

UNIVERSITE SAAD DAHLEB DE BLIDA

Faculté des sciences

Département d'informatique



MEMOIRE DE MASTER

En Informatique

Option : Ingénierie du Logiciel

THÈME :

**Une ligne de produits dynamique pilotée par les
ontologies pour le développement des plateformes
Healthcare auto-adaptative.**

Réalisé par :

AIT SAADI

Lilia

CHABLI Hounaida

Encadré par :

Mme LAHIANI Nesrine

Mme GUESSOUM Dalila

Juin 2023

Remerciements

Au nom d'ALLAH le miséricordieux et le clément

Nous remercions nos parents les plus chères au monde pour leurs sacrifices et leur encouragement tout au long de notre scolarité.

Nous remercions nos fidèles sœurs, frères et amies, nous exprimons toute notre gratitude à notre promotrice, Madame LAHIANI Nesrine, et co-promotrice Madame GUESSOUM Dalila pour les critiques et les conseils hautement fructueux, pour la confiance et le soutien qu'elles nous ont accordé tout au long de ce mémoire.

Nous adressons également nos remerciements aux membres du jury pour avoir accepté d'examiner et de porter leur jugement sur notre travail.

Résumé

La réutilisation de logiciels est une problématique persistante et fondamentale dans le domaine du génie logiciel depuis ses débuts. Son objectif principal est de réduire les coûts de développement et de maintenance en favorisant l'utilisation d'éléments logiciels existants. Actuellement, les approches telles que les lignes de produits logiciels et l'architecture logicielle permettent d'atteindre ces objectifs de manière conjointe.

Dans le cadre de notre projet de fin d'étude, nous visons à développer une ligne de produits dynamique dans le domaine de e-health et à identifier des composants réutilisables. Cette ligne de produits permettra ultérieurement de générer différentes applications en fonction des besoins des utilisateurs.

Un autre aspect de notre projet consiste à construire une ontologie de domaine, utilisée pour représenter le modèle de fonctionnalité de e-health. Ce modèle détermine les fonctionnalités communes ainsi que les variations des logiciels dans ce domaine. Par ailleurs, nous adoptons une approche orientée composants pour modéliser l'architecture, ce qui améliore la réutilisation des composants.

En somme, notre objectif est de promouvoir la réutilisation des logiciels en développant une ligne de produits dynamique pour les plateformes healthcare, en construisant une ontologie de domaine et en adoptant une architecture basée sur des composants.

Mots clés :

Ligne de produits logiciel, variabilité, ontologie, diagramme de caractéristique, e-health, adaptation.

Abstract

Software reuse has been a fundamental and recurring problem in software engineering since its inception. Its main objective is to reduce software development and maintenance costs by facilitating the reuse of existing software components. Currently, the approach of software product lines and software architecture enables the simultaneous achievement of these objectives.

The objectives of our final research project include the development of an e-health dynamic product line and the identification of reusable components. This dynamic product line will later allow the generation of various applications based on user requirements. Additionally, we aim to construct a domain ontology that represents the feature model of e-health, determining the common functionalities and variabilities of software in this domain. The architecture is modeled following component-based principles to enhance reuse.

In summary, our research project aims to address the challenges of software reuse by developing an e-health product line, constructing a domain ontology, and adopting a component-oriented architecture.

Keywords :

Software product line, variability, ontology, feature model, e-health, adaptation.

ملخص

إعادة استخدام البرمجيات هو قضية مستمرة وأساسية في مجال هندسة البرمجيات منذ بداياتها. الهدف الرئيسي لهذا الأمر هو تقليل تكاليف التطوير والصيانة من خلال تشجيع استخدام عناصر برمجية موجودة بالفعل. حالياً، تسمح النهجات مثل خطوط المنتجات البرمجية وهندسة البرمجيات المعمارية بتحقيق هذه الأهداف بشكل مشترك.

في إطار مشروعنا للبحث النهائي، نهدف إلى تطوير خط منتجات في مجال الرعاية الصحية الإلكترونية وتحديد المكونات القابلة لإعادة الاستخدام. ستسمح هذه الخطوط لاحقاً بإنشاء تطبيقات مختلفة وفقاً لاحتياجات المستخدمين.

جانب آخر من مشروعنا يتمثل في بناء مجال لغة المعلومات، يستخدم لتمثيل نموذج السمات الوظيفية للرعاية الصحية الإلكترونية. يحدد هذا النموذج الوظائف المشتركة والاختلافات في البرمجيات في هذا المجال. بالإضافة إلى ذلك، نعتمد نهجاً مركزاً على المكونات لنمذجة الهندسة المعمارية، مما يعزز إعادة استخدام المكونات.

في المجمل، هدفنا هو تعزيز إعادة استخدام البرمجيات من خلال تطوير خط منتجات للرعاية الصحية الإلكترونية، وبناء مجال لغة المعلومات، واعتماد هندسة معمارية مبنية على المكونات.

الكلمات الرئيسية:

خطوط منتجات برمجية، التباين، لغة المعلومات، نموذج السمات الوظيفية، الرعاية الصحية الإلكترونية، تكيف.

Table des matières

Table des figures	VIII
Liste des tableaux	X
Introduction Générale	1
Contexte de travail	1
Problématique	1
Objectifs du travail	2
Organisation du mémoire	2
1 Etat de l'art	3
1.1 Introduction	3
1.2 Les lignes de produits logiciel	3
1.2.1 Définitions	3
1.2.2 Principes de l'approche d'ingénierie de ligne de produits logiciels	4
1.2.2.1 Ingénierie de domaine	5
1.2.2.2 L'ingénierie d'application	6
1.2.3 La variabilité	7
1.2.3.1 La modélisation de la variabilité :	8
1.2.4 Diagrammes de caractéristiques	9
1.3 Systèmes auto-adaptatifs	9
1.3.1 Définitions	9
1.3.2 Intérêts du système auto-adaptatif.....	10
1.3.3 Propriétés du système auto-adaptatif.....	10
1.3.3.1 Auto-configuration.....	10
1.3.3.2 Auto-optimisation.....	11
1.3.3.3 Auto-guérison.....	11
1.3.3.4 Auto-protection.....	11
1.4 Les Lignes de Produits Logiciels Dynamiques.....	11

1.4.1	DSPL et SPL.....	12
1.5	Analyse des travaux existants	12
1.5.1	Comparaison et discussion	14
1.6	Conclusion.....	14
2	Ingénierie du domaine	15
2.1	Introduction.....	15
2.2	Motivation	15
2.3	Processus de développement du DSPL.....	16
2.4	Analyse de domaine	17
2.4.1	Diagramme de cas d'utilisations	17
2.5	Modélisation de la variabilité.....	20
2.5.1	Feature model	21
2.5.1.1	Feature model « Médecin »	21
2.5.1.2	Feature model « Patient »	22
2.5.1.3	Feature model « Technique ».....	23
2.6	Mapping Feature-Ontologie	24
2.6.1	Les concepts de l'ontologie.....	24
2.6.2	Les relations de l'ontologie.....	25
2.6.3	Représentation des fonctionnalités dans l'ontologie	25
2.6.4	Les règles de passage.....	26
2.7	Conception architecturale	27
2.7.1	Raffinement des composants d'e-health.....	28
2.7.1.1	Raffinement du Composant Gestion Patient	28
2.7.1.2	Raffinement du Composant Gestion dossier médical	29
2.8	Diagramme de classe.....	30
2.9	Conclusion	30
3	Ingénierie d'application	31
3.1	Introduction.....	31
3.2	Environnement logistique.....	31
3.2.1	Environnement de développement	31
3.2.1.1	Technologies de développement.....	31
3.2.1.2	Outils de programmation.....	32
3.3	Dérivation d'une application.....	33
3.3.1	Modélisation d'application	33
3.3.1.1	Feature Model d'application Espace Médecin	33
3.3.1.2	Feature Model d'application Espace Patient	33
3.3.1.3	Feature Model d'application Technique	34
3.3.2	Réalisation de l'application.....	34

3.3.3	Les interfaces	34
3.3.3.1	Page d'accueil.....	34
3.3.3.2	Espace Médecin.....	35
3.3.3.3	Espace Patient.....	38
3.4	Conclusion.....	42
Conclusion et perspectives		43
Bibliographie		44

Table des figures

1.1	Processus de développement d'une ligne de produits logiciels [21]	5
1.2	Exemple d'une ligne de produit [16]	7
1.3	Techniques de modélisation de la variabilité [12][14]	8
1.4	Notation des features [23]	9
1.5	Propriétés d'un système auto-adaptatif [5].....	11
2.1	Processus de développement DSPL	17
2.2	Diagramme de cas d'utilisation médecin	18
2.3	Diagramme de cas d'utilisation patient	19
2.4	Notation feature model.....	21
2.5	Feature model médecin.....	21
2.6	Feature model patient	22
2.7	Feature model technique.....	23
2.8	Les classes de l'ontologie proposée	25
2.9	Les propriétés de l'ontologie proposée.....	25
2.10	La hiérarchie des fonctionnalités	26
2.11	Diagramme des composants global.....	28
2.12	Composant Gestion Patient.....	29
2.13	Composant Gestion Dossier Médical	29
2.14	Diagramme de classe.....	30
3.1	Feature model application Espace médecin	33
3.2	Feature model application Espace patient.....	33
3.3	Feature model application technique	34
3.4	Page d'accueil.....	35
3.5	Page d'inscription.....	35
3.6	Page de connexion	36

3.7	Page d'ajout d'ordonnance	36
3.8	Page d'ajout d'analyses.....	37
3.9	Page d'ajout d'une lettre d'orientation.....	37
3.10	Profil médecin.....	38
3.11	Liste médecin.....	38
3.12	Page prise de rendez-vous	39
3.13	Liste rendez-vous	39
3.14	Chat.....	39
3.15	Appel vidéo.....	40
3.16	Profil patient	40
3.17	Liste des ordonnances d'un patient	41
3.18	Liste des analyses d'un patient	41
3.19	Liste des lettres d'orientation d'un patient	41
3.20	Diagnostic	42

Liste des tableaux

1.1	Comparaison entre SPL et DSPL.....	12
2.1	Cas d'utilisations pour chaque acteur.....	18
2.2	Description des features	20
2.3	Exécution des features.....	24
2.4	Passage du Feature model à l'ontologie.....	26

Liste des acronymes et abréviations

CSS *Cascading Style Sheets*

DSPL *Dynamic Software Product Line*

e-health *Electronic health*

FODA *Feature-Oriented Domain Analysis*

HTML *HyperText Markup Language*

LDP *Lignes de produits logiciels*

PLA *Product Line Architecture*

SPL *Software Product Line*

SQL *Structured Query Language*

Introduction Générale

Contexte de travail

Les systèmes modernes doivent être capables de s'adapter aux changements des besoins des utilisateurs et aux changements affectant le système lui-même ou son environnement. Parmi les exemples de systèmes exigeant des capacités d'auto-adaptation figurent les plateformes de télémédecines, qui devraient faire face aux changements environnementaux et les systèmes axés sur le service, qui devraient remplacer les services peu fiables à la volée.

Dans ce contexte, la ligne de produits logiciels dynamiques est une approche d'ingénierie pour développer des systèmes auto-adaptatifs basés sur des points communs et des variabilités pour une famille de systèmes similaires.

Problématique

Une étude approfondie sur les techniques de développement de systèmes autonomes et auto-adaptatifs et leur application dans l'amélioration des capacités dynamiques de produit nous a permis de retirer quelques conclusions relatives aux limites de ce domaine, celles-ci peuvent se résumer dans les points suivants :

- Les solutions DSPL (Dynamic Software Product Line) ne répondent pas aux exigences d'adaptabilité du système,

- Les approches suggérées sont développées de manière ad hoc. Ceci nous amène à nous poser certaines questions cruciales qui sont les suivantes :

QR1. Comment gérer et modéliser la variabilité dans les DSPL ?

QR2. Comment représenter les variabilités statique et dynamique ensemble de manière appropriée dans la construction des DSPL ?

QR3. Quels styles architecturaux conviennent au développement de DSPL ?

QR4. Quelles sont les technologies telles que les modèles de composants, les frameworks, les langages et les outils utilisés pour la conception et le développement de DSPL ?

Objectifs du travail

L'objectif de notre travail consiste à concevoir et développer une ligne de produit dynamique dédié aux plateformes médicales auto-adaptative qui permet de :

- Progresser l'état de l'art en matière de conception et de mise en œuvre de lignes de produits logiciels dynamiques
- Identifier les similarités et variabilités entre les applications de HealthCare.
- Identifier les variabilités statique et dynamique
- Représentation de l'ensemble de caractéristique de DSPL grâce à l'ontologie ainsi que l'ensemble des instances individuelles des concepts constituent une base de connaissances.
- Développer les core asset du domaine (élément réutilisable) :
 - Une ontologie de domaine
 - Les diagrammes de modélisation du domaine.
 - Les composants réutilisables
- Développer des plateformes HealthCare en basant sur les core assets.
- 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 trois chapitres

■ Chapitre1 : Etat de l'art

Est un chapitre de généralités, il représente une vue globale sur les lignes de produits logiciels, les systèmes auto-adaptatifs, les lignes de produits logiciels dynamiques ainsi que la différence entre SPL et DSPL et les travaux connexes.

■ Chapitre2 : Ingénierie du domaine

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

■ Chapitre3 : Ingénierie d'application

Ce dernier chapitre, comporte quant à lui la présentation de l'environnement dont lequel notre application a été réalisé, mettre en évidence les langages et outils utilisés et exposer les interfaces principales de la plateforme.

Chapitre 1

Etat de l'art

1.1 Introduction

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.

Au cours de la dernière décennie, l'importance de l'auto-adaptabilité des systèmes logiciels a été de plus en plus reconnue dans ses divers domaines d'applications et technologies. En effet, différentes interprétations et concepts ont été produits par la communauté des chercheurs, le sujet faisant toujours l'objet d'intenses recherches et développements.

