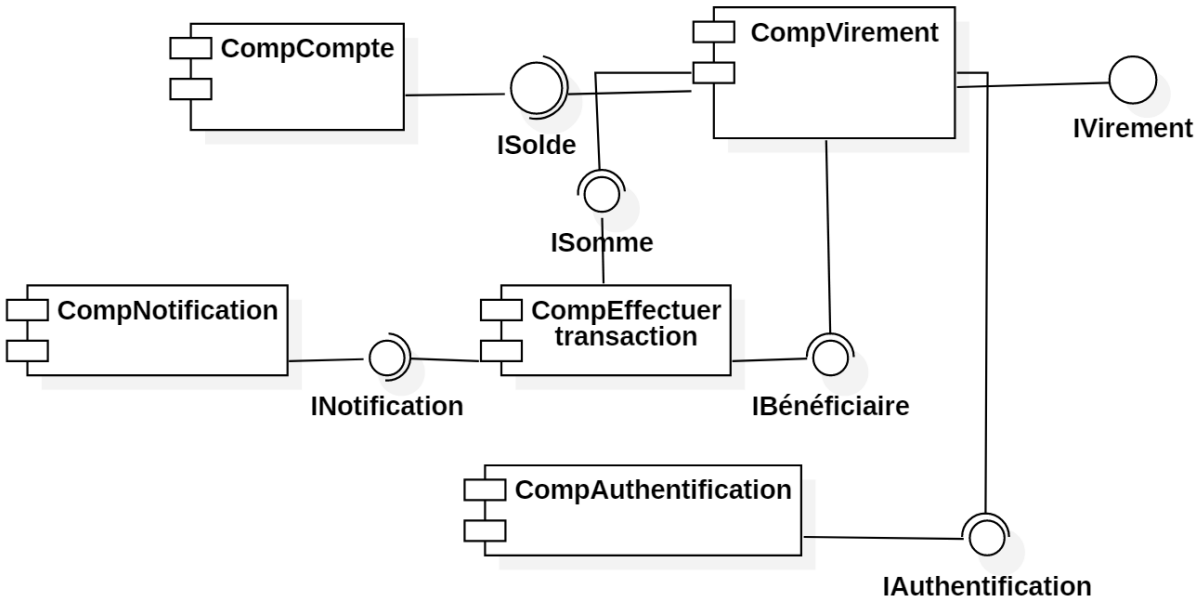


FIGURE 4.18 – Composant Effectuer Transaction

Le composant Gestion Effectuer Transaction est un composant composite qui se charge de :

— Recevoir des demandes d'envoyer d'argent.

— Valider, Rejeter une demande d'envoyer d'argent.

— Envoyer une Notification au Client soit dans le succès ou l'échec de l'envoi.

Ces fonctions sont exécutées à l'aide des composants CompEffectuer Transaction, CompNotification et CompCompte.

4.4 Réalisation du domaine

Après la modélisation de l'architecture, nous devons mettre en œuvre tous les composants de notre ligne se produit e-banking.
Ainsi, pour chaque nouvelle application, il vous suffit d'assembler les composants existants. S'il existe des besoins spécifiques des utilisateurs, ils doivent être pris en compte en implémentant de nouveaux composants qui leur correspondent, et ceux-ci doivent être enregistrés dans une base de composants réutilisable pour une réutilisation ultérieure si nécessaire.

4.5 Conclusion

Dans ce chapitre, nous avons présenté la conception de notre ligne de produits et le rôle de l'ontologie dans la représentation de notre Feature model, ensuite L'architecture de référence , ce qui nous permettra ensuite de dériver plusieurs types d'applications en fonction de besoins spécifiques.

Chapitre 5

Ingénierie d'application

5.1 Introduction

Dans le chapitre précédent, nous avons présenté la conception de notre solution. Ce chapitre va comporter une partie pour spécifier les ressources matérielles et logicielles utilisées et une deuxième partie pour la dérivation d'une application en utilisant notre ligne de produits.

5.2 Environnement logistique

5.2.1 Matériels utilisés

Notre application sera réalisée sur une machine dont les caractéristiques sont les suivantes :

- **Marque :** Asus.
- **Processeur :** intel(R) Celeron(R) CPU N3350 @ 1.10GHz.
- **Système d'exploitation :** Windows 10.
- **Ram :** 4 Go.

5.2.2 Environnement de développement

5.2.2.1 Technologies de développement

❖ **Java EE**

Java Enterprise Edition (Java EE), maintenant connu sous le nom de *Jakarta Enterprise Edition* (Jakarta EE). Il s'agit d'un ensemble de spécifications autour de *Java Standard Edition* (Java SE). Java EE fournit aux développeurs une plate-forme avec des fonctionnalités d'entreprise telles que l'informatique distribuée et les services Web.

Les applications Java EE s'exécutent généralement sur un environnement d'exécution de référence tel qu'un microserveur ou un serveur d'applications. Quelques exemples d'environnements qui utilisent Java EE incluent e-commerce, la comptabilité, les systèmes d'information bancaire [63].

❖ Apache Tomcat

Apache Tomcat [1] est une implémentation open source des technologies Java Servlet, JavaServer Pages, Java Expression Language et Java WebSocket. Ce serveur est particulièrement utilisé pour le déploiement d'applications web développées, son installation est facile, rapide et son utilisation pas très compliqué [64].

❖ HTML

HyperText MarkUp Language (HTML) [2], est un langage de description (et non pas d'un langage de programmation), il a fait son apparition dès 1991 lors du lancement du web, qui permet de réaliser l'hypertexte à base d'une structure de balisage [65].

❖ CSS

Cascading Style Sheets (CSS) [3], soit « feuilles de style en cascade ». Il a été créé en 1996 qui permet de mettre en forme les fichiers HTML ou XML [65].

❖ JavaScript

JavaScript [4] est un langage de script qui a été conçu à l'origine pour fonctionner avec le langage HTML . Cela signifie qu'il a été écrit spécifiquement pour faciliter l'ajout d'interactivité aux pages Web. En fait, JavaScript a été introduit pour la première fois en 1991, lors de la sortie de Netscape Navigator 1.0 [66].

❖ JQurey

jQuery [5] est une bibliothèque JavaScript rapide, petite et riche en fonctionnalités. Elle simplifie la traversée et la manipulation des documents HTML , la gestion des événements, l'animation et Ajax grâce à une API facile à utiliser qui fonctionne sur une multitude de navigateurs [67].

1. https://tomcat.apache.org/
2. https://developer.mozilla.org/en-US/docs/Web/HTML
3. https://developer.mozilla.org/en-US/docs/Web/CSS
4. https://developer.mozilla.org/en-US/docs/Web/JavaScript
5. https://api.jquery.com/

❖ SQL

Structured Query Language (SQL) est un langage de manipulation de bases de données mis au point dans les années 70 par *International Business Machines* (IBM) [68]. Il permet notamment :

- ✻ La manipulation des bases de données : création et suppression.
- ✻ La manipulation des tables : création, suppression, modification de la structure des tables.
- ✻ La gestion des droits d'accès aux tables : contrôle des données et validation des modifications.

5.2.2.2 Outils de programmation :

Pour pouvoir bien réaliser notre application nous avons opté pour quelque outils que nous allons définir ci-dessous :

NetBeans

NetBeans [6] est un environnement de développement intégré placé en open source par Java, Java ee . Apache NetBeans est bien plus qu'un éditeur de texte. Il met en évidence le code source syntaxiquement et sémantiquement, vous permet de refactoriser facilement le code, avec une gamme d'outils pratiques et puissants .

Apache NetBeans fournit des éditeurs, et des modèles pour aider à créer des applications en Java, PHP et de nombreux autres langages. [69].

PostgreSql

PostgreSQL [7] est un système de gestion de bases de données relationnelles et objet basé sur POSTGRES, Version 10 . Ce dernier a été développé à l'université de Californie au département des sciences informatiques de Berkeley.

PostgreSQL supporte une grande partie du standard SQL tout en offrant de nombreuses fonctionnalités modernes :

- ➢ requêtes complexes ;
- ➢ clés étrangères ;
- ➢ triggers ;
- ➢ intégrité transactionnelle ;

6. https://netbeans.apache.org/
7. https://www.postgresql.org/

➢ Contrôle de la concurrence en multiversion (MVCC) ;

PostgreSQL peut être utilisé, modifié et distribué librement, quel que soit le but visé, qu'il soit privé, commercial ou académique [70].

Protégé

Protégé [8] est un éditeur d'ontologie qui permet de construire une ontologie
pour un domaine de donnée, et peut être utilisé dans n'importe quel domaine où les concepts peuvent être modélisés en une hiérarchie des classes.
Protégé est aussi une plate-forme extensible, grâce au système de plug-ins, qui permet de gérer des contenus multimédias, interroger, évaluer et fusionner des ontologies, etc.
Protégé possède une interface utilisateur personnalisable et d'outils de visualisation qui permettent une interaction avec l'ontologie et ses relations [71].