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 [6].

1.2 Les lignes de produits logiciel

1.2.1 Définitions

Plusieurs définitions ont été proposées dans la littérature pour décrire les lignes de produits logiciels, la définition suivante a été proposée par Cléments et Northrop [20] qui donne une idée plus précise sur les SPL :

« un ensemble de systèmes à forte composante logicielle partageant un ensemble commun de caractéristiques gérées qui répondent aux besoins spécifiques d'un segment de marché ou d'une mission particulière et qui sont développés à partir d'un ensemble commun d'actifs de base d'une manière prescrite. »

Cette définition caractérise les produits, les membres de la SPL par un ensemble de propriétés communes (similarité) et aussi par leurs différences (variabilité). Rappelons que la variabilité est le concept clé dans la SPL. Elle permet le développement des produits configurés par la réutilisation des artefacts prédéfinis et ajustables. Elle est modélisée au niveau de la phase d'ingénierie de domaine pour représenter les différences entre les produits planifiés.

Jan Bosch propose dans [11], la définition suivante :

« La ligne de produits logiciels se compose de l'architecture de la ligne de produits et d'un ensemble de composants réutilisables conçus pour être incorporés dans l'architecture de la ligne de produits. En outre, le produit consiste en des produits logiciels développés à l'aide des actifs réutilisables mentionnés. »

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

Une dernière définition proposée par Lai et Weiss :

« Une ligne de produits logiciels est une famille de produits conçus pour tirer parti de leurs aspects communs et des variations prévues. »

1.2.2 Principes de l'approche d'ingénierie de ligne de produits logiciels

L'ingénierie des lignes de produits logiciels se divise en deux phases complémentaires [21] : l'ingénierie de domaine et l'ingénierie d'application tel que montré sur la figure 1.1.

L'ingénierie de domaine adopte la stratégie de développement pour la réutilisation, tandis que l'ingénierie d'application s'appuie sur la stratégie du développement avec réutilisation. La logique derrière ces deux phases combine l'idée de compenser le temps de développement des artefacts communs, pendant la première phase, par la réutilisation de ces artefacts pendant la deuxième phase. Ces deux activités seront présenter on détaille dans la section suivent."

L'objectif des SPL est de promouvoir une réutilisation logicielle effective et systématique. Pour ce faire, la mise en œuvre d'un PL est divisée en deux processus de développement parallèles. La 1ère, appelée ingénierie du domaine ou "Domain Engineering" a pour but le développement des artefacts réutilisables. C'est un processus de développement pour la réutilisation en ce sens qu'il prépare les éléments qui seront plus tard réutilisés. Le 2ème processus, appelé ingénierie des applications ou "Application Engineering" a pour but de construire de nouvelles applications spécifiques. C'est un processus de développement pour la réutilisation puisqu'il se base sur les artefacts logiciels réutilisables développés lors de l'ingénierie du domaine [21].

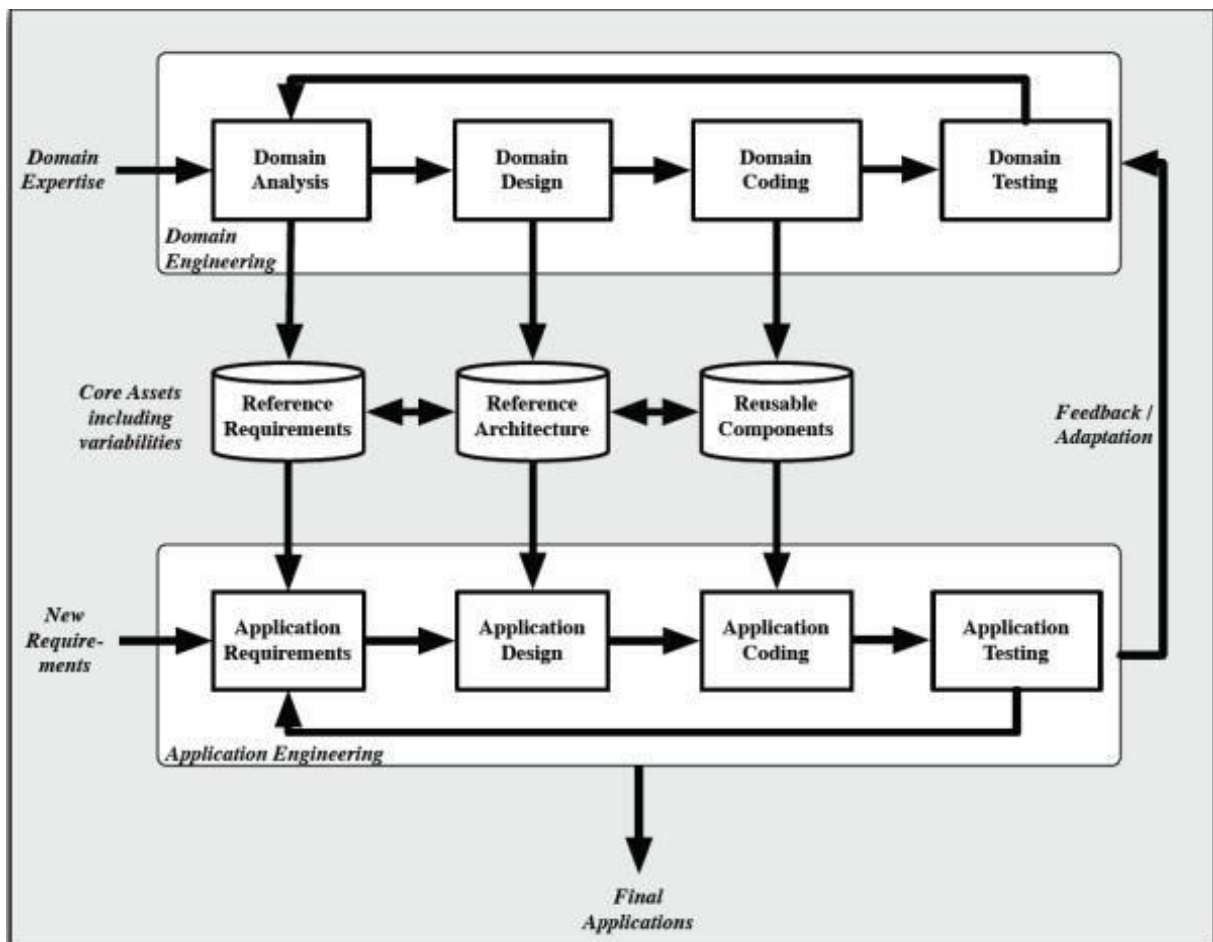


FIGURE 1.1 – Processus de développement d'une ligne de produits logiciels [21]

1.2.1.1 Ingénierie de domaine

L'ingénierie du domaine consiste à développer et construire les assets qui seront réutilisés pour la construction de produits. Notons qu'un asset est un élément qui permet de développer un logiciel, par exemple un document de spécification, modèle, code, etc. Ainsi, il s'agit donc d'un développement pour la réutilisation. Son but est d'étudier le domaine de la SPL et d'identifier les similarités et les variabilités entre les produits.

L'ingénierie de domaine est composée de quatre activités qui sont l'analyse de domaine, la modélisation de domaine, l'implémentation de domaine et le test de domaine.

- Analyse de domaine : inclut l'ingénierie des exigences liées à ce domaine. En d'autres termes, elle consiste à analyser les exigences afin de distinguer celles qui s'appliquent à tous les produits du domaine de celles qui sont spécifiques à certains produits en particulier.
- - Modélisation de domaine : L'objectif de cette activité est de définir l'architecture l'architecture de la ligne de produits logiciels. Cette architecture donne une vue structurelle sur les différents membres de la ligne de produits logiciels. Cette activité il est essentiel d'intégrer les mécanismes de variabilité utilisés et d'identifier les composants réutilisables de l'architecture.
- Réalisation de domaine : les différentes parties réutilisables de l'architecture sont détaillées et implémentées dans des composants logiciels réutilisables.
- Test de domaine : est chargée de la validation et de la vérification des composants réutilisables obtenus à partir de l'activité précédente. Les tests effectués sur les composants se font par rapport à leur conformité aux spécifications données dans les exigences, l'architecture et les artefacts de modèle.

1.2.1.2 L'ingénierie d'application

Cette phase vise à obtenir des produits concrets à partir des éléments identifiés lors de la phase d'ingénierie de domaine. La stratégie adoptée repose sur le développement avec réutilisation, avec pour objectif de maximiser le taux de réutilisation des éléments définis et développés pour des produits spécifiques de la ligne de produits. Les similitudes et les différences entre les produits sont exploitées pour leur construction, appelée dérivation de produits. Cette phase comprend quatre activités : l'ingénierie des exigences d'application, la modélisation d'application, l'implémentation d'application et le test d'application.

- Ingénierie des exigences de l'application : elle s'intéresse à définir les exigences spécifiques des produits souhaités. Ces exigences ont un impact direct sur les éléments du domaine qui seront réutilisés lors de la construction du produit..
- Modélisation de l'application : Cette étape consiste à obtenir l'architecture du produit en se basant sur l'architecture globale définie lors de la modélisation du domaine. Pour ce faire, les éléments nécessaires du modèle sont sélectionnés et intégrés.
- Réalisation de l'application : cette activité consiste à la réalisation du produit concret souhaité. Les composants correspondant aux parties sélectionnées lors de l'étape précédente sont assemblés pour former le produit final.
- Test de l'application : cette activité a pour objectif de valider et de vérifier que le produit obtenu à partir de l'activité précédente est conforme aux exigences, à l'architecture et aux parties sélectionnées du modèle.

1.2.2 La variabilité

Dans le domaine des lignes de produits logiciels, la variabilité est une notion essentielle qui permet le développement de produits configurés en réutilisant des éléments prédéfinis et ajustables. Elle est modélisée lors de la phase d'ingénierie de domaine pour représenter les différences entre les produits planifiés. Il existe plusieurs définitions de la variabilité. Par exemple, Weiss et Lai la définissent comme : « *une hypothèse sur la façon dont les membres d'une famille peuvent se différencier entre eux* ».

Une autre définition dans [19] : « *regroupement des caractéristiques qui différencient les produits de la même famille* ». Ce concept est lié à celui de «point de variation» qui permet d'identifier la partie du système ou une variation va être introduite dans la famille de produits, donnant naissance à un nouveau membre ou produit dans une ligne.

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

Pour mieux mettre en évidence la notion des SPL, les points de variation et les variantes, un exemple de ces concepts est donné à la figure 1.2.

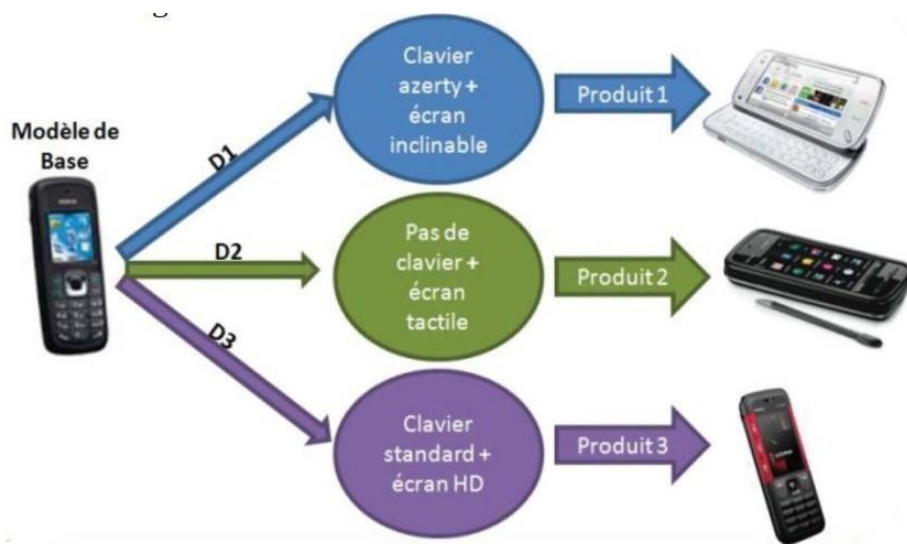


FIGURE 1.2 – exemple d'une ligne de produit [16]

Cette SPL contient un nombre de composants et de fonctionnalités de base communes à tous les téléphones de la famille qui sont représentées par le téléphone simple qui se trouve à gauche de la figure. Ces fonctionnalités de base peuvent être, par exemple, l'envoi de SMS, le module d'appels, la gestion des contacts, etc. Ensuite, il y a trois décisions de conception ou configurations différentes qui mélangent les options disponibles en termes de type d'écran et type de clavier.

Finalement, dans la partie droite de la figure, nous pouvons observer trois produits dérivés à partir de notre PL. Dans cet exemple nous pouvons identifier deux points de variation, le clavier et l'écran, avec ses variantes azerty, pas de clavier, standard et inclinable, tactile, HD.

1.2.2.1 La modélisation de la variabilité :

La modélisation de la variabilité est importante pour gérer facilement la variabilité dans les familles de produits logiciels, en particulier lors de la dérivation du produit. Au cours des dernières années, plusieurs techniques de modélisation de la variabilité ont été développées, chacune utilisant ses propres concepts pour modéliser la variabilité apportée par une famille de produits. La figure 1.3 montre ces différentes techniques.