5.3 Dérivation d'une application

5.3.1 Analyse d'application

L'application dérivée à partir de notre ligne de produit s'occupe de la majorité des fonctionnalités de feature model de domaine d'e-banking (Figures 4.4, 4.3, 4.6) de chapitre précédent avec des interfaces graphiques en Arabe, Français et Anglais, l'authentification des utilisateurs à l'aide d'un nom d'utilisateur et un mot de passe et utilise une base de données relationnelle.

5.3.2 Modélisation d'application

Le feature modèle de cette application après la sélection des fonctionnalités est le suivant :

8. `https://protege.stanford.edu/products.php`

5.3.2.1 Feature Model d'application Espace Client

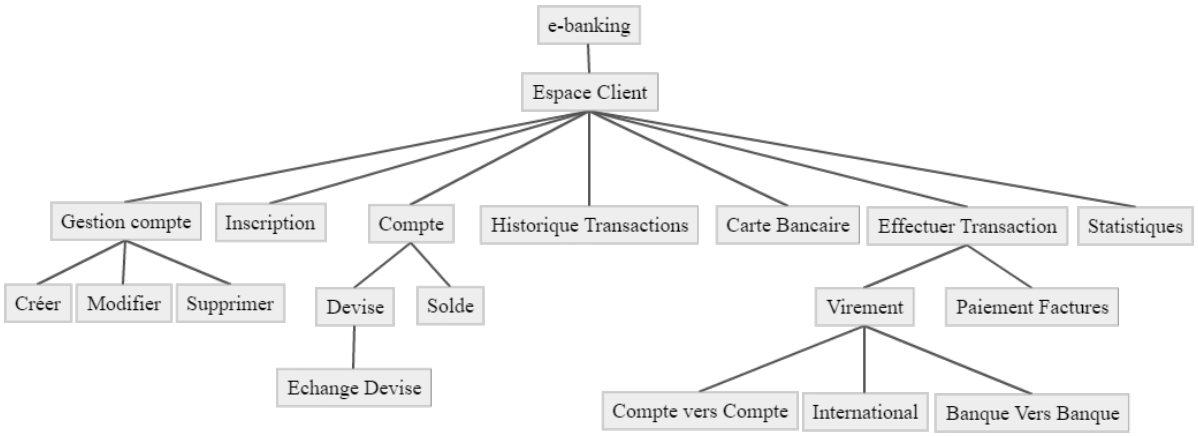


FIGURE 5.1 – Feature model application Espace client

Les contraintes :

Espace Client **requires** Authentification.

Effectuer Transaction **requires** Compte.

Créer Compte **excludes** Demander Compte.

Inscription **requires** Valider Inscription.

Carte Bancaire **requires** Compte.

Historique transactions **requires** Effectuer Transaction.

5.3.2.2 Feature Model d'application Espace Admin

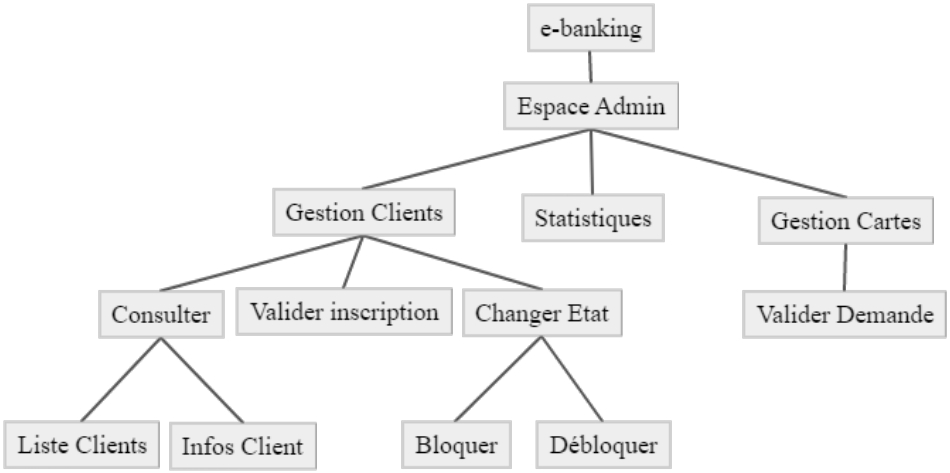


FIGURE 5.2 – Feature model d'application Espace Admin

Espace Admin **requires** Authentification.

Gestion Cartes **requires** Carte Bancaire.

Gestion Comptes **requires** Compte.

5.3.2.3 Feature Model d'application technique

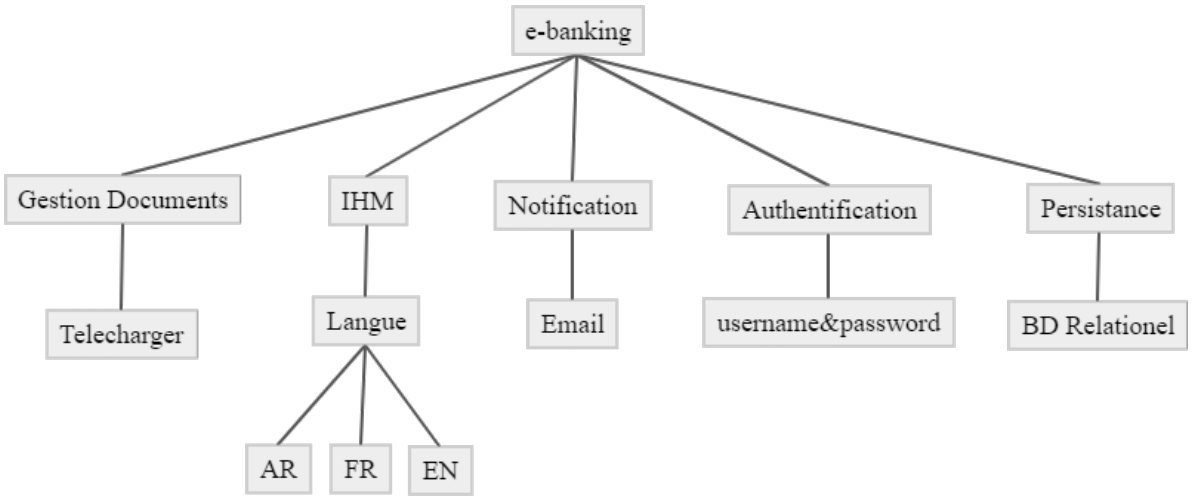


FIGURE 5.3 – Feature model d'application technique

5.3.3 Réalisation d'application

Les applications dérivées à partir de notre LDP sont des applications web qui suivent le modèle de conception MVC (Model, View, Controler), pour lesquelles nous utilisons Java EE qui permet la création d'applications web. Java EE facilite le développement d'applications Web déployées et exécutées sur des serveurs d'applications.

Pour le stockage des données on choisie PostgreSQL. Pour exécuter des applications Web Java EE, nous avons besoin d'un serveur d'applications. Nous avons choisi d'utiliser Tomcat car c'est un serveur léger, gratuit, multiplateforme et complet.

5.3.4 Les interfaces

Les figures suivantes présentent quelques interfaces de Notre ligne de produits :

5.3.4.1 Le configurateur

Les étapes de configuration d'une nouvelle application e-banking :

1. Sélectionner les features a inclure dans l'application.
2. Remplir les informations de la banque et l'administrateur.
3. Tester les applications Espace client et Espace Admin.
4. Générer un fichier WAR pour les deux applications, et exporter la BD comme code sql.

Le configurateur a pour objectif de générer un fichier de configuration d'après les features sélectionnes, ce fichier sera responsable a l'activation des features dans l'application web, et il aussi utiliser pour filtrer la création des tables de la Base de données.

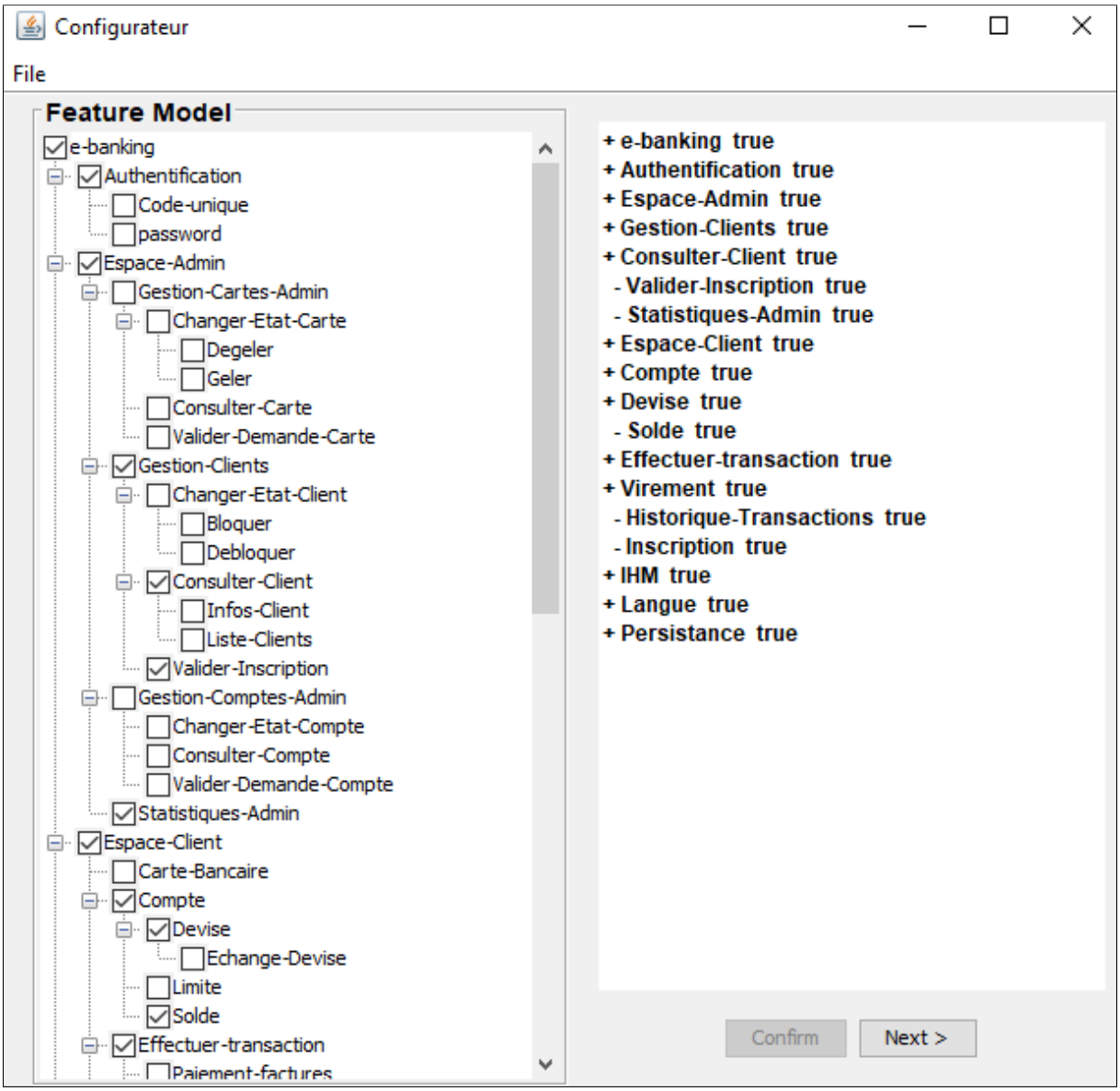


FIGURE 5.4 – Interface configurateur étape 1 de la configuration

Cette interface est pour le choix des features ; la représentation de feature model est crée a partir l'ontologie des features.

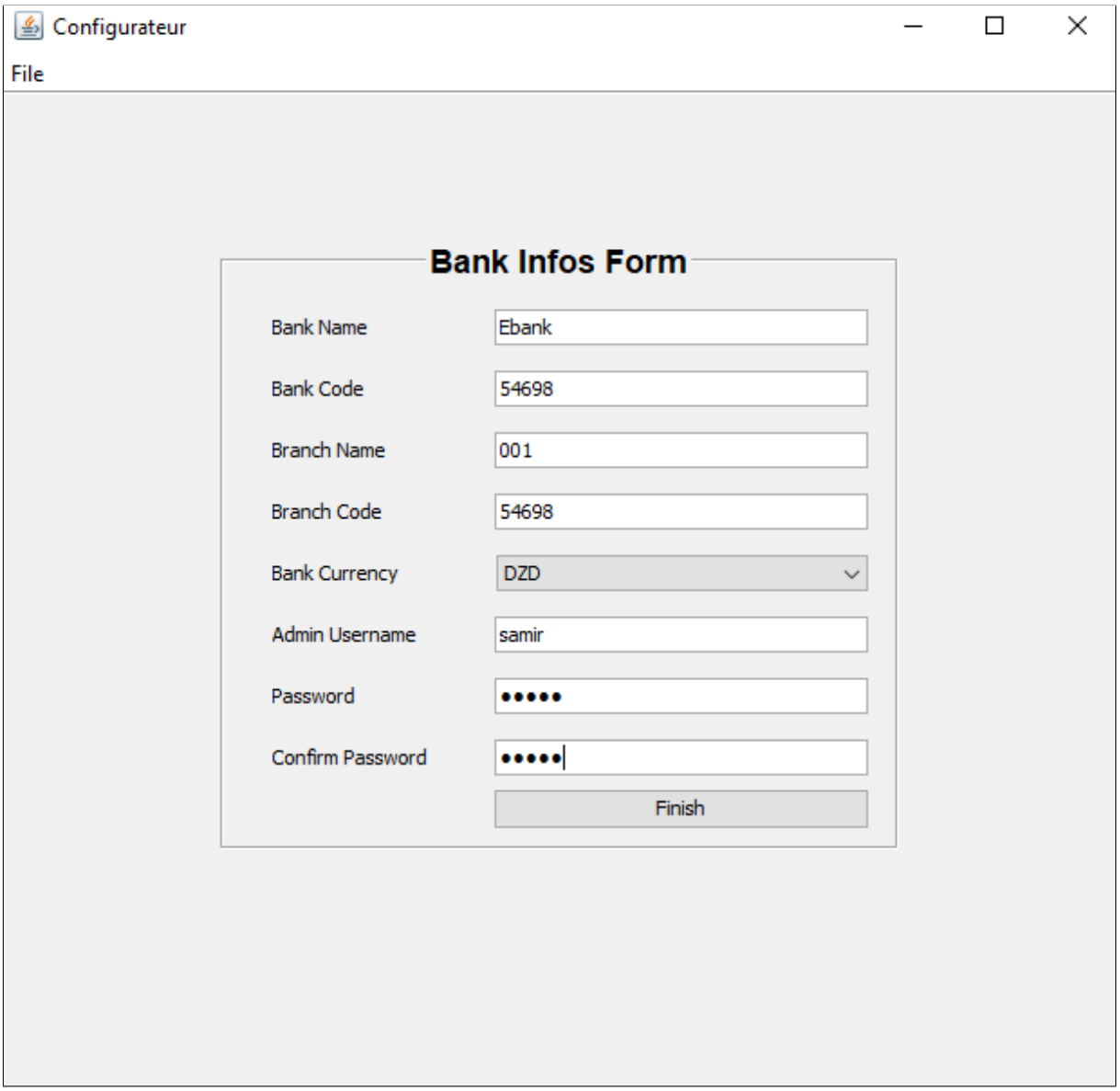


FIGURE 5.5 – Interface Configurateur étape 2 de la configuration

Cette interface représente la deuxième étape de configuration, remplir les informations de la banque et de l'administrateur.

5.3.4.2 Espace Client

Au début, quand un visiteur essaie d'accéder à notre application, via son navigateur Web, il se retrouve sur la page d'accueil du site la Figure 5.6.

FIGURE 5.6 – Page d'accueil

Si le visiteur n'est pas déjà inscrit, il peut créer un compte en cliquant sur lien "Register" ou les buttons "Open an Account " trouver dans la page d'accueil. Ces liens va le rédigé vers l'interface d'inscription ou il doit introduire ses informations comme le montre la Figure 5.7. Il doit également préciser s'il souhaite rejoindre en tant que Individuel ou en tant que organisation.

Sinon si il a déjà un compte il peur se connecter depuis l'interface de connexion qui est illustré dans la Figure 5.8.