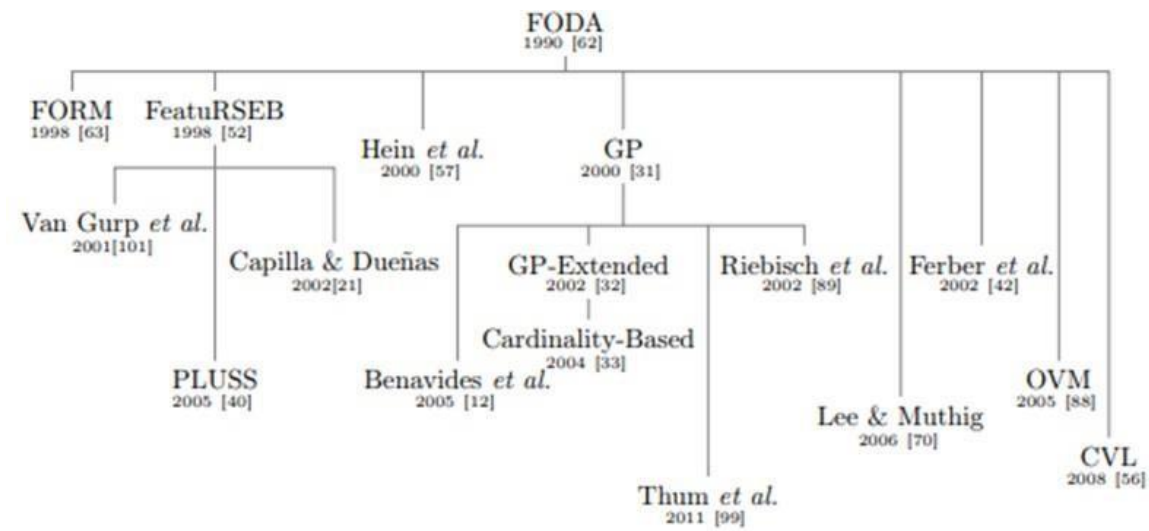


FIGURE 1.3 – Techniques de modélisation de la variabilité [12][14]

Dans le but de décrire les différents types de variabilité présents dans les lignes de produits, une classification a été proposée dans l'article référencé [22] :

- **Similitude (Commonality)** : Il s'agit d'une caractéristique commune à tous les produits de la famille. Cette caractéristique est intégrée en tant que partie de la plateforme ou du noyau commun de la famille de produits.
- **Variabilité (Variability)** : Il s'agit d'une caractéristique qui peut être présente chez certains produits, mais pas chez tous les membres de la famille de produits. Cette caractéristique doit être modélisée en tant que point de variation et implémentée de manière à ce qu'elle soit présente uniquement dans les produits sélectionnés.
- **Produit-Spécifique (Product-specific)** : Il s'agit d'une caractéristique qui est incluse uniquement dans un produit spécifique de la famille. Elle est généralement une réponse à une demande particulière du client pour ce produit spécifique. Cette caractéristique ne sera pas intégrée dans la plateforme de la ligne de produits, mais elle doit être compatible avec celle-ci.

1.2.3 Diagrammes de caractéristiques

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

Parmi les techniques les plus utilisées on trouve les Features models, Cette technique concerne l'emploi de diagrammes de caractéristiques, qui permettent à l'utilisateur de représenter de manière hiérarchique la structure d'une ligne de produits, ses différentes options de variation et ses variantes. De plus, l'objectif des diagrammes de caractéristiques est de décrire les relations existantes entre les différentes fonctionnalités ou caractéristiques du système. Cette approche a été largement utilisée et plusieurs travaux ont été proposés pour étendre la notation de base présentée par Kang et al. Dans [13].

Par la suite, un exemple de l'utilisation de cette notation graphique est présenté dans La figure 1.4 qui montre les différentes notations des «features», pour illustrer la façon dont les lignes de produits sont représentées en utilisant FODA proposé par Kang et al. Dans [13].

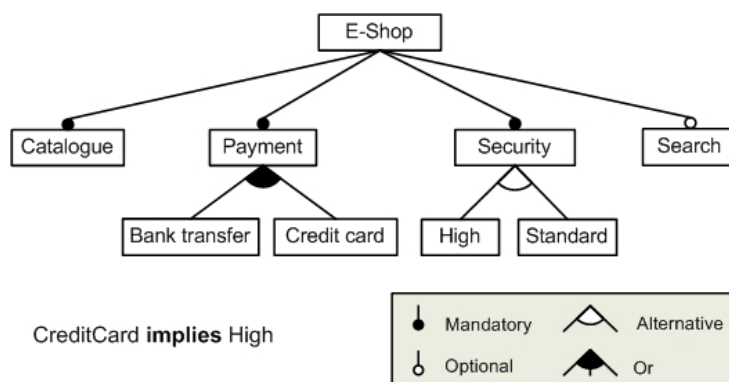


FIGURE 1.4 – notation des features [23]

1.3 Systèmes auto-adaptatifs

1.3.1 Définitions

Définition1 : Un système auto-adaptatif évalue son propre comportement et modifie sa propre performance lorsque l'évaluation indique que n'est pas accompli ce que le logiciel est destiné à faire ou lorsque de meilleures fonctionnalités ou performance sont possibles [17].

Définition2 : Un système auto-adaptatif est un système en boucle fermée qui peut se modifier lui-même dû aux changements continus du système, ses exigences et les tendances existantes de développement et de déploiement des systèmes complexes, réduisant les interactions humaines. La conception de système auto-adaptatif dépend des besoins de l'utilisateur et des propriétés et caractéristiques environnementales du système. Les logiciels auto-adaptatifs nécessitent une grande fiabilité, robustesse, adaptabilité et disponibilité [3].

Définition3 : Un système auto-adaptatif modifie son propre comportement en réponse aux changements dans son environnement d'exploitation. Par environnement d'exploitation, nous entendons tout ce qui peut être observé par le système, tel que l'entrée de l'utilisateur final, les dispositifs matériels externes et les capteurs ou l'instrumentation du programme [18].

1.3.2 Intérêts du système auto-adaptatif

Les systèmes d'auto-adaptation permettent de suivre les changements fréquents qui se produisent dans leur environnement afin d'assurer une qualité de service optimal pour l'utilisateur final. Ces systèmes sont gérés de manière autonome. L'objectif inhérent aux systèmes adaptatifs est de maintenir la qualité du service rendu quel que soit l'état de l'environnement. Il s'agit de stabiliser la qualité de service [7].

Trois capacités essentielles caractérisent un système adaptatif : l'observation, la décision et l'intro-action [7].

a/ Observation : Un système adaptatif est un système qui observe son environnement et en détecte les changements. D'un point de vue technique, cette capacité se traduit par l'existence de sondes (logicielles ou matérielles) permettant de mesurer les propriétés pertinentes de l'environnement [7].

b/ Décision : En fonction de l'état de l'environnement, mais également fonction de sa configuration actuelle (capacité d'introspection), un système adaptatif décide par lui-même de la nouvelle configuration à adopter pour fonctionner de manière optimale [7].

c/ Intro-action : pour modifier sa propre configuration, le système adaptatif manipule et modifie les éléments qui le constituent. Ces intro-actions sont des actions de l'intérieur sur l'intérieur. Il s'agit d'une capacité d'introspection active [7].

1.3.3 Propriétés du système auto-adaptatif

Un système auto-adaptatif s'adapte au moment de l'exécution aux changements de lui-même et de l'environnement. Pour y parvenir, idéalement parlant, les systèmes devraient avoir certaines caractéristiques adaptatives connues sous le nom de propriétés auto-adaptatives. Ces propriétés ont été introduites dans le calcul autonome [5] [15] et ont été ci-après mentionnées dans le contexte de l'auto-adaptation. Ces propriétés sont composées de quatre catégories, qui sont discutées en détail ci-après. La figure 1.5 montre ces différentes catégories.

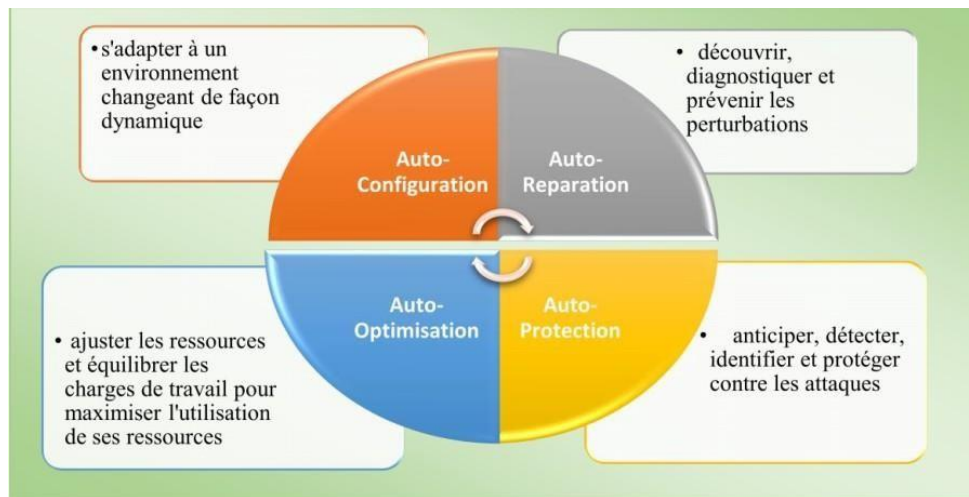


FIGURE 1.5 – Propriétés d'un système auto-adaptatif [5]

1.3.3.1 Auto-configuration :

Est la capacité de configurer automatiquement et dynamiquement en réponse aux changements. Cela peut inclure l'installation, l'intégration, la suppression et composition/décomposition des éléments du système.

1.3.3.2 Auto-optimisation :

Est la capacité de gérer les performances et l'allocation des ressources tout en satisfaisant les besoins des utilisateurs. Cela inclut des préoccupations telles que débit, temps de réponse, etc.

1.3.3.3 Auto-guérison :

Est la capacité de découvrir, de diagnostiquer et de réagir à perturbations. Cela inclut la guérison réactive ou proactive. En proactif, les problèmes potentiels de guérison sont anticipés et pris en compte dès le début pour l'échec, tandis que l'auto-réparation se concentre la récupération d'eux.

1.3.3.4 Auto-protection :

Est la capacité de détecter les failles de sécurité et de récupérer de leurs effets. Cela comprend à la fois une protection réactive et proactive, à savoir la récupération des attaques existantes et anticipées.

1.4 Les Lignes de Produits Logiciels Dynamiques

Au cours des 15 dernières années, les approches SPL traditionnelles [11] [20] [16] ont abordé avec succès le développement de familles de systèmes à partir d'une architecture commune, maximisant la réutilisation et exploitant la variabilité des systèmes pour produire des produits moins chers et de meilleure qualité en moins temps. Cependant, dans un monde en mutation, le besoin croissant de logiciels adaptatifs exige un comportement autonome et des propriétés d'autogestion apportant de nouveaux défis d'adaptation dynamique dans les familles de système. Alors que la variabilité devient plus dynamique, il y a une nette tendance à passer à des approches de développement récentes comme les lignes de produits logiciels dynamiques DSPL [9] [10] comme paradigme émergent pour gérer la variabilité à l'exécution et à tout moment.

1.4.1 DSPL et SPL

DSPL est une approche d'ingénierie pour développer les systèmes auto-adaptatifs basé sur SPL. SPL traite de la modélisation des points communs et des variabilités au sein d'une famille de systèmes similaires, la modélisation se fait par les modèles de caractéristiques qui peuvent être classés comme : obligatoires, facultatives ou alternatives. DSPL permet au SPL d'être reconfigurable au moment d'exécution.

TABLE 1.1 – Comparaison entre SPL et DSPL

	SPL	DSPL
variabilité	Statique	dynamique, tardive, d'exécution
Temps de liaison	Produit au moment de conception pour générer un produit	Peut se produit au moment de conception ou au moment de l'exécution pour adapter le produit permettant une variabilité dynamique.
Architecture	Fournit un cadre commun pour un ensemble d'architectures de produits individuelles.	Architecture à système unique qui fournit une base pour toutes les adaptations possibles du système.
Cycle de vie	Deux phases : <ul style="list-style-type: none"> • l'ingénierie de domaine • l'ingénierie d'application. 	Deux cycles de vie : 1 ^{er} vise à un développement du système adaptatif et le 2 ^{ème} exploite l'adaptabilité d'utilisation.

1.5 Analyse des travaux existants

Dans ce qui va suivre, nous allons analyser quelques articles de la littérature qui traitent sur le domaine de l'adaptation dans les lignes de produits logiciels.

➤ **Liwei Shen et al., 2012** [19]

Dans cette étude, afin de donner une solution aux problèmes pratiques, Liwei Shen et al., proposent un méta-modèle d'exigences de domaine dans SPLASA. Il est décrit avec différents points de vue et la variabilité des contraintes de liaison à l'intérieur est soulignée. Sur cette base, un guide est conclu pour soutenir la personnalisation cohérente vers le modèle de domaine. De plus, une étude expérimentale sur une ligne de produits d'entreprise basée sur le Web impliquant une capacité d'auto-adaptation est menée pour évaluer le modèle.

➤ **Liliana Pasquale et al., 2012** [17]

Cette étude traite de la convergence entre l'architecture orientée services (SOA) et les lignes de produits logiciels dynamiques (DSPL). Leur contribution est donc double : d'une part, ils présentent une solution technique SOA pour DSPL; d'autre part, ils fournissent aux processus BPEL le support de modélisation dont ils ont besoin pour comprendre et adopter des degrés de variabilité plus élevés. L'approche définit les variabilités pour leur processus BPEL en utilisant CVL (Common Variability Language).

La proposition a été validée à travers un scénario simple dans le contexte des maisons intelligentes dans lesquelles on automatise le contrôle de systèmes domestiques (par exemple, chauffage, éclairage, détection de présence) pour économiser l'énergie.