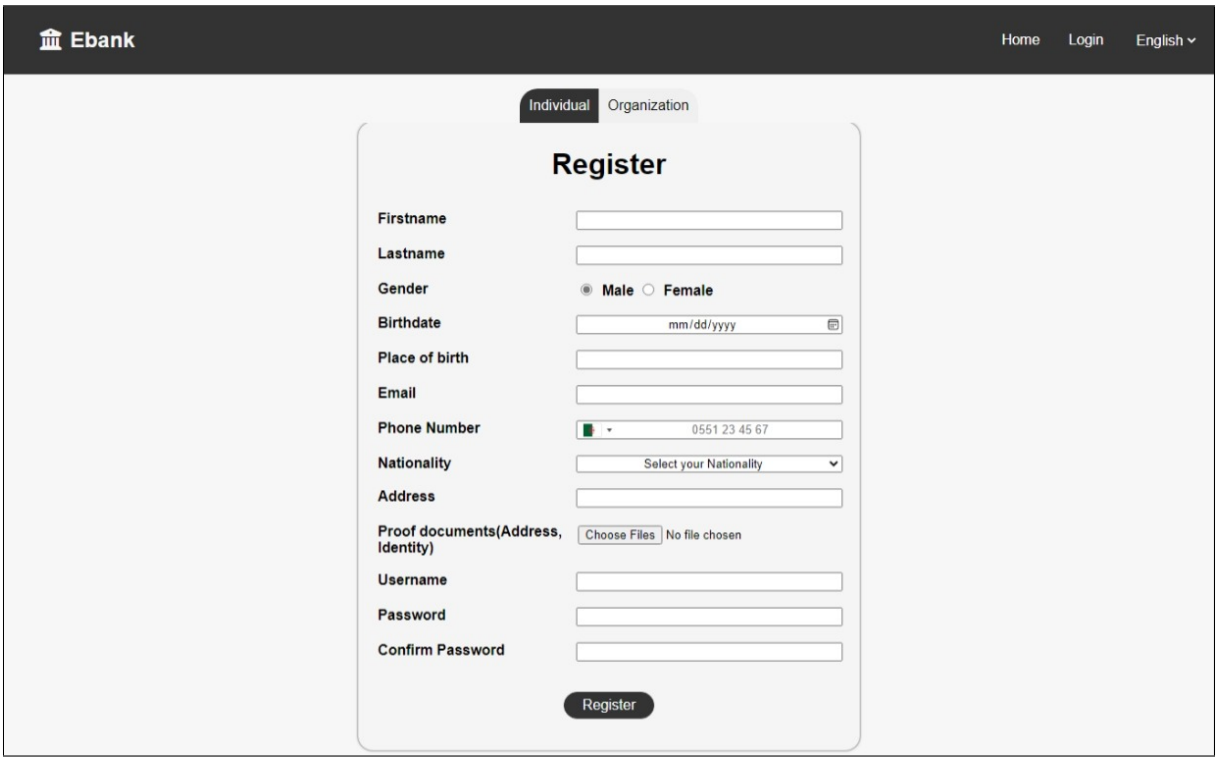


FIGURE 5.7 – Page d'inscription

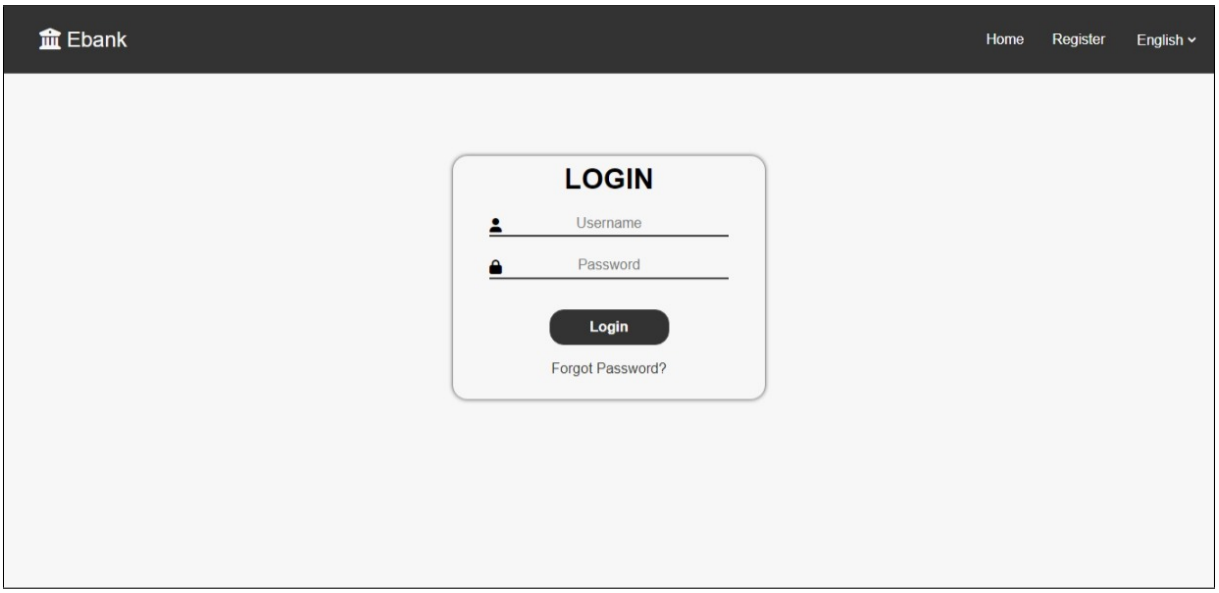


FIGURE 5.8 – Page de connexion

Une fois que le client se connecte, il est dirigé directement vers sa page des comptes ou il peut voir ses comptes, effectuer une transaction de compte sélectionné, voir ses informations, ses dix dernière transactions, les cartes liée a ce compte, créer un compte, demander une carte, etc. Comme représenté ci-dessous :

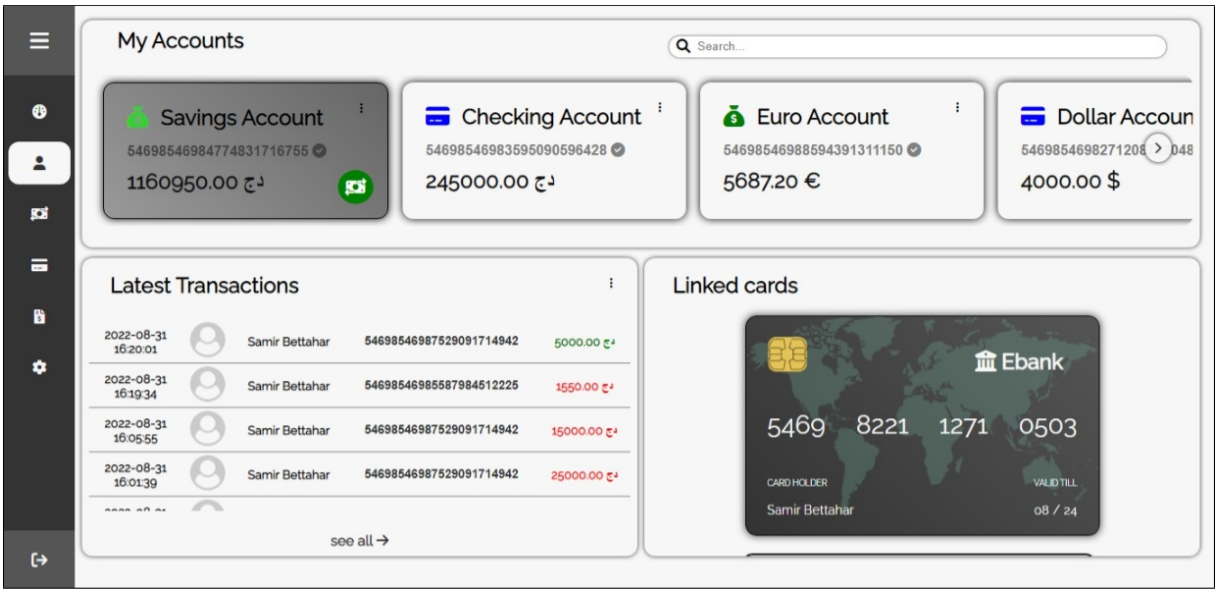


FIGURE 5.9 – Page des Comptes

La Figure 5.10 montre l'interface de tableau de bord (Dashboard) de client qui contient diffèrent statistiques sur ses comptes et ses activités.

FIGURE 5.10 – Dashboard

La Figure 5.11 montre l'interface de l'historique des transactions de toutes les comptes de client ou il peut filtrer les transactions par date ou rechercher directement une transaction avec la barre de recherche.