➤ **Tom Dinkelaker et al., 2010** [25]

Dans cette étude, Tom Dinkelaker et al. proposent une nouvelle approche pour les DSPL qui utilise un modèle de caractéristiques dynamiques pour décrire la variabilité dans les DSPL et qui utilise un langage spécifique au domaine pour implémenter de manière déclarative les variations et leurs contraintes. L'approche combine plusieurs tendances dans la programmation orientée aspect pour les DSPL, à savoir les aspects dynamiques, les modèles d'exécution des aspects, ainsi que la détection et la résolution des interactions d'aspect. L'avantage est que les reconfigurations ne doivent pas être spécifiées pour chaque combinaison de fonctionnalités, mais uniquement pour les fonctionnalités en interaction. Ils ont validé l'approche dans un exemple de ligne de produits logiciels dynamiques de l'industrie et évalué l'approche de manière préliminaire.

1.5.1 Comparaison et discussion

Afin d'avoir une vue globale sur les travaux réalisés précédemment, quelques critères ont été sélectionnés pour comparer ces derniers dont : Modélisation de la variabilité, Compatibilité Extensibilité, Domaine d'application et Tools support.

❖ **Modélisation de la variabilité :**

Chaque approche aborde la gestion de la variabilité dans les DSPL de manière différente. Liwei Shen et al. se concentrent sur l'introduction d'un modèle de rôle pour améliorer la traçabilité des variations, tandis que Liliana Pasquale et al. proposent une solution SOA pour gérer des degrés de variabilité plus élevés. Tom Dinkelaker et al., quant à eux, utilisent un modèle de caractéristiques dynamiques combiné à des aspects pour gérer les interactions entre les fonctionnalités.

❖ **Compatibilité :**

Pour la compatibilité, Liwei Shen et al., introduisent le méta-modèle qui inclut le mécanisme pour modéliser formellement la variabilité de la logique d'adaptation. Liliana Pasquale et al., utilisent le diagramme de composant(UML).Tom Dinkelaker et al., modélisent la VP tradifs sous forme d'annotations sur des éléments de modélisation. Par exemple, dans un diagramme d'activités UML, une activité est annotée comme étant dynamiquement variable.

❖ **Extensibilité :** les trois travaux sont extensibles.

❖ **Domaine d'application**

Liwei Shen et al. se concentrent sur Les systèmes de gestion des commandes des maisons d'édition (PHOM), tandis que Liliana Pasquale et al. Génèrent un SPL industriel de télévision en temps réel, Tom Dinkelaker et al., quant à eux, proposent un scénario de vente.

❖ **Tool support :**

Liwei Shen et al. se concentrent sur un outil qui fournit une modélisation de domaine ainsi qu'une personnalisation cohérente, tandis que Liliana Pasquale et al. Concentrent sur des solveurs CSP : pour vérifier s'il est possible de configurer le produit avec les fonctionnalités demandées, les solveurs Logic Truth Maintenance Systems (LTMS) et SAT peuvent être utilisés pour propager la désactivation/activation des fonctionnalités et l'outil FAMA, Tom Dinkelaker et al., quant à eux, proposent Popart qui est une technologie de langage sous-jacente pour l'implémentation du DFL (Dynamic Feature Language).

1.6 Les ontologies

1.6.1 Définitions

Définition1: An ontology is a specification of a conceptualization. That is, an ontology is a

description of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as setof-concept-definitions, but more general [29].

Définition2: The subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, called an ontology, is a catalogue of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D. The types in the ontology represent the predicates, word senses, or concept and relation types of the language L when used to discuss topics in the domain D [30].

1.7 Conclusion

Dans ce premier chapitre, nous avons présenté l'état de l'art commençant par les lignes de produits logiciels, les systèmes auto-adaptatifs et les Lignes de produits logiciels dynamiques. Ainsi, nous avons fait une comparaison entre SPL et DSPL, ce qui nous aide par la suite dans notre conception qui sera l'objet du chapitre suivant.

Chapitre 2

Ingénierie du domaine

2.1 Introduction

Après avoir vu les principaux domaines de notre système en commençant par les lignes de produits, Les systèmes auto-adaptatifs, et les SPL dynamiques, maintenant il est temps de passer à la conception de notre DSPL. En d'autres termes, il s'agit de déterminer comment utiliser ces derniers afin d'atteindre notre objectif. Dans ce chapitre, nous exposerons à la fois les raisons qui nous ont poussé à entreprendre ce travail et notre proposition de solution. Nous présenterons également une vue d'ensemble du processus de conception, illustrée par un schéma détaillant les différentes étapes impliquées. Tout au long de ce chapitre, nous allons présenter notre système, et détailler la solution proposée.

2.2 Motivation

En Algérie, l'introduction de la télémédecine est encore à ses débuts. Le réseau de « santé-Algérie » opérationnel depuis Janvier 1999, peut permettre grâce à sa plate-forme réseau dont il dispose actuellement, le développement à court terme d'un certain nombre d'actions en matière d'intégration des nouvelles technologies de l'information et de la communication dans notre secteur. Quelques grands hôpitaux d'Algérie ont réussi faire du jumelage soit entre eux ou avec les hôpitaux européens, c'est-à-dire que l'application de la télémédecine fait premier pas de réalisation.

Le premier pas du lancement réel d'un projet pilote de télémédecine est daté en 2008. L'expérience menée entre les hôpitaux de Birtraria à Alger et d'Ouargla dans le sud du pays a été concluante et devrait être élargie aux autres centres de soins et de santé des autres régions.

Du fait de l'étendue de la superficie du territoire algérien, la télémédecine permet ainsi de raccourcir les distances et d'éviter le transfert des malades du sud du pays notamment vers le nord du pays. C'est la raison pour laquelle une expérience a été menée avec un hôpital

d'Ouargla spécialisé dans la pédiatrie. En ce sens, un projet pilote de connexion d'une plateforme de télémédecine entre l'hôpital de Birtraria et celui d'Ouargla a été présenté, à l'occasion d'un workshop international sur la télémédecine (WITU-2008), organisé à Alger par le centre de développement des technologies avancées (CDTA). Grâce à cette nouvelle technique, des diagnostics sont établis à partir d'Alger, suite à des séances de vidéo conférences organisées entre les deux hôpitaux.

La pratique de la télémédecine a été largement appliquée aux autres centres hospitalo-universitaires, avec une convention signée de 3 ans; CHU de Sétif en 2014, CHU de Batna en 2015, CHU de Tlemcen en 2015, CHU de Tizi Ouzou en 2016, dont l'objet de répondre aux besoins illimités d'une population de plus en plus accumulé.[8]

2.3 Processus de développement du DSPL

Notre objectif est de concevoir et développer une ligne de produits dynamique guidée par les ontologies.

Le processus consiste à gérer une DSPL dans son ensemble plutôt que de considérer chaque produit comme une entité distincte. Ainsi, la DSPL doit prendre en compte les besoins de toutes les catégories d'utilisateurs ciblées. La définition des membres de la famille de produits planifiée permet d'identifier et de planifier la mise en œuvre des éléments communs réutilisables ainsi que les différences entre eux. Cette phase adopte donc une stratégie de développement axée sur la réutilisation, car les artefacts logiciels communs tels que les exigences, les composants et les classes sont conçus de manière à pouvoir être réutilisés dans les produits prévus.

Les différentes parties composant notre système sont les suivantes :

- **Analyse de domaine**
- **Modélisation de la variabilité**
- **Mapping Feature-Ontologie**
- **Architecture du Système**
- **Implémentation**

Le détail de chaque phase sera représenté dans les sections suivantes, un schéma récapitulant ces différents points est illustré dans la figure suivante :

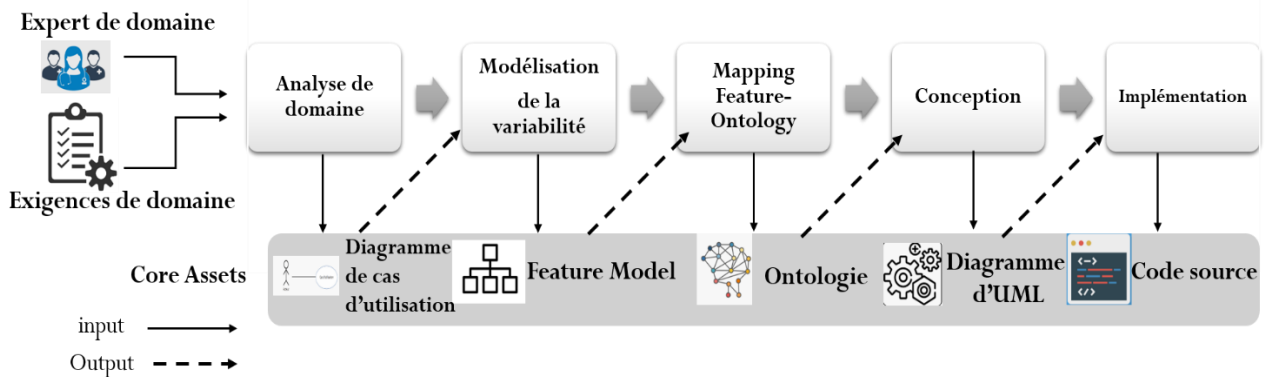


FIGURE 2.1 – Processus de développement DSPL

2.4 Analyse de domaine

La première étape du processus de développement consiste à réaliser une analyse de domaine afin de comprendre le domaine du problème et d'identifier les similitudes et les variations propres au domaine. Pour faciliter davantage cette compréhension, nous utiliserons un diagramme de cas d'utilisation. En utilisant un diagramme de cas d'utilisation, nous pouvons capturer et communiquer efficacement les fonctionnalités fondamentales et les relations présentes dans les plateformes des e_Health.

2.4.1 Diagramme de cas d'utilisations

Avant de modéliser le Feature model et construire l'ontologie, on doit identifier les acteurs et spécifier les exigences minimales de chaque produit dans notre ligne.

a) Identifications des acteurs :

-Médecin : C'est un visiteur ayant déjà créé un compte sur notre site en tant que médecin, il peut donc suivre le processus de la téléconsultation en toute sécurité sachant que notre système doit être l'unique responsable de la confidentialité des données personnelles et des dossiers médicaux de ses patients.

-Patient : C'est un visiteur ayant déjà créé un compte sur notre site en tant que patient, il peut donc suivre le processus de la téléconsultation en toute sécurité sachant que notre système doit être l'unique responsable de la confidentialité des données personnelles et de son dossier médicale.

b) Spécification des besoins :

Acteurs	Besoins
Médecin	-Modifier coordonnées -Gérer Patient -Gérer ordonnance -Gérer dossier médical -Gérer réunion
Patient	-Consulter profil -Gérer médecin -Gérer dossier médical -Gérer rendez-vous -Faire diagnostics -Gérer ordonnance

TABLE 2.1 – Cas d'utilisations pour chaque acteur

c) Diagramme de cas d'utilisation « Médecin » :

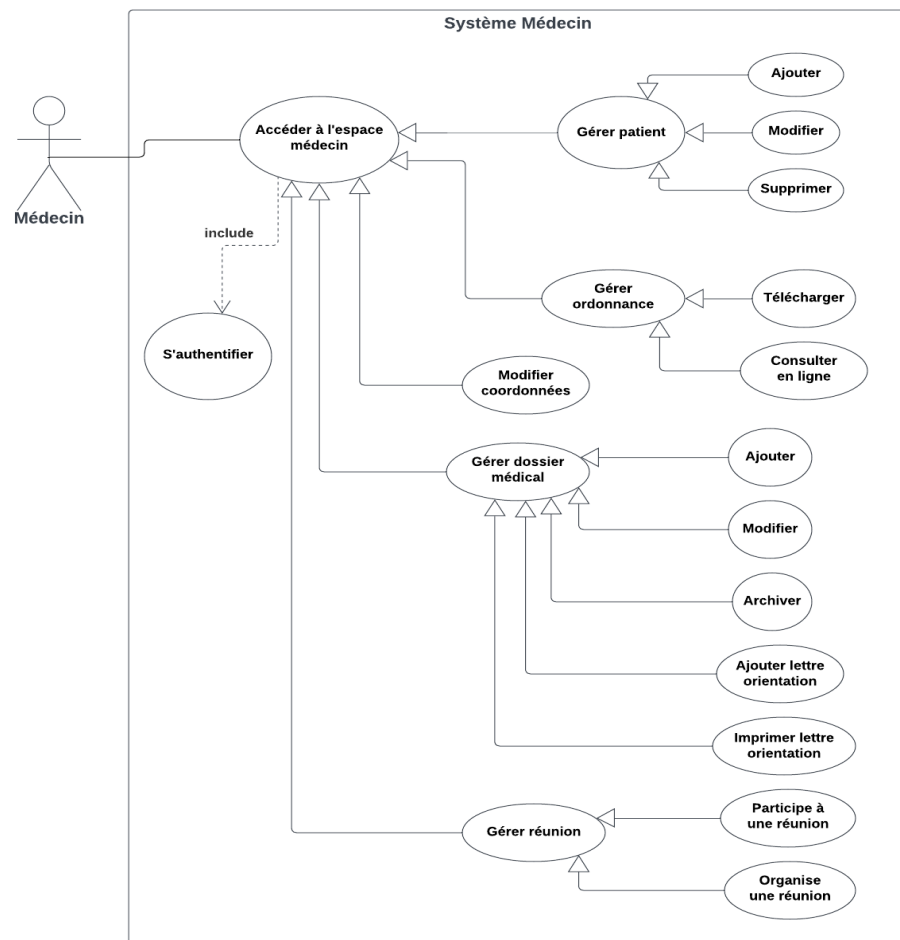


FIGURE 2.2 – Diagramme de cas d'utilisation médecin

d) Diagramme de cas d'utilisation « Patient » :

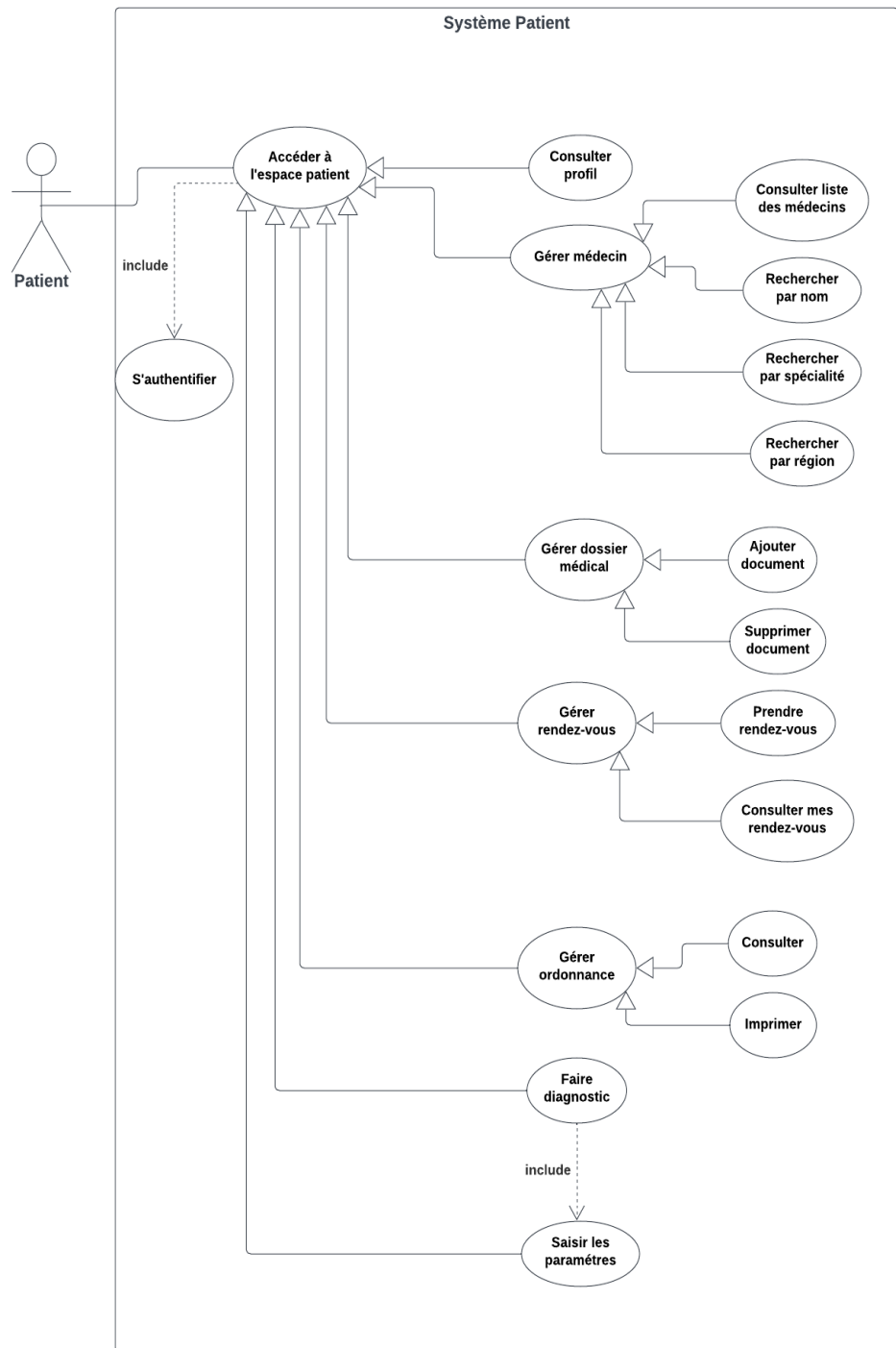


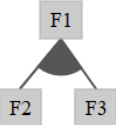
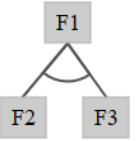




FIGURE 2.3 – Diagramme de cas d'utilisation patient

2.5 Modélisation de la variabilité

Au cours des dernières années, plusieurs techniques de modélisation de la variabilité ont été développées, dans notre travail nous avons utilisé la technique FODA (Feature-Oriented Domain Analysis). Les modèles FODA sont souvent exprimés sous forme de diagrammes constitués d'un ensemble de symboles graphiques formant un arbre de configuration. Czarnecki et Eisenecker [2] présentent une revue des symboles et des définitions formelles correspondantes pour la construction de modèles de configuration. Le « Feature Model » a été défini dans [23] par Ziadi comme une notation standard pour décrire la variabilité dans les lignes de produits. Dans notre travail on a ajouté de plus le feature dynamique par défaut et le feature adaptable.

TABLE 2.2 – Description des features

Symbole	Nom	Description
	Feature racine abstrait	Une caractéristique abstraite utilisée seulement pour l'organisation conceptuelle des features.
	Feature obligatoire	Si le nœud parent F1 est présent, alors le nœud enfant F2 aussi.
	Feature optionnel	Le nœud enfant F2 peut ou peut ne pas être présent alors que le nœud parent F1 l'est.
	Feature OR	Si le nœud parent F1 est présent, alors au moins un des nœuds enfants F2 ou F3 reliés par une arête l'est.
	Feature Alternative	Si le nœud parent F1 est présent, alors un seul des nœuds enfants F2 ou F3 reliés par une arête alternatif'est.
	Feature Dynamique par défaut	Un nœud optionnel sélectionné par Défaut
	Feature adaptable	Un nœud optionnel alternatif déclenché en cas d'adaptabilité
$F1 \Rightarrow F2$	Implication	Le nœud F2 est présent dès que le nœud F1 est présent.
$F1 \Rightarrow \neg F2$	Exclusion	les nœuds F1 et F2 ne peuvent pas être présents en même temps.

2.5.1 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 créée. Le deuxième est le feature model technique, qui contient Les fonctionnalités techniques.

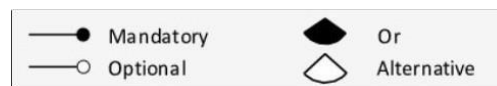


FIGURE 2.4 – Notation feature model

2.5.1.1 Feature model « Médecin »

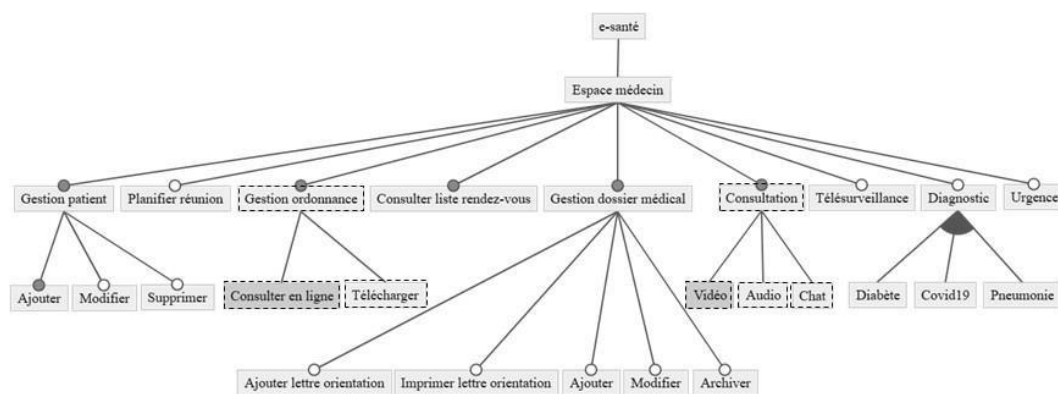


FIGURE 2.5 – Feature model médecin

Les contraintes :

- Télésurveillance **require** urgence
- Espace médecin **require** s'authentifier.
- Planifier réunion **require** consulter liste rendez-vous.

Espace médecin :

- La fonctionnalité obligatoire **Gestion patient** implique trois autres fonctionnalités : **Ajouter** (permet au médecin d'ajouter un nouveau patient), **Modifier** (permet au médecin de modifier quelques coordonnées du patient), **Supprimer** (permet au médecin de supprimer un patient).

- La fonctionnalité optionnelle **Planifier réunion** permet au médecin de se réunir avec d'autres médecins.

- La fonctionnalité obligatoire **Gestion ordonnance** implique deux autres fonctionnalités :

Consulter en ligne (Permet au médecin de consulter l'ordonnance donnée aux patients en ligne), **Télécharger** (permet au médecin de télécharger une ordonnance).

- La fonctionnalité obligatoire **Consultation** implique trois autres fonctionnalités : **Vidéo** (permet au médecin de faire une téléconsultation avec un patient via un appel vidéo), **Audio** (permet au médecin de faire une téléconsultation avec un patient par un audio), **Chat** (permet au médecin de faire une téléconsultation avec un patient par des messages).

- La fonctionnalité optionnelle **Diagnostic** permet de faire des diagnostics soit sur un **Diabète** ou sur **Covid19** ou bien sur **La pneumonie**.

- La fonctionnalité obligatoire **Gestion dossier médical** implique cinq autres fonctionnalités : **Ajouter lettre orientation** (permet au médecin d'ajouter une lettre d'orientation en cas d'urgence), **Imprimer lettre orientation** (permet au médecin d'imprimer la lettre d'orientation), **Ajouter** (permet au médecin d'ajouter un nouveau dossier médical), **Modifier** (permet au médecin de modifier le dossier médical), **Archiver** (permet au médecin d'archiver le dossier médical).

- La fonctionnalité optionnelle **Urgence** permet de faire appel aux urgences en cas d'urgence.

- La fonctionnalité optionnelle **Télésurveillance** permet au médecin de surveiller le patient en ligne.

- La fonctionnalité obligatoire **consulter liste rendez-vous** permet au médecin de consulter la liste des rendez-vous pris par les patients.

2.5.1.2 Feature model « Patient »

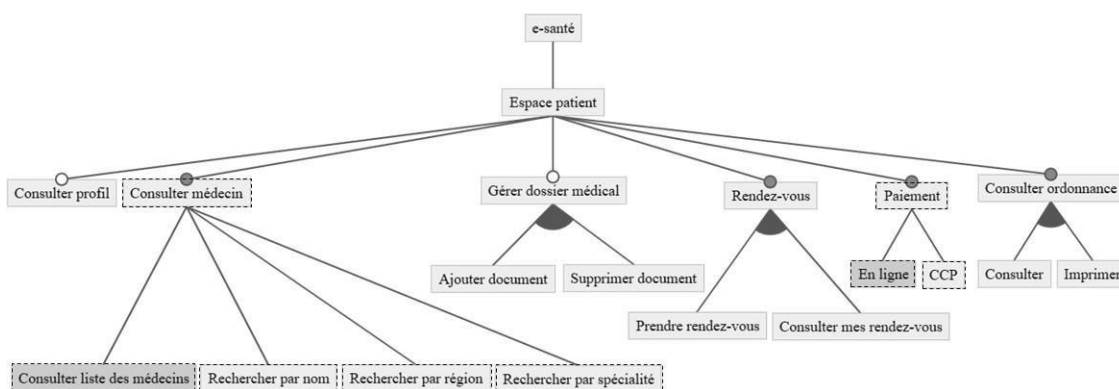


FIGURE 2.6 – Feature model patient

Les contraintes :

-Espace patient **require** s'authentifier.

Espace patient :

- La fonctionnalité optionnelle **Consulter profil** permet au patient de consulter ses coordon-

nées.

- La fonctionnalité obligatoire **Consulter médecin** implique trois autres fonctionnalités : **Consulter liste des médecins** (permet au patient de consulter la liste des médecins), **Rechercher par région** (permet au patient de rechercher un médecin par sa région), **Rechercher par nom** (permet au patient de rechercher un médecin par son nom), **Rechercher par spécialité** (permet au patient de rechercher un médecin par sa spécialité).

- La fonctionnalité optionnelle **Gérer dossier médical** implique deux autres fonctionnalités : **Ajouter document** (permet au patient d'ajouter un document à son dossier médical), **Supprimer document** (permet au patient de supprimer un document de son dossier médical).

- La fonctionnalité obligatoire **Rendez-vous** implique deux autres fonctionnalités : **Prendre rendez-vous** (permet au patient de prendre un rendez-vous), **Consulter mes rendez-vous** (permet au patient de consulter ses rendez-vous).

- La fonctionnalité obligatoire **Paiement** implique deux autres fonctionnalités : **En ligne** (permet au patient de payer en ligne), **CCP** (permet au patient de faire un paiement par compte CCP).

- La fonctionnalité obligatoire **Consulter ordonnance** implique deux autres fonctionnalités : **Consulter** (permet au patient de consulter son ordonnance), **Imprimer** (permet au patient d'imprimer son ordonnance).

2.5.1.3 Feature model « technique »

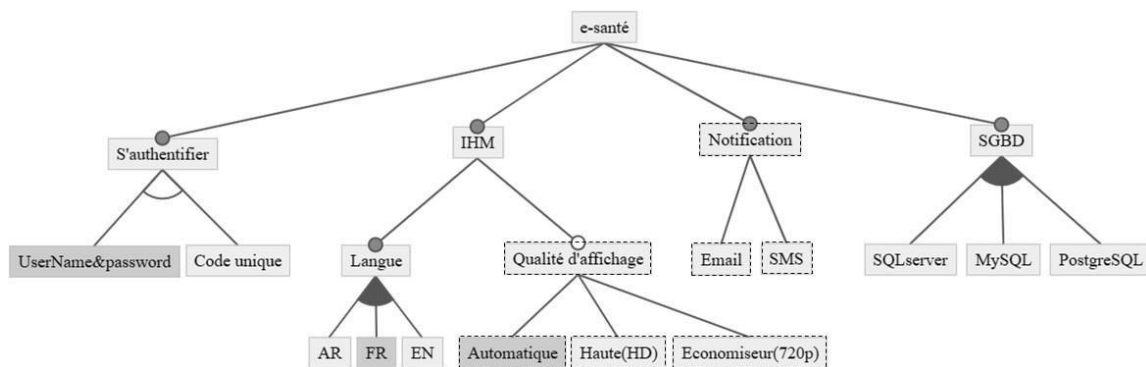


FIGURE 2.7 – Feature model technique

Les contraintes :

- Code unique **exclude** UserName & password.
- SMS **exclude** Email.