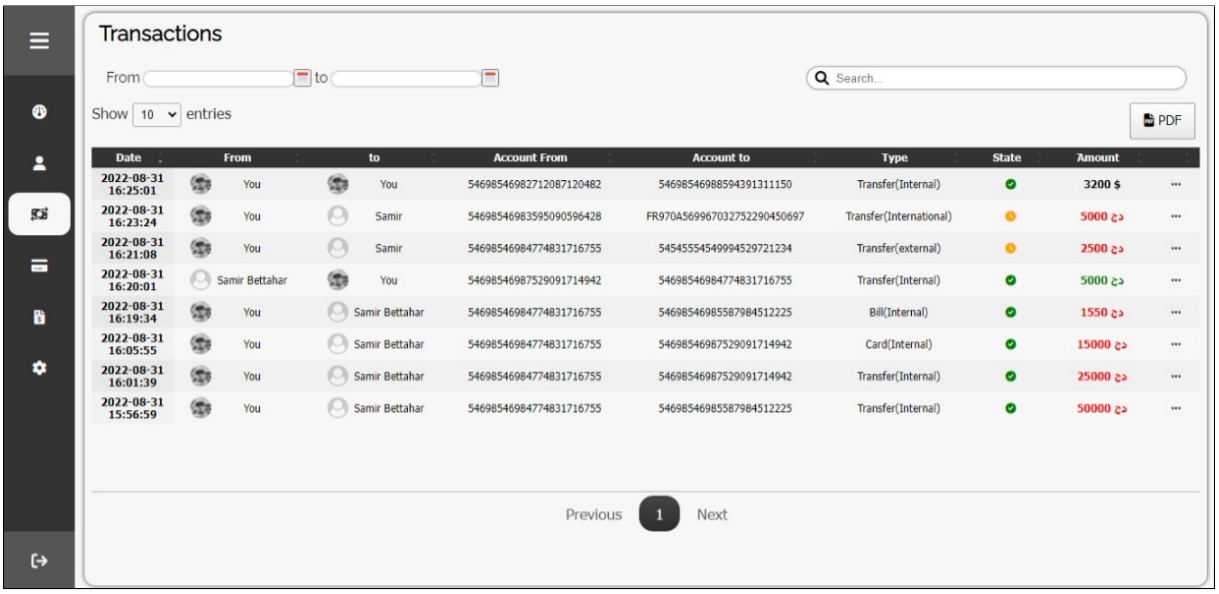


FIGURE 5.11 – Page des transactions

La Figure 5.12 montre l'interface des cartes bancaire de toutes les comptes de client ou il peut voir les informations de ses cartes, les dernières transactions effectuer par la carte sélectionnée , et il peut aussi réinstaller le code PIN de cette carte.

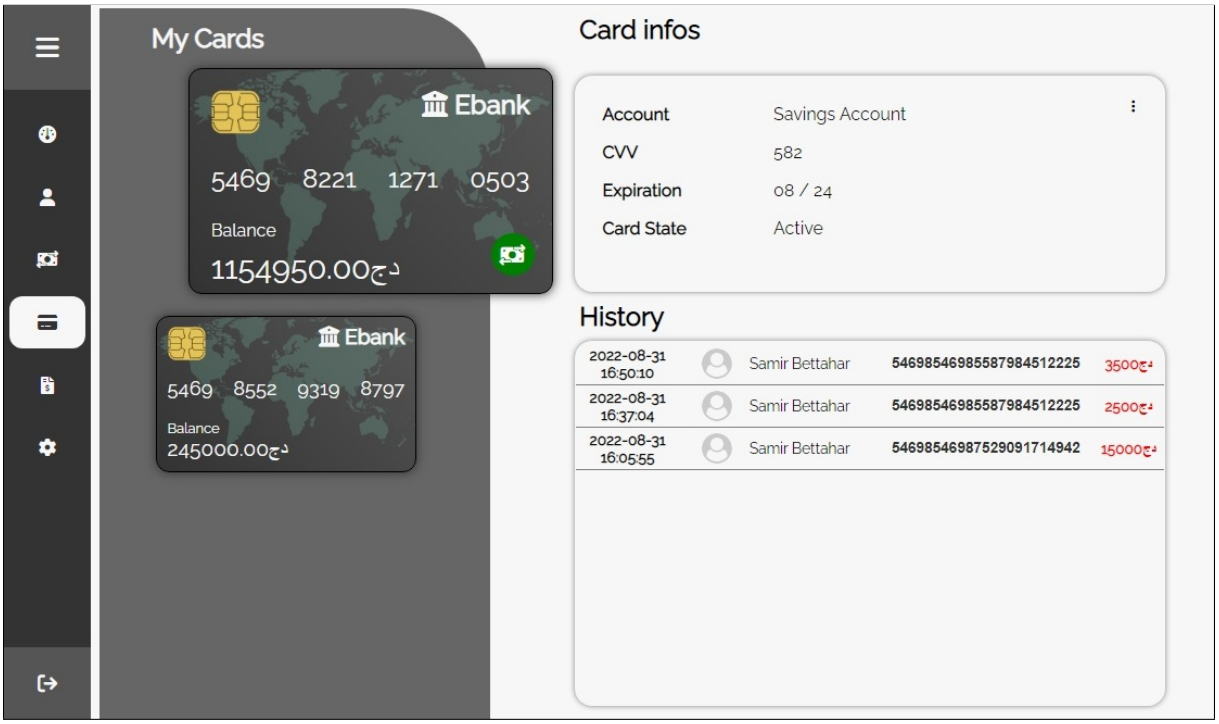


FIGURE 5.12 – Page des Carte Bancaires

La Figure 5.13 montre l'interface des factures paye par le client ou il peut filtrer les factures par date ou rechercher directement une facture avec la barre de recherche.

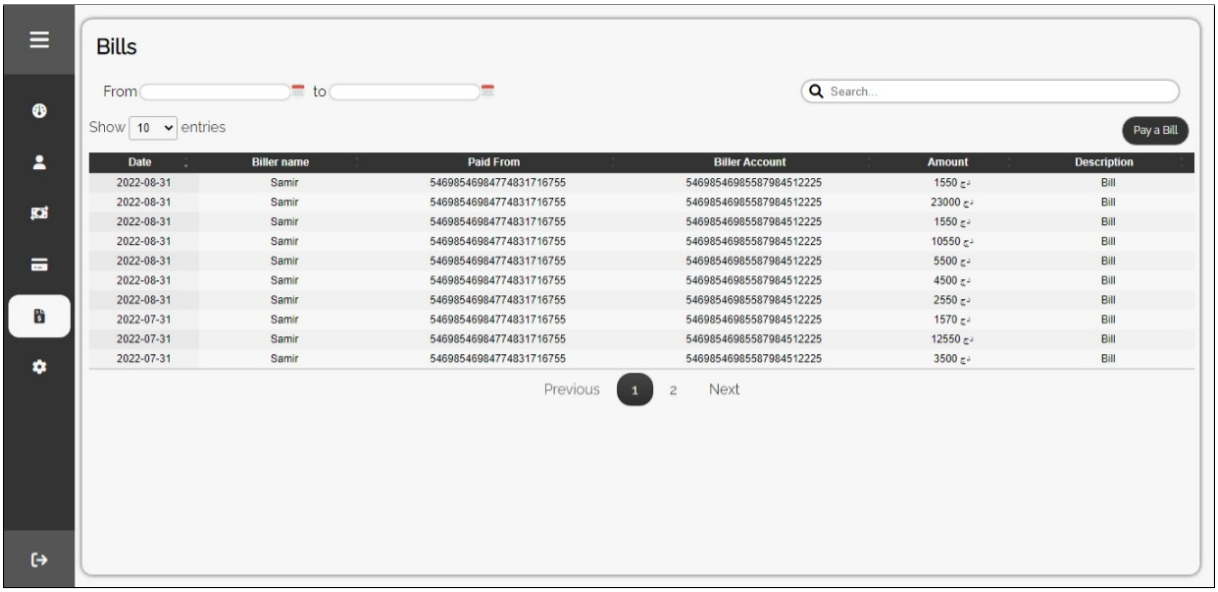


FIGURE 5.13 – Page des Factures

La Figure 5.14 montre l'interface des paramètres dont le client peut modifier ses informations (username, email, numéro de téléphone et son adresse), il peut changer son mot de passe ou la langue d'affichage (Arabe, Français ou Anglais).

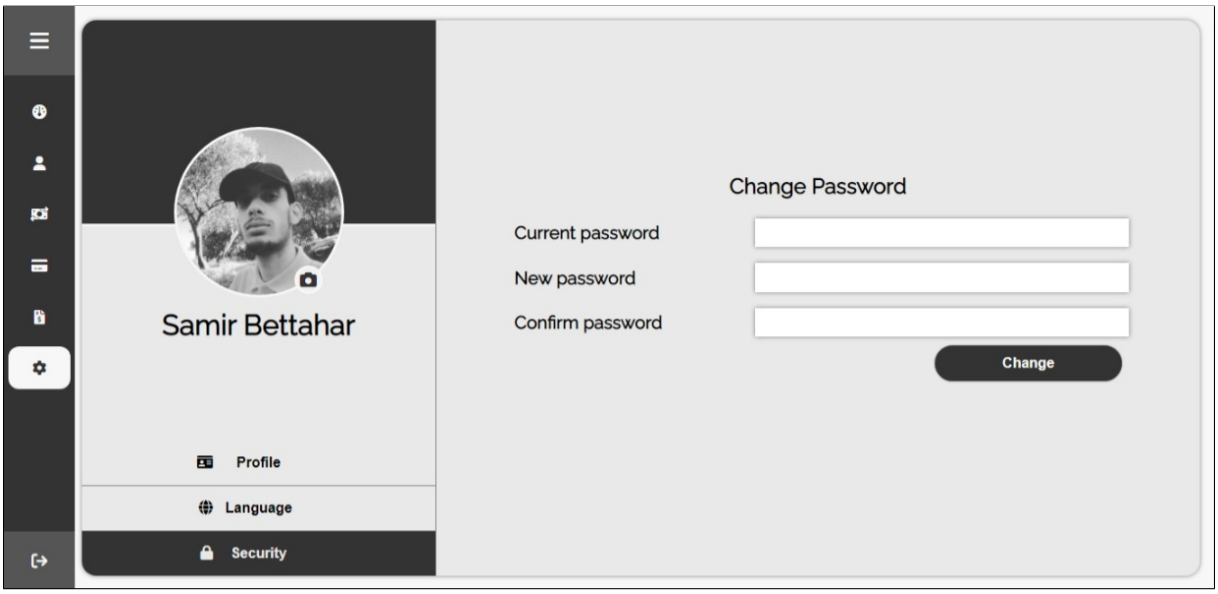


FIGURE 5.14 – Page des Paramètres

5.3.4.3 Espace Admin

Au début, quand l'administrateur essaie d'accéder à L'espace Admin, via un navigateur Web, il se retrouve sur la page de connexion La Figure 5.15.

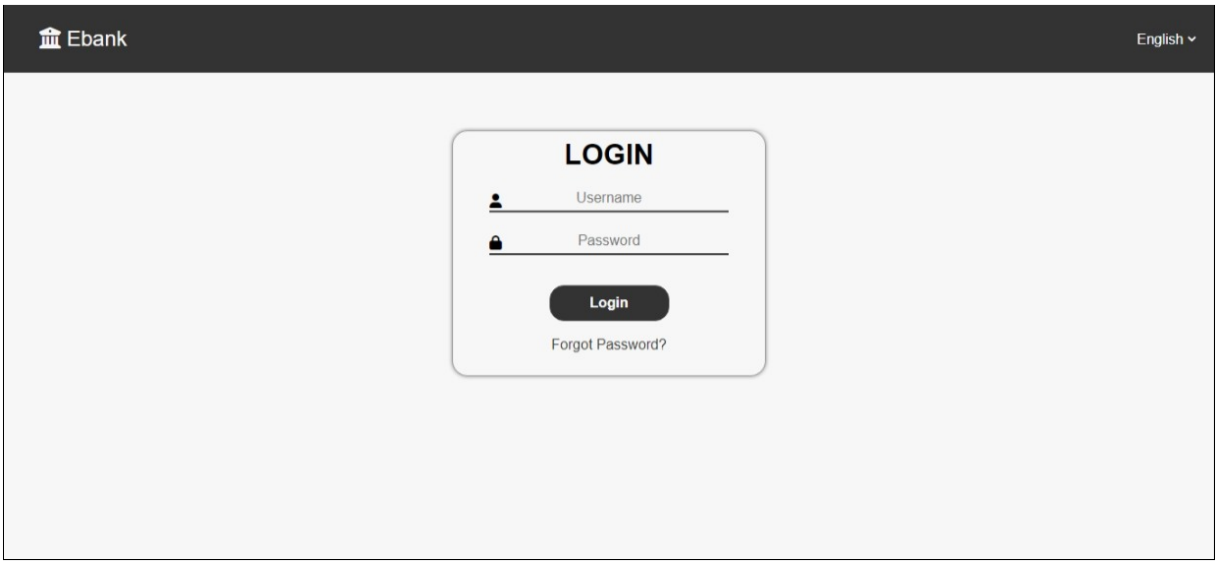


FIGURE 5.15 – Page de connexion admin

La Figure 5.16 montre l'interface de tableau de bord (Dashboard) de l'administrateur qui contient des statistiques générale de la banque.

FIGURE 5.16 – Dashboard Admin

La Figure 5.17 représente l'interface des demandes d'inscription des clients, ou l'administrateur examiner les demandes et les accepter ou les rejeter.

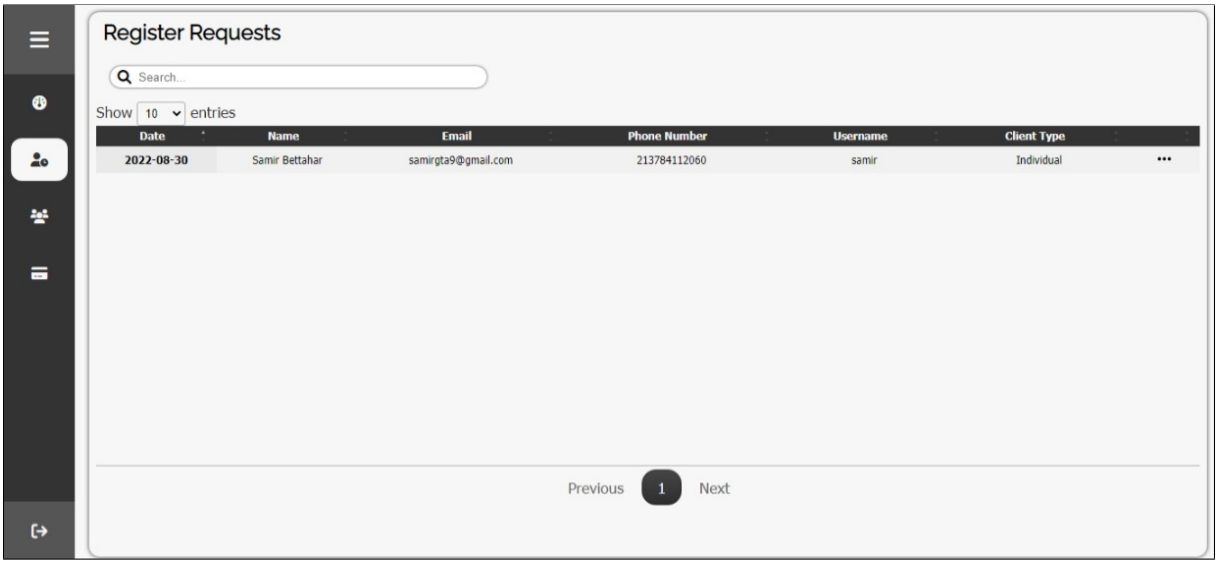


FIGURE 5.17 – Demande d'inscription

La Figure 5.18 représente la Liste des clients, ou l'administrateur peut voir les informations des clients et les bloquer ou les débloquer, et aussi il peut rechercher un client en utilisant la barre de recherche.

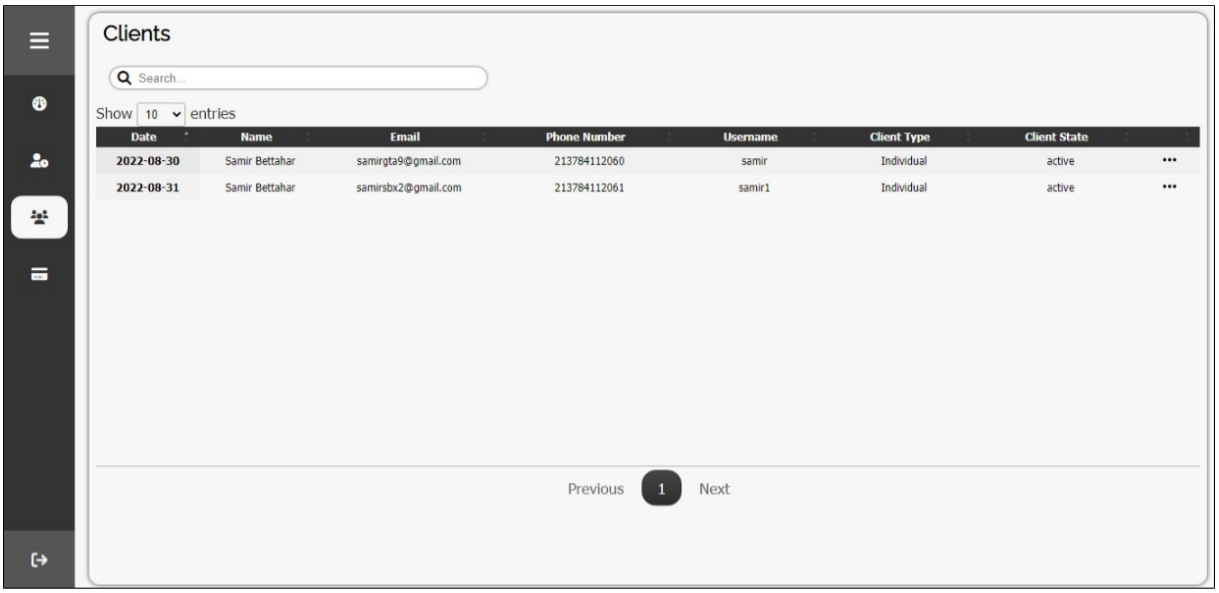


FIGURE 5.18 – Liste des clients

La Figure 5.19 représente les demandes des cartes effectuées par les clients, ou l'administrateur peut accepter ou rejeter une demande en fonction de nombre de cartes liées a un compte (la limite est 2 cartes par comptes), et aussi il peut rechercher une demande en utilisant la barre de recherche.