Feature model technique :

- La fonctionnalité obligatoire **IHM** implique deux autres fonctionnalités : la fonctionnalité

obligatoire **Langue** (permet de changer la langue d’affichage) et la fonctionnalité optionnelle **Qualité d’affichage** qui implique soit qualité **Automatique**, soit **Haute**, soit **Economiseur**.

- La fonctionnalité obligatoire **S’authentifier** permet aux utilisateurs de s’authentifier soit avec un **UserName password** soit avec un **Code unique**.

- La fonctionnalité obligatoire **Notification** implique deux autres fonctionnalités : **Email** (permet de notifier les utilisateurs du système par email), **SMS** (permet de notifier les utilisateurs du système par SMS).

- La fonctionnalité obligatoire **SGBD** implique soit l’utilisation du **SQLserver** ou bien **MySQL** ou bien **PostgreSQL**.

TABLE 2.3 – Exécution des features

Features	Vitesse de connexion<2mb/s	2mb/s<=Vitesse de connexion<8mb/s
Gestion ordonnance	Télécharger	Consulter en ligne
Consultation	-Audio -SMS	Vidéo
Gestion médecin	Consulter liste des médecins	-Rechercher par région -Rechercher par nom -Rechercher par spécialité
Paiement	CCP	En ligne
Qualité d’affichage	Economiseur	-Automatique -haute
Notification	SMS	Email

2.6 Mapping Feature-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. Protégé est utilisé pour construire notre ontologie.

2.6.1 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. La figure représente ces différents types de fonctionnalités.

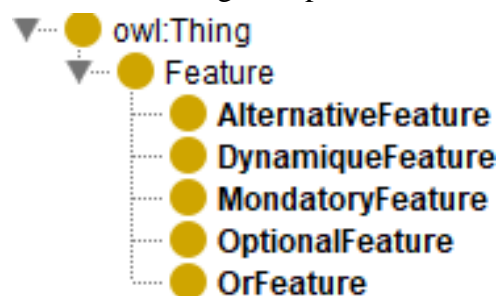


FIGURE 2.8 – Les classes de l’ontologie proposée

2.6.2 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 les relations suivantes : **require**, qui exprime la dépendance entre deux fonctionnalités, symétriquement ou non, **exclude**, qui signifie l’exclusion d’une deuxième caractéristique en sélectionnant la première et **Adapt**. La figure montre ces différentes propriétés.

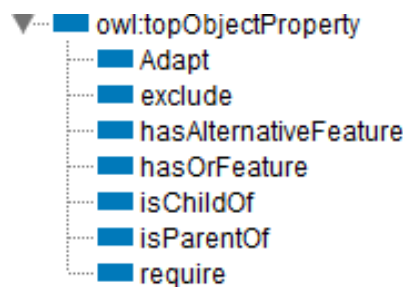


FIGURE 2.9 – Les propriétés de l’ontologie proposée

2.6.3 Représentation des fonctionnalités dans l’ontologie :

Les fonctionnalités de notre LDP sont représentées par des instances des concepts de l’ontologie, et les relations **isParentOf** et **isChildOf** sont utilisées pour construire la hiérarchie de Feature Model comme illustré dans la Figure 2.10.



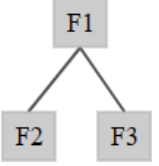
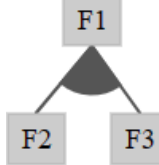
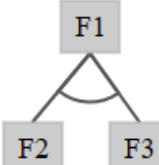
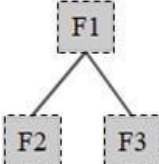
FIGURE 2.10 – La hiérarchie des fonctionnalités

2.6.4 Les règles de passage :

Les principales transformations sont définies et illustrées dans le tableau.

TABLE 2.4 – Passage du Feature model à l'ontologie.

Feature model	Signification	Ontologie
	Root feature F	F : Feature
	Mandatory feature (Fonctionnalité obligatoire)	F : Mandatory feature
	Optional Feature (Fonctionnalité Optionnelle)	F : OptionalFeature
	Dynamique Feature (Fonctionnalité dynamique)	F : DynamiqueFeature

	<p>F1 est le parent de F2 et de F3</p>	<p>isParentOf(F1, F2) isParentOf(F1, F3)</p>
	<p>Or décomposition</p>	<p>F2 :OrFeature F3 :OrFeature hasOrFeature(F1,F2) hasOrFeature(F1, F3)</p>
	<p>Xor décomposition</p>	<p>F2 :AlternativeFeature F3 :AlternativeFeature hasAlternativeFeature(F1, F2) hasAlternativeFeature(F1, F3)</p>
	<p>Adapt</p>	<p>F1 :DynamiqueFeature F2 :DynamiqueFeature F3 :DynamiqueFeature isParentOf(F1,F2) isParentOf(F1,F3)</p>

2.7 Conception architecturale :

La figure suivante montre l'architecture globale de l'e-health.

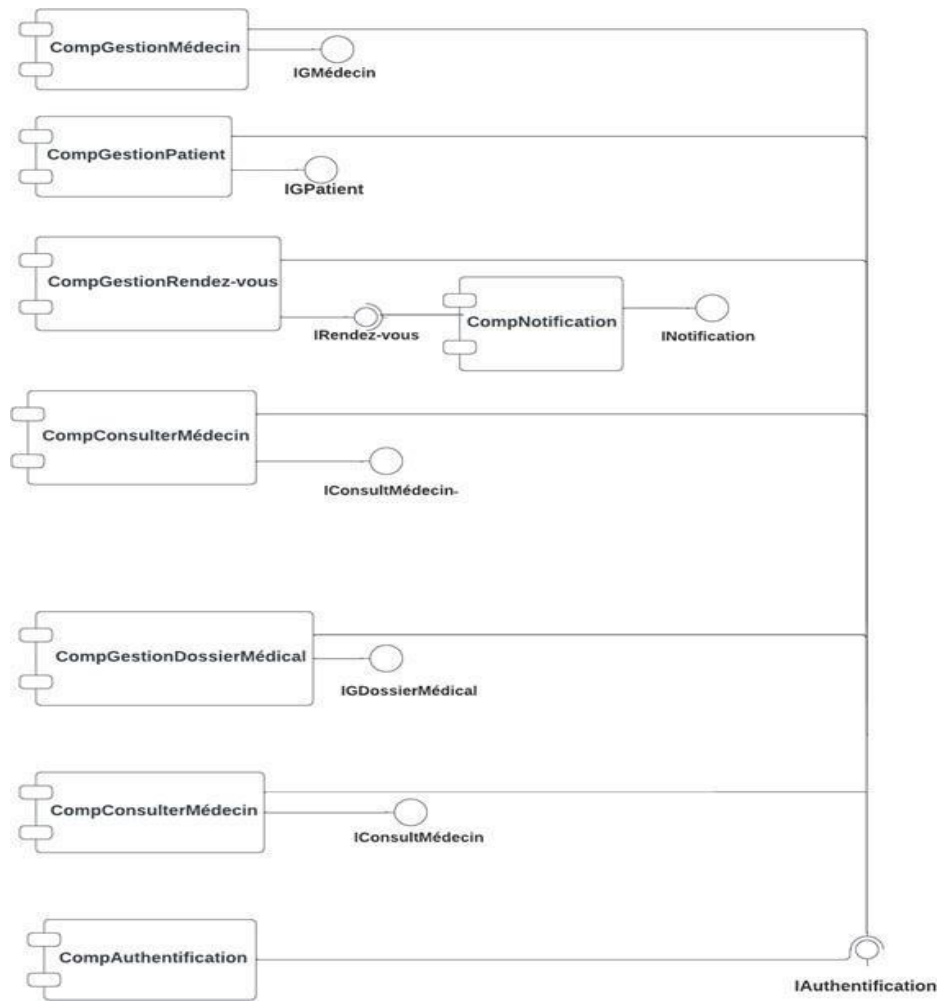


FIGURE 2.11 – Diagramme des composants global

2.7.1 Raffinement des composants d'e-health :

Après avoir défini l'architecture globale de l'e health, nous allons présenter par la suite quelques sous composants composites pour donner un aperçu de leurs structure.

2.7.1.1 Raffinement du Composant Gestion Patient :

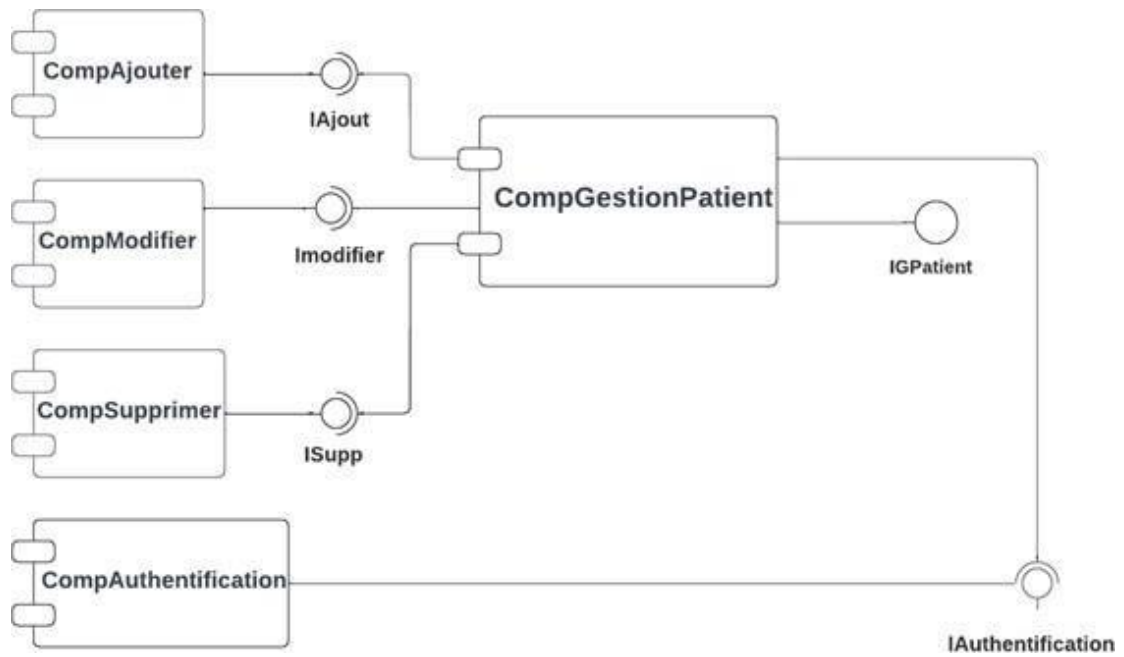


FIGURE 2.12 – Composant Gestion Patient

2.7.1.2 Raffinement du Composant Gestion dossier médical :

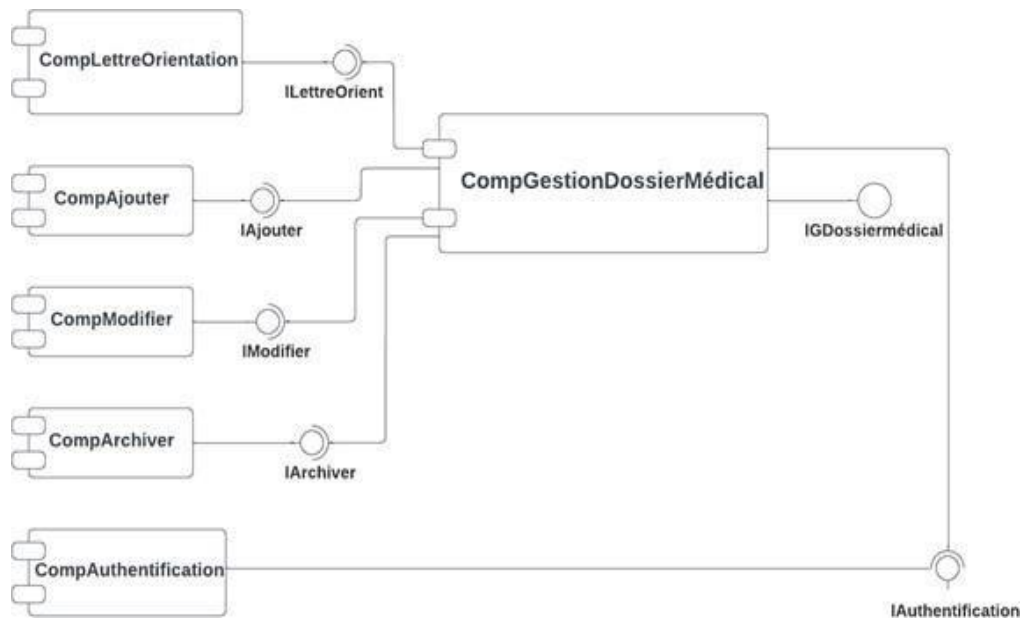


FIGURE 2.13 – Composant Gestion Dossier Médical

2.8 Diagramme de classe :

Le diagramme de classes montre la structure interne du système. Après l'analyse précédente, nous avons obtenu le diagramme de classes, comme le montre la Figure suivante :

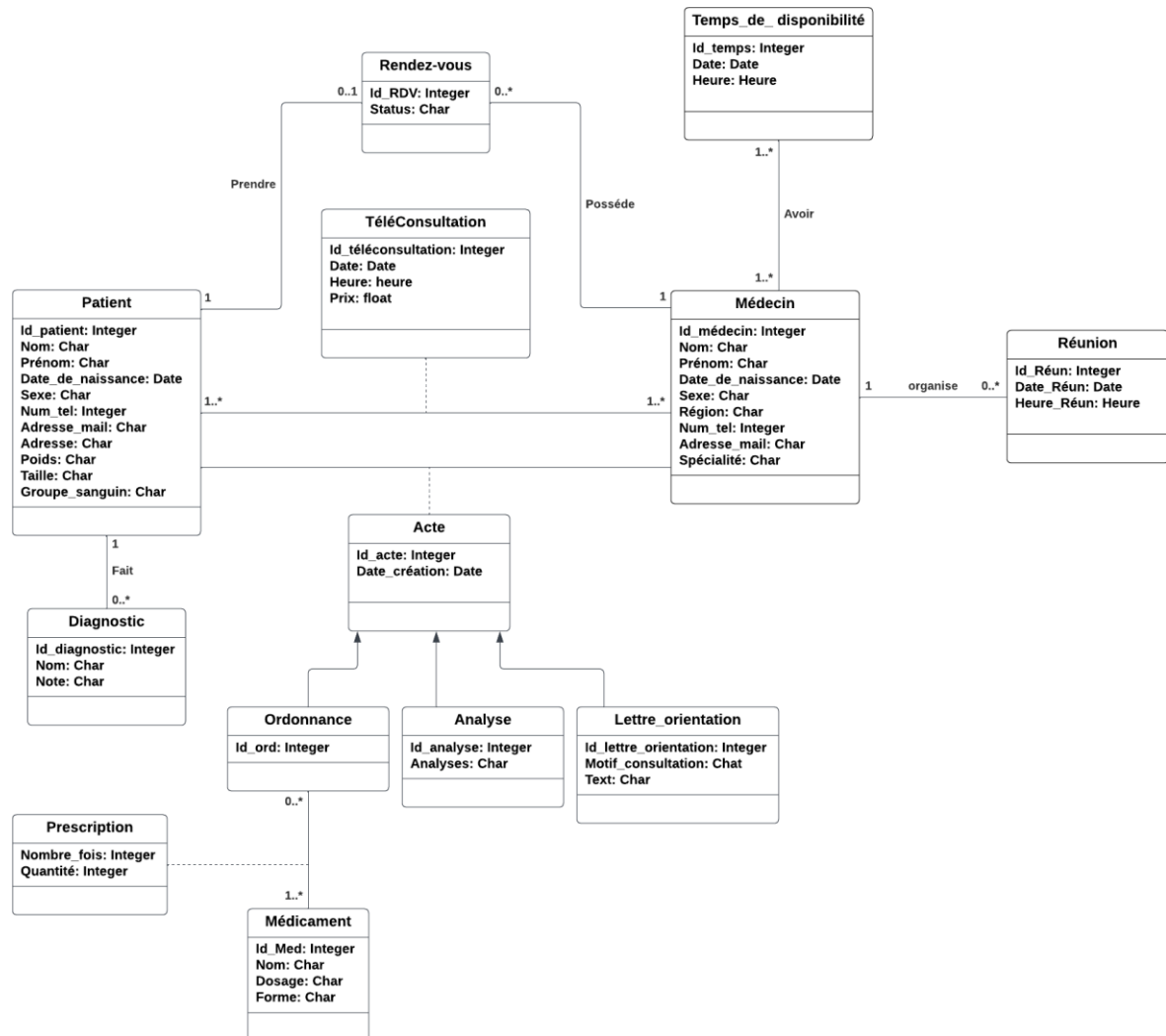


FIGURE 2.14 – Diagramme de classe

2.9 Conclusion

Durant ce chapitre, nous avons présenté l'étape analyse et conception des besoins, dont nous avons décrit d'une façon détaillée la modélisation en se basant sur les diagrammes du langage UML à savoir le diagramme de cas d'utilisation, le diagramme de classes et le diagramme de composants. Ainsi le rôle de l'ontologie dans la représentation de notre feature model, ce qui facilite la réalisation concrète de l'application qui fera l'objet du chapitre suivant.

Chapitre 3

Ingénierie d'application

3.1 Introduction

Dans le chapitre précédent, nous avons présenté la conception de notre solution. Dans ce chapitre nous allons présenter les ressources logicielles utilisées ainsi que la réalisation de l'application en utilisant notre ligne de produits.

3.2 Environnement logistique

3.2.1 Environnement de développement

3.2.1.1 Technologies de développement

✓ Python

Python est le langage de programmation open source le plus employé par les informaticiens. Ce langage s'est propulsé en tête de la gestion d'infrastructure, d'analyse de données ou dans le domaine du développement de logiciels.

✓ Django

Django est un framework Python de haut niveau, permettant un développement rapide de sites internet, sécurisés, et maintenables. Django s'inspire du principe MVC(Model, View, Controler).

✓ HTML

HyperText Markup Language (HTML), désigne un type de langage informatique descriptif. Il s'agit plus précisément d'un format de données utilisé dans l'univers d'Internet pour la mise en forme des pages Web. Il permet de réaliser l'hypertexte à base d'une structure de balisage.

✓ CSS

CSS est l'acronyme de « *Cascading Style Sheets* » ce qui signifie « feuille de style en cascade ». Le CSS correspond à un langage informatique permettant de mettre en forme des pages web (HTML ou XML).

✓ JavaScript

JavaScript est un langage de programmation qui permet de créer du contenu mis à jour de façon dynamique, de contrôler le contenu multimédia, d'animer des images, et tout ce à quoi on peut penser.

✓ **JQuery**

jQuery, est une bibliothèque JavaScript gratuite, libre et multiplateforme. Compatible avec l'ensemble des navigateurs Web (Internet Explorer, Safari, Chrome, Firefox, etc.), elle a été conçue et développée en 2006 pour faciliter l'écriture de scripts.

✓ **SQL**

SQL (structured Query Language) est un langage de « programmation » standardisé qui est utilisé pour gérer des bases de données relationnelles et effectuer diverses opérations sur les données qu'elles contiennent.

3.2.1.2 Outils de programmation

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

✓ **Visual Studio Code**



Visual Studio Code est un éditeur de code open-source développé par Microsoft supportant un très grand nombre de langages grâce à des extensions. Il supporte l'autocomplétion, la coloration syntaxique, le débogage, et les commandes git.

✓ **PostgreSQL**



PostgreSQL est un système de gestion de base de données relationnelle orienté objet puissant et open source qui est capable de prendre en charge en toute sécurité les charges de travail de données les plus complexes.

✓ **Protégé**



PROTEGE est une interface modulaire, développée au Stanford Medical Informatiques de l'Université de Stanford⁷, permettant l'édition, la visualisation, le contrôle, l'extraction à partir de sources textuelles, et la fusion semi-automatique d'ontologies.

PROTEGE autorise la définition de méta-classes, dont les instances sont des classes, ce qui permet de créer son propre modèle de connaissances avant de bâtir une ontologie. De nombreux plugins sont disponibles ou peuvent être ajoutés par l'utilisateur.

3.3 Dérivation d'une application

3.3.1 Modélisation d'application

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

3.3.1.1 Feature Model d'application Espace Médecin

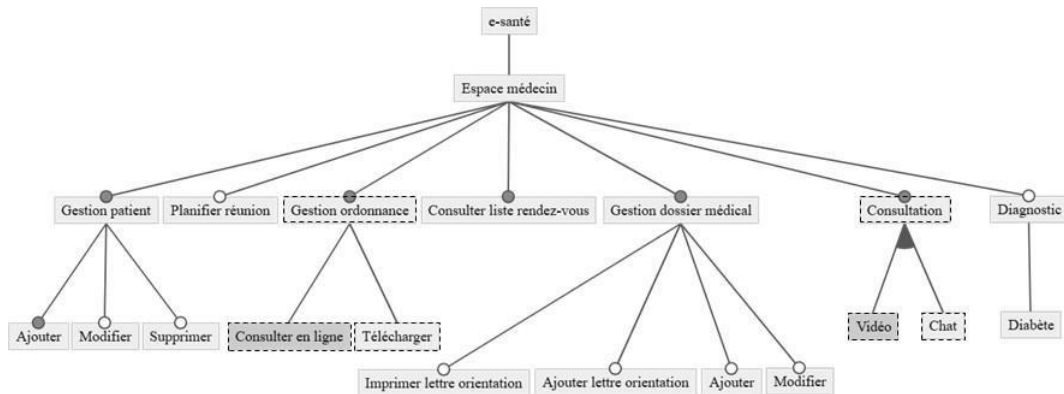


FIGURE 3.1 – Feature model application Espace médecin

Les contraintes :

- Espace médecin **require** s'authentifier.
- Planifier réunion **require** consulter liste rendez-vous.

3.3.1.2 Feature Model d'application Espace Patient

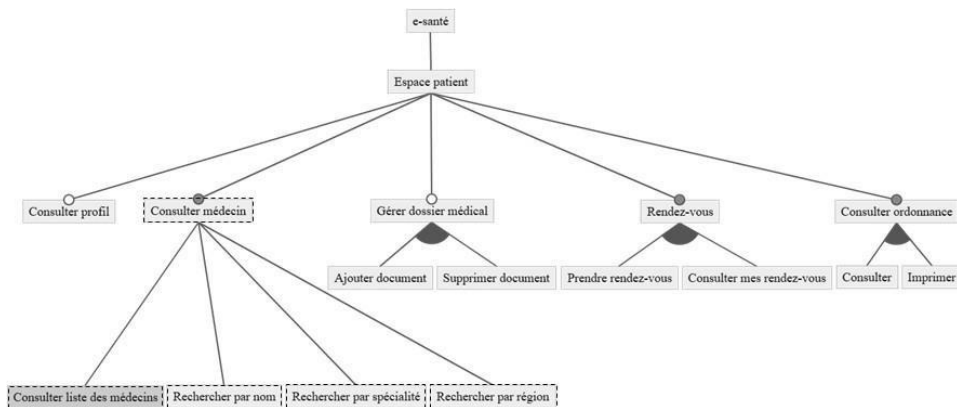


FIGURE 3.2 – Feature model application Espace patient

Les contraintes :

-Espace patient **require** s'authentifier.

3.3.1.3 Feature Model d'application Technique

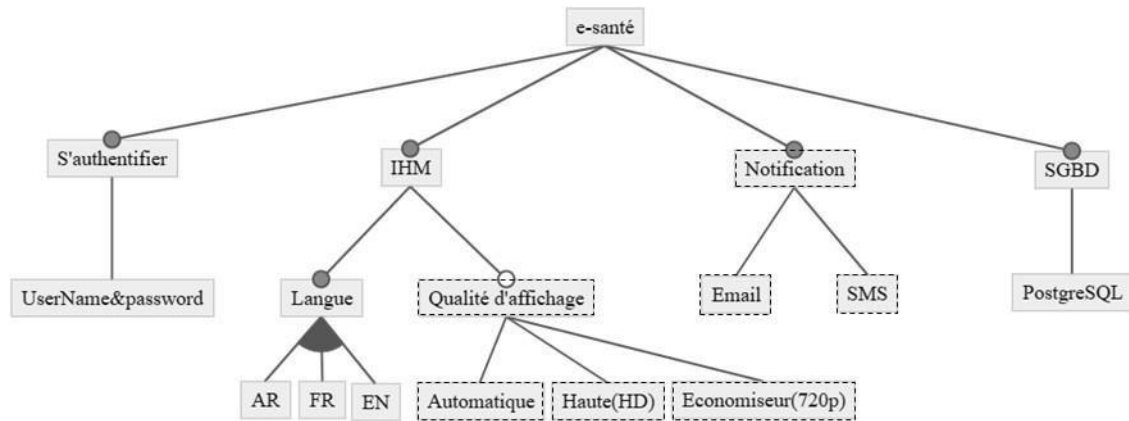


FIGURE 3.3 – Feature model application technique

Les contraintes :

-SMS **exclude** Email.

3.3.2 Réalisation de l'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 Python qui permet la création d'applications web.

Pour le stockage des données on a choisi PostgreSQL. Pour exécuter des applications Web Python, nous avons choisi d'utiliser le framework Django car c'est un serveur web léger permettant de développer et tester ses applications en temps réel sans déploiement.

3.3.3 Les interfaces

3.3.3.1 Page d'accueil

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.



POUR LES MÉDECINS

Langue ▾

Pour les patients

Se connecter |

Créer un compte

Consultation en-ligne

Trouver un médecin et prendre un rendez-vous.

ou

ou


LA TÉLÉCONSULTATION SIMPLE, RAPIDE ET EFFICACE !



Consultation



Prise de RDV en-ligne



Ordonnance en-ligne

FIGURE 3.4 – Page d'accueil

3.3.3.2 Espace Médecin

Si le visiteur n'est pas déjà inscrit, il peut créer un compte en cliquant sur "Inscrire" trouvé dans la page d'accueil. Ce lien va le rediriger vers l'interface d'inscription où il doit introduire ses informations comme le montre la Figure 3.5. Il doit également préciser s'il souhaite rejoindre en tant que Médecin ou en tant que patient. Sinon s'il a déjà un compte il peut se connecter depuis l'interface de connexion qui est illustrée dans la Figure 3.6.

Vous êtes qui?!

<p>je suis Patient</p> <p>Si vous êtes un patient,inscrivez vous ici.</p> <p>Inscrire</p>	<p>je suis médecin</p> <p>si vous êtes un médecin ,inscrivez vous ici.</p> <p>Inscrire</p>
---	--

FIGURE 3.5 – Page d'inscription



Se connecter

si vous n'avez pas un compte!! [Créer un compte](#)

Nom d'utilisateur:

Mot de pass

Continuer

FIGURE 3.6 – Page de connexion

Dès que le médecin accède à son espace il peut ajouter des ordonnances, des analyses, ou bien des lettres d'orientation en cas d'urgence comme montrent les figures 3.7 3.8 3.9.