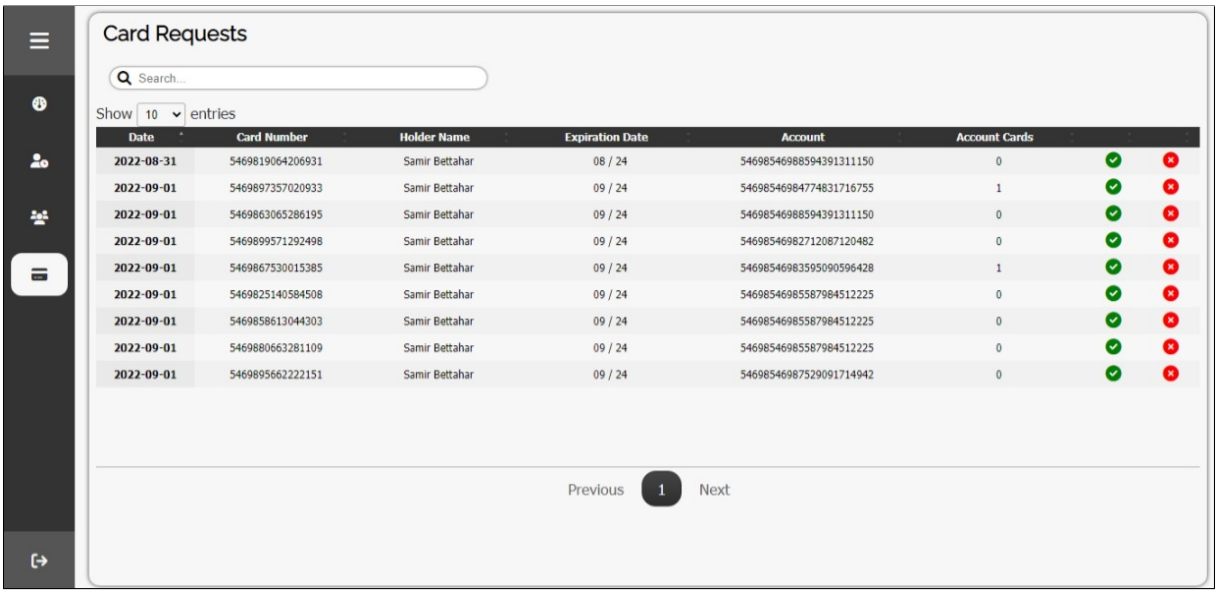


FIGURE 5.19 – Demande des cartes

5.4 Conclusion

Dans ce chapitre, nous avons vu le deuxième processus de développement de la LDP, l'ingénierie d'application, qui est répétée pour chaque nouvelle application (membre de la LDP). Dans notre projet de fin d'études, nous avons retenu le cas d'une application e-banking.

Conclusion et perspectives

Le travail présenté dans ce mémoire est le résultat d'environ six mois de recherche et de travail. Pendant ce temps, nous avons réalisé une étude bibliographique sur les approches des lignes de produits logiciels et les ontologies afin d'acquérir les concepts de base.

Le principe des lignes de produits lui-même est un défi car au lieu de penser à une application dans un domaine de recherche, nous devons penser à tout ce que nous pouvons avoir en tant que membres de notre LDP, un autre défi est l'utilisation d'une ontologie pour guider la dérivation des applications.

Après l'étude bibliographique, nous avons commencé à développer notre ligne de produits pour le domaine de l'e-banking. Ce processus est divisé en deux sous-processus : l'ingénierie de domaine et l'ingénierie d'application.

Nos contributions dans le premier processus sont classes comme suit : D'abord, nous avons commencé par la conception est la construction de l'ontologie pour déterminer les features a inclure dans le feature model. Ensuite, l'implimentation des composants logiciels réutilisables. Enfin nous avons développé le configurateur pour la sélection des features. Le deuxième processus est appliqué dans la dérivation d'une application pour montrer comment fonctionnent les résultats du premier sous processus.

D'une part, ce travail nous a permis d'entamer un nouveau champ de recherche, les lignes de produits logiciels. D'autre part, améliorer nos connaissances des ontologies et développer des applications web en Java. Nous espérons que notre travail sera à la hauteur de vos attentes.

Les Perspectives :

Les perspectives envisagées sont :

- Test des composants réutilisables en dérivant d'autres membres de la ligne de produits.
- Implémentation des variantes qui peuvent être utilisés par d'autres membres de notre ligne de produits.
- Automatisation de mapping de Feature model a notre ontologie.
- Amélioration de configurateur pour pouvoir directement modifier le feature model (Ajout, Suppression et modification des features).
- En outre, nous voudrions être en mesure de tester notre ligne de produits auprès des banques afin de constater la performance de notre solution en action.

Bibliographie

[1] H. MIA, M. A. RAHMAN et M. UDDIN, « E-Banking: Evolution, Status and Prospect », *Journal of ICMAB*, t. 35, jan. 2007.

[2] J. S. POULIN, *Measuring software reuse: principles, practices, and economic models*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[3] V. SUGUMARAN, S. PARK et K. C. KANG, *Introduction*, déc. 2006.

[4] F. van der LINDEN, K. SCHMID et E. ROMMES, *Software Product Lines in Action*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007.

[5] M. D. MCILROY, J. BUXTON, P. NAUR et B. RANDELL, « Mass-produced software components », in *Proceedings of the 1st international conference on software engineering, Garmisch Pattenkirchen, Germany*, 1968, p. 88-98.

[6] D. L. PARNAS, *On the Design and Development of Program Families*, 1976.

[7] J. M. NEIGHBORS, *Draco: A method for engineering reusable software systems*, 1989.

[8] W. FRAKES, R. PRIETO et C. FOX, *DARE: Domain analysis and reuse environment*, 1998.

[9] C. ATKINSON, C. BUNSE et J. BAYER, *Component-based product line engineering with UML*. Pearson Education, 2002.

[10] K. C. KANG, J. LEE et P. DONOHOE, *Feature-oriented product line engineering*, 2002.

[11] K. C. KANG, S. G. COHEN, J. A. HESS, W. E. NOVAK et A. S. PETERSON, « Feature-oriented domain analysis (FODA) feasibility study », Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, rapp. tech., 1990.

[12] F. van der LINDEN, *Software product families in Europe: the Esaps 'I&' Cafe projects*, 2002.

[13] P. CLEMENTS et L. NORTHROP, *Software product lines*. Addison-Wesley Boston, 2002.

[14] D. M. WEISS et C. T. R. LAI, *Software product-line engineering: a family-based software development process*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[15] U. AFZAL, T. MAHMOOD, A. H. KHAN et al., *Feature Selection Optimization in Software Product Lines*, 2020.

[16] M. RAATIKAINEN, J. TIIHONEN et T. MÄNNISTÖ, *Software product lines and variability modeling: A tertiary study*, 2019.

[17] C. JEAN, *Les lignes de produits logiciels Réutilisation et variabilité*, 2009.

[18] T. ZIADI et J.-M. JÉZÉQUEL, *Manipulation de Lignes de Produits Logiciels : une approche dirigée par les modèles*, 2005.

[19] K. POHL, G. BÖCKLE et F. VAN DER LINDEN, *Software product line engineering: foundations, principles, and techniques*. Springer, 2005, t. 1.

[20] J. RUMBAUGH, I. JACOBSON et G. BOOCH, *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[21] H. LACKNER, M. THOMAS, F. WARTENBERG et S. WEISSLEDER, « Model-Based Test Design of Product Lines: Raising Test Design to the Product Line Level », in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, IEEE, mars 2014, p. 51-60.

[22] A. GÓMEZ-PÉREZ et R. BENJAMINS, « Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods », in *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI?99) Workshop KRR5: Ontologies and Problem-Solving Methods: Lesson Learned and Future Trends*, Ontology Engineering Group ? OEG, t. 18, IJCAI et the Scandinavian AI Societies. CEUR Workshop Proceedings, août 1999. adresse : `https://oa.upm.es/6468/`.

[23] N. GUARINO, D. OBERLE et S. STAAB, « What Is an Ontology? », in mai 2009, p. 1-17.

[24] T. GRUBER, « Ontology », in *Encyclopedia of Database Systems*, Boston, MA : Springer US, 2009, p. 1963-1965.

[25] T. FRANCART et AUTHOR, *Ontologie, thesaurus et taxonomie sur le web de données*, `http://blog.sparna.fr/2013/12/07/ontologie-thesaurus-taxonomie-web-de-donnees/`, Accessed: 2022-03-06, juill. 2015.

[26] T. R. GRUBER, « A translation approach to portable ontology specifications », *Knowledge acquisition*, t. 5, n° 2, p. 199-220, 1993.

[27] S. STAAB et A. MAEDCHE, *Axioms are objects, too: Ontology engineering beyond the modeling of concepts and relations*. Univ., 2000.