Mes patients les rendez-vous GRITLI ▾

Ajouter Ordonnance

Date

Patient

Médecin

Ajouter médicaments

Médicament	Nombre de fois	Quantité	Supprimer?
<input type="text" value="....."/>	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="checkbox"/>

Ajouter plus

FIGURE 3.7 – Page d'ajout d'ordonnance

Ajouter des analyses

Date et Heure :

Médecin :

Patient :

Autres:

[ajouter](#)

faire:

- Groupe sanguin
- Créatinine
- Natrémie(Na+)
- FNS avec équilibre leucocytaire
- Acide urique
- Kaliémie
- Glycémie à jeun
- Bilirubine:Totale-conjuguée-libre
- Hémoglobine glyquée(Hba1c)
- Transaminases(ASAT,ALAT)
- HGPO
- CPK
- Calcémie
- Phosphorémie
- ECEBU
- Cholestérol total
- Phosphatases alcalines
- TSH
- HDL Cholestérol
- Taux de prothrombine(TP)
- Chimie des urines
- LDL Cholestérol
- TCK-INR
- Protéinurie des 24H
- Triglycérides
- VS,CRP,Fibrinogene
- FT3-FT4
- Urée sanguine
- Fer sérique
- Microalbuminurie

FIGURE 3.8 – Page d'ajout d'analyses

Ajouter une lettre d'orientation

Date de création

Patient

Médecin

Motif consultation

Texte

Cher confrère (Chère consœur) Permettez moi de vous adresser le (la) patient(e) sus-nommé(e), sus-âgé(e) qui s'est présenté au PVM (ou PUC) pour "motif de consultation", dont l'examen clinique retrouve "text", je vous le confie pour une meilleure PEC confraternellement.

[ajouter](#)



FIGURE 3.9 – Page d'ajout d'une lettre d'orientation

Le médecin peut aussi voir ses information en cliquant sur son nom et modifier ses informations en cliquant sur le lien "Editer le profil".

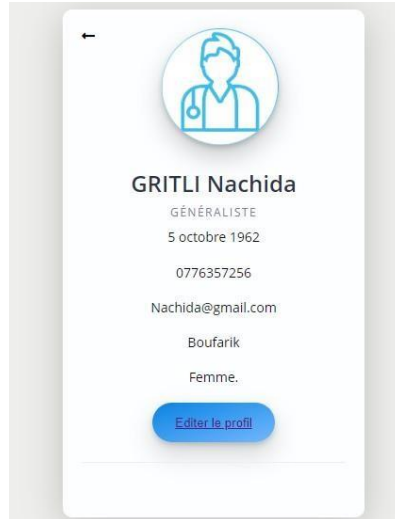


FIGURE 3.10 – Profil médecin

3.3.3.3 Espace Patient

Dès que le patient accède à son espace, il peut consulter la liste des médecins où il peut rechercher un médecin par son nom, sa spécialité ou sa région. Il peut voir aussi le profil du médecin et prendre un rendez-vous comme montrent les figures 3.11 et 3.12.

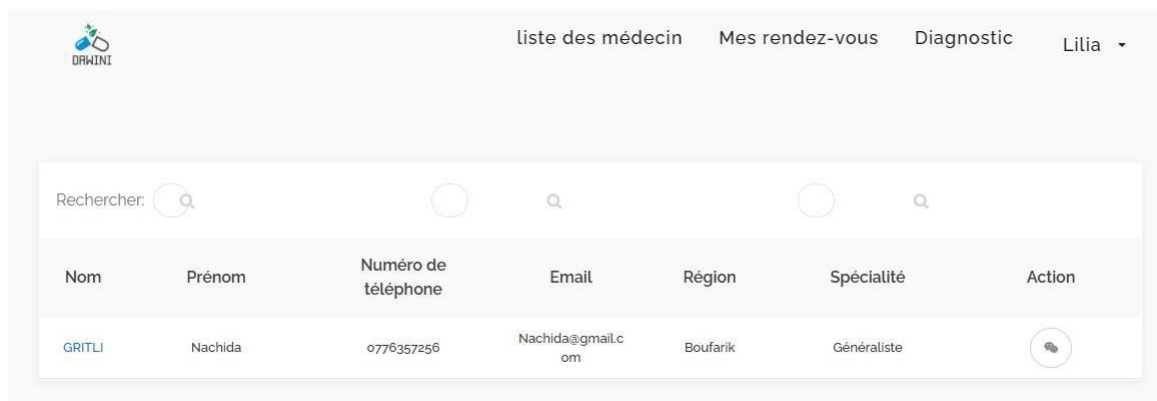


FIGURE 3.11 – Liste médecin

FIGURE 3.12 – Page prise de rendez-vous

La figure 3.13 montre la liste des rendez-vous du patient où il peut faire son rendez-vous via des messages ou un appel vidéo en utilisant google meet.

Date	Médecin	Téléphone	Email	Status	Action
Jun 07 2023-09:00:00 - GRITLI	GRITLI Nachida	0776357256	Nachida@gmail.com	Confirmé	

FIGURE 3.13 – Liste rendez-vous

La figure 3.14 montre l’interface du chat entre le médecin et le patient.

FIGURE 3.14 – Chat

La figure 3.15 montre l’interface où le patient peut faire un appel vidéo avec un médecin.

Date	Heure	Médecin	Lien
7 juin 2023	09:00	GRITLI	https://meet.google.com/hyri-P8fLvh5WA

FIGURE 3.15 – Appel vidéo

Le patient peut aussi consulter ses informations personnelles les modifier, consulter ses ordonnances, ses analyses et ses lettres d'orientation comme montrent les figures 3.16 3.17 3.18 3.19.



FIGURE 3.16 – Profil patient

Date	Patient	Médecin	les médicaments	Action
6 juin 2023 19:11	Aitsaadi	GRITLI	Parasétamol nombre de fois:3 quantité :1	 

FIGURE 3.17 – Liste des ordonnances d’un patient

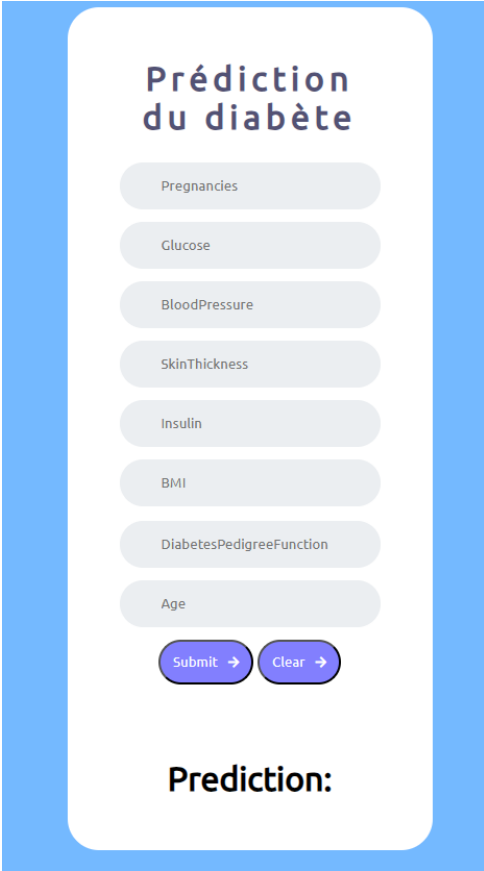
Date	Patient	Médecin	A faire	Action
	Aitsaadi	GRITLI	['GS', 'Acide urique', 'Glycémie', 'Calcémie', 'TSH', 'Urée sanguine', 'Fer sérique']	

FIGURE 3.18 – Liste des analyses d’un patient

Date	Patient	Médecin	les médicaments	Action
6 juin 2023 19:16	Aitsaadi	GRITLI	Cher confrère (Chère consœur) Permettez moi de vous adresser le (la) patient(e) sus-nommé(e), sus-âgé(e) qui s'est présenté au PVM (ou PUC) pour urgence , dont l'examen clinique retrouve , je vous le confie pour une meilleure PEC confraternellement.	

FIGURE 3.19 – Liste des lettres d’orientation d’un patient

La figure 3.17 montre les diagnostics que peut faire le patient sur le diabète.



The image shows a mobile application interface titled "Prédiction du diabète". It features a list of input fields for various health metrics: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, and Age. Below these fields are two buttons: "Submit" and "Clear", both with right-pointing arrows. At the bottom of the interface, the text "Prediction:" is displayed.

FIGURE 3.20 – Diagnostic

3.4 Conclusion

Dans ce chapitre, nous avons présenté le deuxième processus de développement de la LDP "ingénierie d'application". Dans notre projet de fin d'études, nous avons retenu le cas d'une application e-health. Nous précisons que ce cas d'études a été validé par Docteur GRITLI Nachida, docteur généraliste.

Conclusion et perspectives

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

Grâce à une ligne de produits dynamique, les plateformes Healthcare auto-adaptatives peuvent s'adapter rapidement aux changements technologiques, aux évolutions réglementaires et aux besoins des utilisateurs. Cela permet d'améliorer la flexibilité, la réactivité et la pérennité des systèmes de santé, tout en réduisant les coûts et les délais de développement. 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-health. Ce processus est divisé en deux sous processus : l'ingénierie de domaine et l'ingénierie d'application.

Dans le premier processus, nous avons commencé par la conception et la construction de l'ontologie pour déterminer les features à inclure dans le feature model, après l'implémentation des composants logiciels réutilisables. 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 Python. 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.

Bibliographie

- [1] C.Elsner. Automating staged product derivation for heterogeneous multi-product lines.phd dissertation, universität erlangen-nürnberg, [http ://www.opus.ub.uni-erlangen.de/opus/volltexte/2012/3265/pdf/christophelsnerdissertation.pdf](http://www.opus.ub.uni-erlangen.de/opus/volltexte/2012/3265/pdf/christophelsnerdissertation.pdf). 2012.
- [2] Krzysztof Czarnecki and Ulrich Eisenecker. Generative programming : Methods, tools, and applications, addisonwesley professional. pages 82–130, 2000.
- [3] Weyns D., Iftikhar M.U., Malek S., and Andersson J. Claims and supporting evidence for self-adaptive systems- a literature study. pages 89–98. SEMMS'12, Proceedings of the 7th International Symposium on Software Engineering for adaptive and Self-Managing Systems, 2012.
- [4] Lohmann D. Schröder-Preikschat W. Elsner, C. Product derivation for solutiondriven product line engineering. pages 35–41, 2009.
- [5] Autonomic Computing et al. An architectural blueprint for autonomic computing. pages 1–6. In : IBM white Paper 31, 2006.
- [6] A. GÓMEZ-PÉREZ et R. BENJAMINS. « overview of knowledge sharing and reuse components : Ontologies and problem-solving methods ». page adresse : [https ://oa.upm.es/6468/](https://oa.upm.es/6468/). 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.
- [7] F.Chauvel. *Méthodes et outils pour la conception de systèmes logiciels auto-adaptatifs*. 2008.
- [8] I FOUJIL. L'expérience de la télémédecine en algérie : état des lieux et prescriptives cas de chu de tizi-ouzou[mémoire master, université ummto]. 2016/2017.

- [9] Hinchey M. Park S. Schmid K. Hallsteinsen, S. Dynamic software product lines. pages 93–95. IEEE Computer 41(4), 2008.
- [10] Park S. Schmid K. Hinchey, M. Building dynamic software product lines. pages 22–26. IEEE Computer 45(10), 2012.
- [11] Bosch J. Design and use of software architectures. adopting and evolving a product line approach. 2000.
- [12] Rafael Capilla Jan Bosch and Rich Hilliard. Trends in systems and software variability. page 44–51. IEEE Software, 32(3), may 2015.
- [13] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute (SEI), November 1990.
- [14] Kyo C. Kang and Hyesun Lee. Variability modeling. in rafael capilla, jan bosch, and kyochul kang, editors, systems and software variability management. page 25–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [15] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. pages 41–50. In : Computer 36.1, 2003.
- [16] K.Pohl, G.Bockle, and F.V.D.Linden. *Software product-Line Engineering : Foundation, Principles, and Techniques*. Birkhauser, 2005.
- [17] Luciano Baresi Lero Liliana Pasquale, Sam Guinea. *Service-Oriented Dynamic Software Product Lines with DyBPEL*.
- [18] Jindu Liu Liwei Shen, Xin Peng and Wenyun Zhao. *Towards Feature-oriented Variability Reconfiguration in Dynamic Software Product Lines*.
- [19] Wenyun Zhao Liwei Shen, Xin Peng. *Software Product Line Engineering for Developing Self-adaptive Systems : Towards the domain Requirements*.
- [20] Salehie M. and Tahvildari L. Autonomic computing :emerging trends and open problems. pages 1–7. ACM SIGSOFT Software Engineering, vol.30, 2005.
- [21] Oreizy P., Madvidovic N., and Taylor R.N. Architecture based runtime software evolution.dans proceedings of the 20th international conference on software engineering. pages 177–186. IEEE computer Society, 1998.
- [22] Joaquin Pena Pablo Trinidad, Antonio Ruiz-Cortés and David Benavides. *Mapping Feature Models onto Component Models to Build Dynamic Software Product Lines*.

- [23] D.L. Parnas. On the design and development of program families, *iee transactions on software engineering*, se2(1) :1–9. March 1976.
- [24] P.Clements and L.Northrop. *Software product-Line Engineering :Practices and Patterns*. 1er ed.Addison-Wesley, 2001.
- [25] Karin Fetzer Mira Mezini Tom Dinkelaker, Ralf Litschke. *A dynamic Software Product Line Approach using Aspect Models at Runtime*.
- [26] T.Ziadi, H elouet, Loic, J ez equel, and Jean-Marc. Towards a UML Profile for Software Product Line. pages 129–139, 2004. PFE 2003, LNCS 3014.
- [27] Schmid K. Rommes E. Van der Linden, F. *Software Product Lines in Action.The Best Industrial Practice in Product Line Engineering*. 2007.
- [28] T. Ziadi. *Les Lignes de Produits Logiciels (Software Product Lines)*, UPMC/LIP6. 2013.
- [29] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199-220, 1993.
- [30] John F Sowa. International Conference on conceptual structures, 55-81, 2000.