[28] A. GÓMEZ-PÉREZ, « Ontology Evaluation », in *Handbook on Ontologies*, Berlin, Heidelberg : Springer Berlin Heidelberg, 2004, p. 251-273.

[29] N. GUARINO, *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*. IOS press, 1998, t. 46.

[30] G. VAN HEIJST, A. SCHREIBER et B. WIELINGA, « Using explicit ontologies in KBS development », *International Journal of Human-Computer Studies*, t. 46, n° 2, p. 183-292, 1997.

[31] N. GUARINO, M. CARRARA et P. GIARETTA, « Formalizing ontological commitment », in *AAAI*, t. 94, 1994, p. 560-567.

[32] R. MIZOGUCHI, J. VANWELKENHUYSEN et M. IKEDA, « Task ontology for reuse of problem solving knowledge », *Towards very large knowledge bases: Knowledge building & knowledge sharing*, t. 46, n° 59, p. 45, 1995.

[33] F. FÜRST, « Contribution à l'ingénierie des ontologies : une méthode et un outil d'opérationalisation », thèse de doct., École doctorale sciences et technologies de l'information et des matériaux (Nantes), 2004, 171 p.

[34] M. USCHOLD et M. GRUNINGER, « Ontologies: principles, methods and applications », *The Knowledge Engineering Review*, t. 11, n° 2, p. 93-136, 1996.

[35] M. FERNÁNDEZ-LÓPEZ, A. GÓMEZ-PÉREZ et N. JURISTO, « METHONTOLOGY: From Ontological Art Towards Ontological Engineering », in *Proceedings of the Ontological Engineering AAAI-97 Spring Symposium Series*, Ontology Engineering Group ? OEG, American Asociation for Artificial Intelligence, mars 1997.

[36] D. MCGUINNESS et F. van HARMELEN, *OWL Web Ontology Language Overview*, `https://www.w3.org/TR/2004/REC-owl-features-20040210/`, W3C Recommendation, Accessed: 2022-03-06, fév. 2004.

[37] D. BRICKLEY et R. GUHA, *RDF Schema 1.1*, "`https://www.w3.org/TR/2014/REC-rdf-schema-20140225/`, W3C Recommendation, Accessed: 2022-03-06, fév. 2014.

[38] Y. RAIMOND et G. SCHREIBER, *RDF 1.1 Primer*, `https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`, W3C Note, Accessed: 2022-03-06, juin 2014.

[39] L. RAMOS, « Semantic Web for manufacturing, trends and open issues: Toward a state of the art », *Computers & Industrial Engineering*, t. 90, p. 444-460, 2015. adresse : `%5Curl%7Bhttps://www.sciencedirect.com/science/article/pii/S0360835215004064%7D`.

[40] K. CZARNECKI, C. HWAN, P. KIM et K. KALLEBERG, « Feature models are views on ontologies », in *10th International Software Product Line Conference (SPLC'06)*, IEEE, 2006, p. 41-51.

[41] C. BU-QING, L. BING et X. QI-MING, « A process-driven and ontology based software product line variability modeling approach », in *2009 Eighth International Conference on Grid and Cooperative Computing*, IEEE, 2009, p. 385-390.

[42] L. A. ZAID, F. KLEINERMANN et O. DE TROYER, « Applying semantic web technology to feature modeling », in *Proceedings of the 2009 ACM symposium on Applied Computing*, 2009, p. 1252-1256.

[43] R. D. A. FALBO, G. GUIZZARDI et K. C. DUARTE, « An ontological approach to domain engineering », in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, 2002, p. 351-358.

[44] X. PENG, W. ZHAO, Y. XUE et Y. WU, « Ontology-based feature modeling and application-oriented tailoring », in *International Conference on Software Reuse*, Springer, 2006, p. 87-100.

[45] D. DAHDOUBI, « L'E-BANKING ET SON IMPACT SUR L'ENVIRONNEMENT DE LA BANQUE », thèse de doct., ABDELHAMID BEN BADIS – MOSTAGANEM, 2019, p. 26-38. adresse : http://e-biblio.univ-mosta.dz/handle/123456789/12959.

[46] H. CHRISTIANSEN, « Electronic finance: economics and institutional factors », *Occasional Paper*, t. 2, 2001.

[47] H. MIA, M. A. RAHMAN et M. UDDIN, « E-Banking: Evolution, Status and Prospect », *Journal of ICMAB*, t. 35, jan. 2007.

[48] K. A. ANNAN, « E-Commerce and Development Report 2002 », in *United Nations Conference on Trade and Development, UNCTAD Secretraiat, UN New York and Geneva, CD version*, 2002.

[49] O. LUSTSIK, « Can E-Banking Services be Profitable? », *SSRN Electronic Journal*, 2011.

[50] R. DEWAN et A. SEIDMANN, « Current issues in e-banking: introduction », en, *Commun. ACM*, t. 44, n° 6, p. 31-57, juin 2001.

[51] I. DRIGĂ et C. ISAC, « E-banking services–features, challenges and benefits », *Annals of the University of Petrosani. Economics*, t. 14, p. 49-58, 2014.

[52] A. CHOVANOVÁ, « Forms of Electronic Banking », *BIATEC*, t. XIV, p. 22-25, 2006.

[53] S. J. BARNES et B. CORBITT, « Mobile banking: concept and potential », *International journal of mobile communications*, t. 1, n° 3, p. 273-288, 2003.

[54] S. LIAO, Y. P. SHAO, H. WANG et A. CHEN, « The adoption of virtual banking: an empirical study », *International journal of information management*, t. 19, n° 1, p. 63-74, 1999.

[55] T. QURESHI, M. ZAFAR et M. KHAN, « Customer Acceptance of Online Banking in Developing Economies », *Journal of Internet Banking and Commerce*, t. 13, p. 1-9, avr. 2008.

[56] J. LOW et M. ABDUL, « Internet banking-benefits and challenges in an emerging economy », *International Journal of Research in Business Management*, t. 1, p. 19-26, 2013.

[57] D. SINGHAL et V. PADHMANABHAN, « A study on customer perception towards internet banking: Identifying major contributing factors », *Journal of Nepalese business studies*, t. 5, nº 1, p. 101-111, 2008.

[58] S. BAHL, « E-banking: Challenges & policy implications », *International Journal of Computing & Business Research*, p. 229-6166, 2012.

[59] S. M. NSOULI et A. SCHAECHTER, « Les enjeux de la banque électronique », *Finances & Développement / Septembre*, p. 48-51, 2002.

[60] S. COOK, C. BOCK, P. RIVETT et al., « Unified Modeling Language (UML) Version 2.5.1 », Object Management Group (OMG), Standard, déc. 2017. adresse : https://www.omg.org/spec/UML/2.5.1.

[61] J. B. F. FILHO, O. BARAIS, B. BAUDRY, W. VIANA et R. M. C. ANDRADE, « An approach for semantic enrichment of software product lines », in *Proceedings of the 16th International Software Product Line Conference on - SPLC '12 -volume 1*, New York, New York, USA : ACM Press, 2012, p. 188. adresse : http://dl.acm.org/citation.cfm?doid=2364412.2364444.

[62] M. A. MUSEN, « The protégé project: a look back and a look forward », *AI Matters*, t. 1, nº 4, p. 4-12, 2015. adresse : https://doi.org/10.1145/2757001.2757003.

[63] S. JAISWAL, *Java EE | Java Enterprise Edition - Javatpoint — javatpoint.com*, https://www.javatpoint.com/java-ee, [Accessed 01-Sep-2022], 2007.

[64] A. T. PROJECT, *Apache Tomcat*, https://tomcat.apache.org/, [Accessed 01-Sep-2022], 2018.

[65] M. NEBRA, *Réalisez votre site web avec HTML 5 et CSS 3*. Editions Eyrolles, 2017.

[66] I. PRICE-EVANS, *What is JavaScript?*, https://www.snapt.net/glossary/what-is-javascript, mai 2022.

[67] J. F. .-. JS.FOUNDATION, *jQuery API Documentation — api.jquery.com*, https://api.jquery.com/, [Accessed 01-Sep-2022], 2019.

[68] C. SOUTOU, *Apprendre SQL avec MySQL: avec 40 exercices corrigés*. Editions Eyrolles, 2006.

[69] TECHOPEDIA, *What is Netbeans? - definition from Techopedia*, août 2011. adresse : %5Curl%7Bhttps://www.techopedia.com/definition/24735/netbeans%7D.

[70] P. G. D. GROUP, *PostgreSQL: The World's Most Advanced Open Source Relational Database*, https://www.postgresql.org/, sept. 2022.

[71] M. A. MUSEN, « The protégé project: a look back and a look forward », *AI Matters*, t. 1, nº 4, p. 4-12, 2015. adresse : https://doi.org/10.1145/2757001.2757003.